CIS 205 Study Guide
Fall 2014

Don Colton
Brigham Young University–Hawai‘i

December 9, 2014

This is a study guide for the CIS 205 class, Discrete Mathematics I, taught by Don Colton, Fall 2014, at Brigham Young University–Hawai‘i.

This study guide is focused directly on the grading of the course, as taught by Don Colton. Specifically, the exams are explained and the skills they measure are taught.

Extra credit is often given to students who report errors of any kind, or improvements that can be made. This applies to the study guide.

Contents

1 Syllabus 4

2 DCQuiz: My Learning Management System 36

3 GradeBot 43

4 Programs Assigned 54

5 QuizGen 77

6 Skills Tests 78

7 Tutorial on Formal Logic 81

8 S1: Resolution 89

9 S2: Big Oh Analysis 96

10 Tutorial on Sets 106

11 S3: Counting 113

12 Tutorial on Discrete Probability 120

13 S4: Conditional Probability 122

14 S5: BST: Binary Search Trees **138**

15 S6: Huffman Coding **144**

16 S7: MST: Minimum Spanning Trees **148**

A Spelling Errors **153**

B Test Bank **155**

Index **157**

Chapter 1

Syllabus

The original, separate syllabus is the official version. This is a copy of that syllabus, and is provided for your convenience and as a place for me to correct minor errors such as spelling mistakes.

Contents

1.1 Overview **5**
1.1.1 So, What *is* Discrete Math? 6
1.1.2 Expected Proficiencies 6
1.2 Course and Faculty **7**
1.2.1 Course Information 7
1.2.2 Faculty Information 7
1.2.3 Course Readings and Materials 8
1.3 Calendar **8**
1.4 Grading **9**
1.4.1 Tracking Your Grade 10
1.4.2 Daily Update (35 points) 10
1.4.3 Daily Quiz (85 points) 11
1.4.4 Readings (100 points) 11
1.4.5 Study Time (80 points) 12
1.4.6 Some Points are Optional 13
1.4.7 Skill: (300 points) GradeBot 13
1.4.8 Do Your Own Work 14
1.4.9 Skill: Exams (400 pts + 35 ec) 15
1.4.10 Other Extra Credit 16

1.5 Instructional Methods 16
1.5.1 BYUH Learning Framework 17
1.5.2 Support 18
1.6 Course Policies 19
1.6.1 Excused Absences 20
1.6.2 Reasonable Accommodation 20
1.6.3 Communication 21
1.7 Learning Outcomes 23
1.7.1 ILOs: Institutional Outcomes 23
1.7.2 PLOs: Program Outcomes 24
1.7.3 CLOs: Course Outcomes 24
1.8 General Topics 28
1.8.1 Academic Integrity 28
1.8.2 Sexual Misconduct 30
1.8.3 Dress and Grooming Standards 31
1.8.4 Accommodating Special Needs 32
1.9 Syllabus Summary 33

1.1 Overview

Programming is the art of building up those simple things that computers can do into those fun but complicated things we want them to do.

Part of this involves certain "tricks of the trade," so to speak. Much as the human body has to fire certain nerve cells in order to make an arm pick up a pencil, the computer has to carry out certain primitive instructions to make the desired result happen.

The linkage between nerves firing and arms moving may not always be obvious.

The good news is that those "tricks of the trade" are kind of fun to learn and use. The scary part is they were discovered and developed by mathematicians originally, so they are considered to be a branch of Mathematics. But really, they belong to Computer Scientists. Like us.

Discrete Mathematics 1 introduces many of the underlying mathematical principles used by computer scientists. While the usefulness may not always be immediately apparent, these concepts will be integral to our understanding of the principles of computing. Where possible, we will discuss the immediate application.

Although the word Mathematics is in the course title, this is basically a computer science course. Mathematical concepts are the focus of study, but they are reinforced through programming projects and exams. Other activities may also be assigned to support those concepts that are not otherwise adequately reinforced or measured.

1.1.1 So, What *is* Discrete Math?

Discrete means chunks, as opposed to Calculus, which is continuous.

With discrete, we are dealing with things like the natural numbers: 1, 2, 3, 4, 5, and so on. We are not dealing with things like $\pi = 3.1415...$ and $e = 2.71828...$ and natural logarithms.

So, it's whole things. Things that are either totally present or totally absent. Ones and zeros. True and false. Six sides on dice. Two sides on coins. Branches in trees. Nodes and edges in graphs.

It's kind of fun. Like solving a puzzle.

1.1.2 Expected Proficiencies

I expect that you can already program in a language that I support for this class. The supported languages are: C, C++, Java, Perl, Python, Ruby, and Tcl. If you only know another language, such as Visual Basic, that will be helpful, but you will need to learn one of the supported languages well enough to do your assignments for this class.

You should already be able to use variables, arrays, decisions, and loops. Ideally you should be able to use subroutines. And if you have done recursion already, that would be truly awesome.

If you pass CIS 101 at BYUH with a grade of C or better, you probably have the required level of programming skill.

1.2 Course and Faculty

1.2.1 Course Information

- **Title:** Discrete Mathematics I
- **Course Number:** CIS 205
- **Course Description:** (from the catalog) Functions, relations, and sets; basic logic; proof techniques; basics of counting.
- **Prerequisites:** CIS 101
- **Semester/Year:** Fall 2014
- **Semester Code:** 2145
- **Meeting Time:** MWF 08:40 to 09:40
- **Location:** GCB 140
- **First Day of Instruction:** Mon, Sep 08
- **Last Day to Withdraw:** Fri, Oct 31
- **Last Day for Late Work:** Mon, Dec 08
- **Last Day of Instruction:** Mon, Dec 08
- **Final Exam:** Wed, Dec 10, 07:00 to 09:50

1.2.2 Faculty Information

- **Instructor:** Don Colton
- **Office Location:** GCB 128
- **Office Hours:** MWF 14:30 to 15:30, GCB 111
- **Email:** doncolton2@gmail.com
- **Campus Homepage:**
  http://byuh.doncolton.com/ is my campus homepage. It has my calendar and links to the homepages for each of my classes.
- **Off-Campus Homepage:**
  http://doncolton.com/ is my off-campus homepage.

I have reserved GCB 111 on MWF 14:30 to 15:30 so my CIS 205 students (and others) can study in a lab setting and meet with me and each other. I allow the room as an Open Lab for your use either individually or in groups, for my class or for other classes. MWF 14:30 to 15:30 I will be present in GCB 111 or in my office to assist students that come.

1.2.3 Course Readings and Materials

- **Textbook (Rental):** Mathematical Structures for Computer Science (6th Edition), by Judith L. Gersting. ISBN 071676864X.
- **Learning Management System:** `https://dcquiz.byuh.edu/` is the learning management system for my courses.
- **Course Homepage:** `http://byuh.doncolton.com/cis205/` is my course homepage. It has links to many things including the syllabus, study guide, and textbook.
- **Study Guide:** `http://byuh.doncolton.com/cis205/2145/sguide.pdf` is the study guide for this course. It includes an indexed copy of some or all of this syllabus. The study guide is updated frequently throughout the semester as assignments are made and deadlines are established or updated.

1.3 Calendar

Mo	Sep	08	syllabus, 56 **p01** (20p) Factors
We	Sep	10	57 **p02** (20p) Perfect Numbers
Fr	Sep	12	textbooks, formal logic
Mo	Sep	15	G1.1 (001-013) Symb, Taut; Guilty
We	Sep	17	G1.2 (021-031) Propositional Logic
Fr	Sep	19	89 **S1**v1 q41 Prop Calc (40m)
Mo	Sep	22	G2.1 (088-098) Proofs, 58 **p03** (30p)
We	Sep	24	G2.2 (100-112) Induc, 60 **p04** (20p)
Fr	Sep	26	89 **S1**v2 q41 Prop Calc (40m)
Mo	Sep	29	G2.3 (118-123) Proof of Correctness
We	Oct	01	96 **S2**v1 Big Oh Analysis (30m)
Fr	Oct	03	96 **S2**v2 Big Oh Analysis (30m)
Mo	Oct	06	G2.4 (129-140) Recursive Defs
We	Oct	08	113 **S3**v1 Counting (30m)
Fr	Oct	10	113 **S3**v2 Counting (30m)
Mo	Oct	13	G3.1 (186-201) Sets, 62 **p05** (20p) Fac
We	Oct	15	G3.2 (211-219) Cnt, 64 **p11** (20p) Mult
Fr	Oct	17	122 **S4**v1 Conditional Probability (50m)
Mo	Oct	20	G3.3 (225-231) Pigeon, 65 **p13** (20p)

We	Oct	22	G3.4 (233-246) Perm, 66 **p14** (25p)
Fr	Oct	24	122 **S4**v2 Conditional Probability (50m)
Mo	Oct	27	G3.5 (252-261) Cond P, 67 **p15** (25p)
We	Oct	29	G4.1 (285-301) Relations
Fr	Oct	31	138 **S5**v1 Binary Search Trees (50m)
Mo	Nov	03	G5.1 (401-422) Graphs, 68 **p21** (50p)
We	Nov	05	G5.2 (433-444) Trees
Fr	Nov	07	138 **S5**v2 Binary Search Trees (50m)
Mo	Nov	10	G5.3 (451-458) Decision Trees
We	Nov	12	G5.4 (462-469) Huffman, 70 **p22** (50p)
Fr	Nov	14	144 **S6**v1 Huffman (50m)
Mo	Nov	17	G6.1 (475-487) Directed Graphs
We	Nov	19	G6.2 (490-496) Euler, Hamiltonian
Fr	Nov	21	144 **S6**v2 Huffman (50m)
Mo	Nov	24	G6.3 (499-507) Shortest Path, MST
We	Nov	26	G6.4 (513-521) Traversal: BFS, DFS
Fr	Nov	28	** No Class: Thanksgiving Friday
Mo	Dec	01	148 **S7**v1 MST (50m)
We	Dec	03	
Fr	Dec	05	148 **S7**v2 MST (50m)
Mo	Dec	08	
We	Dec	10	** Final Exam, 07:00 to 09:50

1.4 Grading

I use a 60/70/80/90 model based on 1000 points.

Based on 1000 points

930+	A	900+	A–	870+	B+
830+	B	800+	B–	770+	C+
730+	C	700+	C–	670+	D+
630+	D	600+	D–	0+	F

The points are divided up as follows.

- Daily Update 35 (max 39)
- Daily Quiz 85 (max 90)
- Readings 100

- Study Time 80 (max 84)
- Programming 300
- Exams 400

1.4.1 Tracking Your Grade

I keep an online grade book so you can see how your points are coming along. It also lets you compare yourself with other students in the class (without seeing their names).

`https://dcquiz.byuh.edu/` is my personal Learning Management System. That is where I maintain my online grade book.

2145 CIS 205 Overall Grade Book: This includes the totals from all the other grade books. This is where you can find your final grade at the end of the course.

2145 CIS 205 Daily Update Grade Book

2145 CIS 205 Daily Quiz Grade Book

2145 CIS 205 RST Grade Book: This tracks readings and study time.

2145 CIS 205 Activities Grade Book: This tracks your performance on programming activities.

2145 CIS 205 Exam Grade Book: This tracks your performance on exams.

1.4.2 Daily Update (35 points)

Each day in class starts with the "daily update" (DU). It is my way of reminding you of due dates and deadlines, sharing updates and news, and taking roll. It is your way of saying something anonymously to each other and to me. It must be taken in class at a classroom computer during a window of time that starts a few minutes before class and ends 5 minutes into class.

Tardiness: My tardiness policy is that you should arrive in time to complete the daily update. Generally if you are only four minutes late or less, you will have time to complete the daily update before the deadline.

The DU is worth one point per class period, with 35 points expected (for 35 hours out of 39 class periods).

Attendance: My attendance policy is that you will attend at least 35 hours during the course. Anything beyond 35 points is extra credit. It is also a reward for coming on time, or close enough that you can do the update.

As part of the Daily Update, when readings are due I will ask you whether you read the assigned pages. I will use your report to update your readings points.

As part of the Daily Update, once a week I will ask you how much time you spent studying the previous week. I will use your report to update your study time points. See "Study Time" below for information.

1.4.3 Daily Quiz (85 points)

Right after prayer on many days there will be a short quiz (about a minute). It will typically consist of one or more questions that are based on the assigned readings in the text book.

1.4.4 Readings (100 points)

Mo	Sep	15	G1.1 (001-013) 5p
We	Sep	17	G1.2 (021-031) 5p
Mo	Sep	22	G2.1 (088-098) 5p
We	Sep	24	G2.2 (100-112) 6p
Mo	Sep	29	G2.3 (118-123) 3p
Mo	Oct	06	G2.4 (129-140) 5p
Mo	Oct	13	G3.1 (186-201) 7p
We	Oct	15	G3.2 (211-219) 4p
Mo	Oct	20	G3.3 (225-231) 3p
We	Oct	22	G3.4 (233-246) 6p
Mo	Oct	27	G3.5 (252-261) 4p
We	Oct	29	G4.1 (285-301) 8p
Mo	Nov	03	G5.1 (401-422) 9p
We	Nov	05	G5.2 (433-444) 5p
Mo	Nov	10	G5.3 (451-458) 4p
We	Nov	12	G5.4 (462-469) 4p
Mo	Nov	17	G6.1 (475-487) 6p
We	Nov	19	G6.2 (490-496) 3p
Mo	Nov	24	G6.3 (499-507) 4p

We Nov 26 G6.4 (513-521) 4p

Readings should be completed before class on the day assigned. They are listed here as items of the form Gx.y where G means the Gersting textbook, x is the chapter, and y is the section.

When reading each section, read through the “Main Ideas” summary in the “Review” section at the end of each section. You do not have to read the Exercises, but you may if you like. When reading section 1, also read the chapter introduction.

Readings should prepare you for the learning activities of the day. Do your best to understand the readings, but please read them even if you do not understand things fully. Then ask questions.

We award points for doing the readings, which means reading every word of the narrative portions assigned, and looking over the programs that are presented. The expectation is not 100% comprehension, but is 100% familiarity and as much comprehension as you can reasonably gain by normal reading. This provides a basis for in-class activities.

Readings are worth full credit if completed before class on the date they are due, and are worth half credit (rounded up) if completed within the week after the due date.

Credit is based on an all-or-nothing statement by the student in response to the question: Did you complete all of the assigned readings?

Time you spend doing the readings also counts as study time.

1.4.5 Study Time (80 points)

Keep a written daily log of the time you spend studying. We award points for study time (ST), which is time spent outside of class engaging with materials directly related to this course.

Each week you are invited to report, on your honor, how many hours outside of class you studied during the previous week, Sunday morning through Saturday night. We award one point per hour of “study,” for a total of 6 points per week, whether there is a holiday or not.

There are 14 weeks. 14 x 6 = 84. Points beyond 80 are extra credit.

Carry Forward: If you study more hours than the maximum for which I will give credit, you are invited to report them, and also carry forward the extra hours and report them in the next week. For example, if 6 hours is the maximum that counts and you studied 15 hours, you can report 15 hours of study, and I would count the first 6 hours. You would then take the remaining 9 hours and count it toward the following week.

There is no Carry Backward.

Most students max out the study time points each week.

1.4.6 Some Points are Optional

The readings and study time points are partly there as a safety net. They are meant to be easy to earn. They help to make sure you will pass the class.

But when I calculate your final grade, I do it several ways:

(a) Counting every point, based on 1000 total points.

(b) Counting all but readings and study time, based on 820 total points.

I grade several ways because some students have previous experience (or natural genius) and do not need to study as much.

I use whichever method gives you the best grade.

1.4.7 Skill: (300 points) GradeBot

Mo	Sep	08	56	**p01**	(20p)	Factors
We	Sep	10	57	**p02**	(20p)	Perfect Numbers
Mo	Sep	22	58	**p03**	(30p)	Fibonacci
We	Sep	24	60	**p04**	(20p)	Greatest Common Div
Mo	Oct	13	62	**p05**	(20p)	Prime Factors
We	Oct	15	64	**p11**	(20p)	Multiplication Rule
Mo	Oct	20	65	**p13**	(20p)	OSWOR
We	Oct	22	66	**p14**	(25p)	Choose
Mo	Oct	27	67	**p15**	(25p)	OWII
Mo	Nov	03	68	**p21**	(50p)	Binary Search Tree
We	Nov	12	70	**p22**	(50p)	Huffman Coding

During the semester we will do several computer programs. Supported languages include Perl, which is taught in the CIS 101 class, as well as C, C++, Java, Python, Ruby, and Tcl.

http://gbot.is2.byuh.edu/ has more.

Each programming assignment has its own section in the study guide where the task is described and deadlines are given. The study guide will be updated as needed throughout the semester.

GradeBot (aka GradeBot Lite, aka GBot) will accept your program and test it. When it finds an error, it will inform you so you can fix it. When it runs all its tests without finding any errors, it will inform you so you can turn it in to me. (GradeBot does not turn it in.)

Time you spend working on these programs also counts as study time.

Assignments are correct when they work properly and comply with all of the stated requirements in the study guide. Commonly that includes algorithm and style. Points are assigned according to the date on which the correct work is received. Details are provided in the study guide, but in general it works like this:

Full Credit: Correct by 23:59 the day it was due.

Penalty (-5pt): Each class day it is late.

1.4.8 Do Your Own Work

Help each other, but do your own work.

These two goals are in conflict with each other. To resolve this conflict, I draw the line at copying.

Except during exams, you are strongly encouraged to work with your fellow students. We want everyone to receive full credit on every assignment. Please help each other learn.

If You are Looking at Someone's Program:

You **may**: look, read, make mental notes, ask questions, point out possible errors, and try to understand.

You **must not**: fix their program, take written notes, take pictures, take a copy of all or part of their program.

If Someone is Looking at Your Program:

You **may**: explain your approach, ask for help.

You **must not**: let them fix your program, give them a copy of all or part of your program.

This includes working with tutors or students who took the class in previous semesters.

What Cheating Looks Like:

If 100 people in this class try to write the same program, parts of it will be identical. The overall flow of the program will likely be the same. Sometimes variable names will actually be the same.

But often people do a few random quirky things. They may have lines of code that do not hurt, but do not really help either.

I tend to notice those quirky things. When two or more students have the same quirks, that is evidence of cheating.

If I "call you out" on what looks like copying, your job will be to explain to me why you put those lines in your program. What is their purpose?

1.4.9 Skill: Exams (400 pts + 35 ec)

Fr	Sep	19	89 **S1** (60 pts + 6 ec)
Fr	Sep	26	89 **S1** second try
We	Oct	01	96 **S2** (50 pts + 5 ec)
Fr	Oct	03	96 **S2** second try
We	Oct	08	113 **S3** (50 pts)
Fr	Oct	10	113 **S3** second try
Fr	Oct	17	122 **S4** (60 pts + 6 ec)
Fr	Oct	24	122 **S4** second try
Fr	Oct	31	138 **S5** (60 pts + 6 ec)
Fr	Nov	07	138 **S5** second try
Fr	Nov	14	144 **S6** (60 pts + 6 ec)
Fr	Nov	21	144 **S6** second try
Mo	Dec	01	148 **S7** (60 pts + 6 ec)
Fr	Dec	05	148 **S7** second try
We	Dec	10	third try at anything

Each exam has its own chapter in the study guide where it is explained.

During the semester we will do in class seven "skills" exams that test your skill with certain concepts and procedures. Each exam will be available for practice, and will be given twice for credit. On the final exam day, each exam will be available again to let you try to improve your grade.

Most of the exams are designed on an 80/30 model, where the first 80% of the questions are of normal difficulty, and the last 30% of the questions are somewhat harder. Yes, that adds up to 110%. The other 10% is extra credit for those that can do it.

1.4.10 Other Extra Credit

Report an error in my formal communications (the published materials I provide), so I can fix it. In this class, the materials include the following:

- The course website, parts relating to this semester.
- The course syllabus.
- The course study guide.

Each error reported can earn you extra credit. (Typos in my email messages are all too common and do not count.)

Syllabus errors (unless they are major) will probably be fixed only in the study guide. Check there before reporting it.

1.5 Instructional Methods

Exams happen on scheduled exam days. Exams are an instructional method that brings you, the student, face to face with the challenges you need to be able to solve.

Lecture days happen occasionally. I review material that was assigned from the text book and do what I can to make it clear and interesting. These can take up most of the class hour, and happen more often at the start of the course than they do later on.

Activity days are usually the most common. A learning activity is assigned. Typically it is a program to be written. The program will be described in the study guide. I will give an overview of the problem and the techniques that I think will be helpful to solve it. Typically this takes about 15 minutes, but the actual time varies widely. Then I sit down at the front of the room and invite students to visit with me, one on one, for assistance. Students are also encouraged to help each other.

Help in Class: As students come to visit with me, I call up their computer screen from the place they were sitting, and we look at their program code or whatever else the student is asking about. We review the situation together. The student then returns to work on their program at their seat and I work with the next student waiting in line.

I want to help as many students as I can. You can help by doing these kinds of things before coming up.

(a) Have my study guide available on your desktop, turned to the relevant assignment so we can review the requirements.

(b) If it is a question about grades, have my gradebook available on your desktop.

(c) If it is about an exam question, have that exam available on your desktop.

1.5.1 BYUH Learning Framework

I believe in the BYUH Framework for Learning. If we follow it, class will be better for everyone.

Prepare for CIS 205

Prepare: Before class, study the course material and develop a solid understanding of it. Try to construct an understanding of the big picture and how each of the ideas and concepts relate to each other. Where appropriate use study groups to improve your and others' understanding of the material.

In CIS 205: Make reading part of your study. There is more than we could cover in class because we all learn at different rates. Our in-class time is better spent doing activities and answering your questions than listening to a general lecture.

Engage in CIS 205

Engage: When attending class actively participate in discussions and ask questions. Test your ideas out with others and be open to their ideas and insights as well. As you leave class ask yourself, "Was class better because I was there today?"

In CIS 205: Participate in the in-class activities. Those that finish first are encouraged to help those that want assistance. It is amazing what you can learn by trying to help someone else.

Improve at CIS 205

Improve: Reflect on learning experiences and allow them to shape you into a more complete person: be willing to change your position or perspective on a certain subject. Take new risks and seek further opportunities to learn.

In CIS 205: After each exam, I normally allow you to see every answer submitted, every score given, and every comment I wrote, for every question. Review your answers and those of other students. See how your answers could be improved. If you feel lost, study the readings again or ask for help.

1.5.2 Support

The major forms of support are (a) open lab, (b) study groups, and (c) tutoring.

If you still need help, please find me, even outside my posted office hours.

Office Hour / Open Lab

I have reserved GCB 111 on MWF 14:30 to 15:30 so my CIS 205 students (and others) can study in a lab setting and meet with me and each other. I allow the room as an Open Lab for your use either individually or in groups, for my class or for other classes. MWF 14:30 to 15:30 I will be present in GCB 111 or in my office to assist students that come.

Study Groups

You are encouraged to form a study group. If you are smart, being in a study group will give you the opportunity to assist others. By assisting others you will be exposed to ideas and approaches (and errors) that you might never have considered on your own. You will benefit.

See section 1.4.8 (page 14) for some rules.

If you are struggling, being in a study group will give you the opportunity to ask questions from someone that remembers what it is like to be totally new at this subject. They are more likely to understand your questions because they sat through the same classes you did, took the same tests as you did, and probably thought about the same questions that you did.

Most of us are smart some of the time, and struggling some of the time. Study groups are good.

Tutoring

The CIS department provides tutoring in GCB 111, Monday through Friday, typically starting around 17:00 and ending around 23:00 (but earlier on Fridays). Normally a schedule is posted on one of the doors of GCB 111.

Tutors can be identified by the red vests they wear when they are on duty.

The best way to work with a tutor is to show them something that you have written and ask them why it does not work the way you want. This can open the door to a helpful conversation.

Another good way to work with a tutor is to show them something in the textbook and ask about it.

The worst way to work with a tutor is to plunk down next to them and say, “I don’t understand. Can you teach me?” If you did not try hard to read carefully, you are wasting everybody’s time.

1.6 Course Policies

Subject to Change: Like all courses I teach, I will be keeping an eye out for ways this one could be improved. Changes generally take the form of opportunities for extra credit, so nobody gets hurt and some people may be helped. If I make a change to the course and it seems unfair to you, let me know and I will try to correct it. If you are brave enough, you are welcome to suggest ways the class could be improved.

Digital Recording: I may digitally record the audio of my lectures some days. This is to help me improve my teaching materials.

1.6.1 Excused Absences

There are many good reasons why students request special treatment. Instead of dealing with these as they arise, based on my years of experience, I have adopted general policies that are intended to accommodate all but the most difficult cases.

1.6.2 Reasonable Accommodation

This section covers special needs, including qualified special needs, as well as all other requests for special treatment.

I have carefully designed each of my classes to provide reasonable accommodation to those with special needs. Beyond that, further accommodation is usually considered to be unreasonable and only happens in extreme cases. Please see the paragraph on "Accommodating Special Needs" below for more information.

Ample Time: Specifically, I allow ample time on tests so that a well-prepared student can typically finish each test in half of the time allowed. This gives everyone essentially double the amount of time that should normally be needed.

Exam Retakes: I allow each test to be taken three (or more) times, and I keep the highest score that was earned. This handles the case of persons that are unable to attend class or function at their best on any given day. I consider the first attempt to be routine and the second and third attempts to be an accommodation for anyone that might need it. The scheduled final exam time consists of that third opportunity to retake **any** exam that was offered during the semester. If you are happy enough with your previous scores, **you can probably skip the final.**

As a side effect of this three-tries approach to exams, it is also true that any student can miss any one or two days of class for any reason without messing up their grade.

On the other hand, the retakes are limited. If you have issues every single time a certain test is given, I do draw the line, and I will not give additional chances. Additional retakes are not considered to be a "reasonable" accommodation.

Specifically, if you tend to miss class a lot because you do not wake up early enough, you will probably fail the class, even if you have a legitimate medical reason. It goes beyond reasonable accommodation.

Extra Credit: I have built about 10% of slack into the grading so you can miss a few points here or there and make them up elsewhere.

Deadlines: Most assignments are due soon after they are discussed, but I normally allow late work at full credit for several more weeks (except at the end of semester).

Even though I truly believe that these methods provide reasonable accommodation for almost everyone in almost every case, you might have a highly unusual situation for which I can and should do even more. You are welcome to see me about your situation.

1.6.3 Communication

We communicate with each other both formally and informally.

Formal communication tends to be written and precise. Formal is for anything truly important, like grades. Formal is authoritative.

Informal communication tends to be more casual and impromptu. Informal is meant to be helpful and efficient. Reminders are informal. Emails are informal. Explanations are usually informal.

Me to You, Formal

I communicate formally, in writing, through (a) the syllabus, (b) the study guide, and (c) the learning management system.

(a) Syllabus: `http://byuh.doncolton.com/cis205/2145/syl.pdf` is the syllabus for this course. It tells our learning objectives and how you will be graded overall. You can rely on the syllabus. After class begins, it is almost never changed except to fix major errors.

(b) Study Guide: `http://byuh.doncolton.com/cis205/2145/sguide.pdf` is the study guide for this course. It includes a copy of the syllabus. The study guide is updated frequently throughout the semester, as assignments are made and deadlines are established or updated.

(b1) Calendar: The study guide tells when things will happen. It contains specific due dates.

(b2) Assignments: The study guide tells what assignments have been made and how you will be graded, item by item. It provides current details and specific helps for each assignment. It provides guidance for taking the exams.

(c) DCQuiz: `https://dcquiz.byuh.edu/` is my learning management system. I use it to give tests. I use it to show you my grade books.

Me to You, Informal

My main informal channels to you are (a) word of mouth and (b) email.

(a) Word of Mouth, including Lecture: Class time is meant to be informative and helpful. But if I say anything truly crucial, I will also put it into the study guide.

(b) Email: My emails to you are meant to be helpful. But if I say anything truly crucial, I will also put it into the study guide. Normally I put CIS 205 at the front of the subject line in each email I send.

You to Me, Formal

Your formal channels to me, specifically how you turn in class work, are mainly via (a) the learning management system, (b) email, and (c) specifically requested projects.

(a) DCQuiz: To use my learning management system, you must log into it. Then, you can respond to questions I have posted. Each day there will be a "daily update". I say more on that below. Exams will also be given using DCQuiz.

(b) Email: You will use formal email messages to submit some of the programs you write and to tell me certain other things. The study guide tells how to send formal emails, including where to send them, what subject line to use, and what to put in the body of the message.

(c) Student Projects: The study guide may tell you to submit certain work in the form of a webpage or web-based program. If so, it will say specifically where to put it. I will go to that spot to grade it.

You to Me, Informal

Your informal channels to me, typically how you ask questions and get assistance, are mainly face to face and by email or chat.

Face to Face: If you need help with your class work, I am happy to look at it and offer assistance. Often this happens during class or during office hours. Often I will have you put your work on your computer screen, and then I will take a look at it while we talk face to face.

Email / Chat: You can also get assistance by sending me an email or doing a chat. I will do my best to respond to it in a reasonable and helpful way. If you want something formal, use the formal rules.

If you are writing about several different things you will usually get a faster response if you break it up into several smaller emails instead of one big email. I try to respond to a whole email at once, and not just part of it. I usually answer smaller and simpler emails faster than big ones.

1.7 Learning Outcomes

Outcomes (sometimes called objectives) are stated at several levels: Institutional (ILO), Program (PLO), and Course (CLO). In this section we set forward these outcomes and tell how they are aligned with one another.

1.7.1 ILOs: Institutional Outcomes

ILO: Institutional Learning Outcomes (ILOs) summarize the goals and outcomes for all graduates of BYUH.

Brigham Young University Institutional Learning Objectives (ILOs) Revised 24 February 2014

Graduates of Brigham Young University–Hawai‘i (BYUH) will:

Knowledge: Have a breadth of knowledge typically gained through general education and religious educations, and will have a depth of knowledge in their particular discipline.

Inquiry: Demonstrate information literacy and critical thinking to understand, use, and evaluate evidence and sources.

Analysis: Use critical thinking to analyze arguments, solve problems, and reason quantitatively.

Communication: Communicate effectively in both written and oral form, with integrity, good logic, and appropriate evidence.

Integrity: Integrate spiritual and secular learning and behave ethically.

Stewardship: Use knowledge, reasoning, and research to take responsibility for and make wise decisions about the use of resources.

Service: Use knowledge, reasoning, and research to solve problems and serve others.

1.7.2 PLOs: Program Outcomes

PLO: Program Learning Outcomes (PLOs) summarize the goals and outcomes for graduates in programs for which this course is a requirement or an elective. These support the ILOs, but are more specific.

At the end of this section, we include the relevant page from the CIS Program Outcomes Matrix, dated April 2011.

The following outcomes are pursued at the "Introduced" level, and apply to one or more of the majors that use this course.

(a) An ability to apply knowledge of computing and mathematics appropriate to the discipline.

(b) An ability to analyze a problem, and identify and define the computing requirements appropriate to its solution.

(i) An ability to use current techniques, skills, and tools necessary for computing practice.

CS (j) An ability to apply mathematical foundations, algorithmic principles, and computer science theory in the modeling and design of computer-based systems in a way that demonstrates comprehension of the trade-offs involved in design choices.

CS (k) An ability to apply design and development principles in the construction of software systems of varying complexity.

1.7.3 CLOs: Course Outcomes

CLO: Course Learning Outcomes (CLOs, also called Student Learning Outcomes, or SLOs) summarize the goals and outcomes for students who successfully complete this course. These support the PLOs, but are more specific.

Course Goals and Student Learning Outcomes are that by the conclusion of this course, students will understand:

- Formal Logic. (We learn modus ponens, modus tollens, and resolution. There is an exam.)
- Recursion and Memoization. (We write a recursive Fibonacci program that relies on memoization to save time.)
- Proofs and Induction. (We discuss proofs and how they relate to formal logic.)
- Big Oh Analysis. (We do simple big-oh analysis, no recursion. There is an exam.)
- Set Theory. (We do union, intersection, and counting. There is an exam on counting.)
- Conditional Probability. (Given certain probabilities, calculate related probabilities. There is an exam.)
- Functions and Relations. (We talk about it.)
- Graphs and Trees and Recursion. (We learn to construct binary search trees. We learn to traverse trees depth first and breadth first. Recursion is explained. There is an exam.)
- Decision Trees. (We learn to do Huffman coding. There is an exam.)
- Euler Path and Hamiltonian Circuit. (We discuss them. We show EP is easily solvable, and HC is really hard.)
- Minimum Spanning Trees. (We learn how. There is an exam.)

These support and roughly correspond to the following higher-level outcomes.

- Demonstrate the ability to understand and apply knowledge appropriate for Computer Science.

- Understand and be able to apply mathematical foundations, algorithmic principles, and computer science theory in the modeling and design of computer-based systems in a way that demonstrates comprehension of the tradeoffs involved in design choices.

CIS Department Outcomes Matrix, April 2011

Program Outcomes

(a) An ability to apply knowledge of computing and mathematics appropriate to the discipline.
(b) An ability to analyze a problem, and identify and define the computing requirements appropriate to its solution.
(c) An ability to design, implement, and evaluate a computer-based system, process, component, or program to meet desired needs.
(d) An ability to function effectively on teams to accomplish a common goal.
(e) An understanding of professional, ethical, legal, security and social issues and responsibilities.
(f) An ability to communicate effectively with a range of audiences.
(g) An ability to analyze the local and global impact of computing on individuals, organizations, and society.
(h) Recognition of the need for and an ability to engage in continuing professional development.
(i) An ability to use current techniques, skills, and tools necessary for computing practice.

CS Only

(j) An ability to apply mathematical foundations, algorithmic principles, and computer science theory in the modeling and design of computer-based systems in a way that demonstrates comprehension of the tradeoffs involved in design choices. [CS]
(k) An ability to apply design and development principles in the construction of software systems of varying complexity. [CS]

IS Only

(j) An understanding of processes that support the delivery and management of information systems within a specific application environment. [IS]

IT Only

(j) An ability to use and apply current technical concepts and practices in the core information technologies. [IT]
(k) An ability to identify and analyze user needs and take them into account in the selection, creation, evaluation and administration of computer-based systems. [IT]
(l) An ability to effectively integrate IT-based solutions into the user environment. [IT]
(m) An understanding of best practices and standards and their application. [IT]
(n) An ability to assist in the creation of an effective project plan. [IT]

R = Required in that program | **CSS** = CS B.S. |**CIS** = CIS B.S. | **IS** = IS B.S. | **IT** = IT B.S.
# = choose at least 9 cr hrs | O = optional as a substitute | L = Introduced, M = Practiced with feedback, H = Demonstrated at the Mastery level

Course	**Description**	**CSS**	**CIS**	**IS**	**IT**	**a**	**b**	**c**	**d**	**e**	**f**	**g**	**h**	**i**	**CSj**	**CSk**	**ISj**	**ITj**	**ITk**	**ITl**	**ITm**	**ITn**
CIS 100	Fundamentals of Info. Systems & Tech.			R	R	L	L	L	L	L	L	L	L	L			L	L	L			
CIS 101	Beginning Programming	R	R	R	R	L	L							L	L	L						
CIS 202	Object-Oriented Programming	R	R	R	R	M	M	M		L			L	M	L	L		M	L		L	L
CIS 205	Discrete Mathematics I	R	R	R	R	M	M	L	L					M	M	M						
CIS 206	Discrete Mathematics II	R	R	R		M	M	L	L					M	M	M						
CIS 305	Systems Engineering I	R	R	R	R	M	M	M	M	L	L	M	L	M	L	L	M	L	H	L	H	M
CIS 401	Web Application Development	R		R	R	M	L	L						M			L	M	L	L		
CIS 405	Systems Engineering II	R	R	R	R	M	M	M	M	L	M	M	M	M	M	M	M	M	H	M	H	M
CIS 470	Ethics in Computer & Info. Sciences	R	R	R	R		L	L	M	H	H	H	H									
CS 203	Object-Oriented Programming II	R				M	M	M						M	M	M						
CS 210	Computer Organization	R			R	H	M	L							M	L		M				
CS 301	Algorithms & Complexity	R				L	M	L	L		M		L	M	H							
CS 320	Computational Theory	R				H	M			L		L	M		H	M						
CS 415	Operating Systems Design	R				H	H	H		M	M	M	H	H	H	H					M	
CS 420	Programming Languages	R				H	H	H		M	M	M	H	H	H	H						
CS 490R	Adv Topics in Computer Science (6 CR)	R				H	H	H					H		H	H						
IS 330	Management Information Systems					L	L		M	L	M	L	L	L			L					
IS 350	Database Management Systems	R	R	R	R	M	L	M	M	L	L	L	L	M	M	L	L	H	L			
IS 430	ITS – Enterprise Resource Planning			R			L	M	M	M	M	M	M	H			H		L		M	
IS 435	Advanced Concepts ERP Systems					H	H		H	L	M	M	M	H			H			L	H	
IS 485	Project Management & Practice			R		M	H	M	H	M	H	M	H	M	M	H	H	M				H
IT 220	Linux Essentials				R	M								M				M				
IT 224	Computer Hardware & Systems Software			R	R	M	H	L	M	L	M	L	L	L				M	M	L	L	
IT 240	Fund. Of Web Design & Technology			R	R	L	L	L		M	H	M		M		L	L	M	M	M	L	
IT 280	Data Comm. Systems & Networks	R	R	R	R	M	M	M		M	M		L	M				M	L	L		
IT 420	Linux System Administration				R	H	H	M						H				M	M	M		
IT 426	Computer Network Services				R	H	H	M	L	L	L	L	L	M				H	M	M	M	L
IT 440	Foundations of HCI				R	M	H	H	M	H	M	H	M	M			H	M	H	H	H	M
IT 480	Computer Network Design				R	H	H	H					L, M	H				M	M	M		M
IT 481	Information Assurance & Security				R		L	L		L	L	L	L	M				M	M	L	M	L
Math 112	Calculus I	O		R	#																	
Math 113	Calculus II	O			#																	
Math 119	Applied Calculus	R	O	O	#																	
Math 214	Mulitvariable Calculus				#																	

1.8 General Topics

All syllabi are encouraged or required to address certain topics. These are generally considered to be common sense, but we find that it is useful to mention them explicitly anyway.

1.8.1 Academic Integrity

Applicable Actions

`http://honorcode.byuh.edu/` details the university honor code. In the section entitled "Applicable Actions" the following are listed.

Examples of possible actions include but are not limited to the following, for instructors, programs, departments, and colleges:

• Reprimanding the student orally or in writing.

• Requiring work affected by the academic dishonesty to be redone.

• Administering a lower or failing grade on the affected assignment, test, or course.

• Removing the student from the course.

• Recommending probation, suspension, or dismissal.

Depending on the specifics of the offense, any of these responses may be possible.

Cheating on exams is one form of dishonesty that I encounter. Normally this happens when students bring in notes that include answers to past exam questions. I approve the studying of past exams, and bringing in of "memories" based on study, but not the access to written notes, including notes retrieved from other exams or stored on cell phones or other devices. Any such activity, if caught, can result in failure of the entire course.

Cheating on activities also happens. Copy and paste is not allowed, but is sometimes difficult to prove. You should understand the work you submit because it helps you prepare for the exams.

Plagiarism

We learn by watching others and then doing something similar.

Plagiarism: Sometimes it is said that plagiarism is copying from one person, and research is "copying" from lots of people.

When you are having trouble with an assignment, I encourage you to look at not just one, but many examples of work done by others. Study the examples. See what you can learn from them. Do not automatically trust that they are right. They may be wrong.

`http://en.wikipedia.org/wiki/Plagiarism` has a wonderful article on plagiarism. Read it if you are not familiar with the term. Essentially, plagiarism is when you present the intellectual work of other people as though it were your own. This may happen by cut-and-paste from a website, or by group work on homework. In some cases, plagiarism may also create a violation of copyright law. If you borrow wording from someone else, identify the source.

Intentional plagiarism is a form of intellectual theft that violates widely recognized principles of academic integrity as well as the Honor Code. Such plagiarism may subject the student to appropriate disciplinary action administered through the university Honor Code Office, in addition to academic sanctions that may be applied by an instructor.

Inadvertent plagiarism, whereas not in violation of the Honor Code, is nevertheless a form of intellectual carelessness that is unacceptable in the academic community. Plagiarism of any kind is completely contrary to the established practices of higher education, where all members of the university are expected to acknowledge the original intellectual work of others that is included in one's own work.

Do not simply copy. Do your own work. When I review computer code, sometimes I see quirky ways of doing things. They appear to work even though they may be wrong. And then I see someone else that has done it exactly the same wrong way. This does not feel like "doing your own work." Cut and paste is pretty much an honor code violation. Read and learn is totally okay. Copying other ideas is okay. I don't want to see any cut and paste.

CIS 205: In this course study groups are permitted and encouraged. See section 1.4.8 (page 14) for rules.

You must write your own programs. You can look at what other people have done, and you can show other people what you have done, but you are forbidden to copy it. Look at it, yes. Understand it, yes. Ask about it, yes. Explain it, yes. Copy it, no.

CIS 205: On exams you are required to work from personal memory, using only the resources that are normally present on your computer. This means the exams are closed book and closed notes. Students caught cheating on an exam may receive a grade of F for the semester, no matter how many points they may have earned, and they will be reported to the Honor Code office.

Faculty are responsible to establish and communicate to students their expectations of behavior with respect to academic honesty and student conduct in the course. Observations and reports of academic dishonesty shall be investigated by the instructor, who will determine and take appropriate action, and report to the Honor Code Office the final disposition of any incident of academic dishonesty by completing an Academic Dishonesty Student Violation Report. If the incident of academic dishonesty involves the violation of a public law, e.g., breaking and entering into an office or stealing an examination, the act should also be reported to University Police. If an affected student disagrees with the determination or action and is unable to resolve the matter to the mutual satisfaction of the student and the instructor, the student may have the matter reviewed through the university's grievance process.

1.8.2 Sexual Misconduct

Sexual Harassment is unwelcome speech or conduct of a sexual nature and includes unwelcome sexual advances, requests for sexual favors, and other verbal, nonverbal, or physical conduct. Conduct is unwelcome if the individual toward whom it is directed did not request or invite it and regarded the conduct as undesirable or offensive.

Brigham Young University–Hawai'i (BYUH) is committed to a policy of nondiscrimination on the basis of race, color, sex (including pregnancy), religion, national origin, ancestry, age, disability, genetic information, or veteran status in admissions, employment, or in any of its educational programs or activities.

University policy and Title IX of the Education Amendments of 1972 prohibits sexual harassment and other forms of sex discrimination against any participant in an educational program or activity at BYUH, including student-to-student sexual harassment.

The following individual has been designated to handle reports of sexual harassment and other inquiries regarding BYUH compliance with Title IX:

```
Debbie Hippolite-Wright
Title IX Coordinator
Vice President, Student Development & Life
Lorenzo Snow Administration Building
55-220 Kulanui Street
Laie, Hawaii 96762
Office Phone: 808-675-4819
E-Mail: debbie.hippolite.wright@byuh.edu
Sexual Harassment Hotline: 808-780-8875
```

BYUH's Office of Honor upholds a standard which states that parties can only engage in sexual activity freely within the legal bonds of marriage between a man and a woman. Consensual sexual activity outside the bonds of marriage is against the Honor Code and may result in probation, suspension, or dismissal from the University.

1.8.3 Dress and Grooming Standards

The dress and grooming of both men and women should always be modest, neat and clean, consistent with the dignity adherent to representing The Church of Jesus Christ of Latter-day Saints and any of its institutions of higher learning. Modesty and cleanliness are important values that reflect personal dignity and integrity, through which students, staff, and faculty represent the principles and standards of the Church. Members of the BYUH community commit themselves to observe these standards, which reflect the direction given by the Board of Trustees and the Church publication, "For the Strength of Youth." The Dress and Grooming Standards are as follows:

Men. A clean and neat appearance should be maintained. Shorts must cover the knee. Hair should be clean and neat, avoiding extreme styles or colors, and trimmed above the collar leaving the ear uncovered. Sideburns should not extend below the earlobe. If worn, moustaches should be neatly trimmed and may not extend beyond or below the corners of mouth. Men are expected to be clean shaven and beards are not acceptable. (If you have an exception, notify the instructor.) Earrings and other body piercing are not acceptable. For safety, footwear must be worn in all public places.

Women. A modest, clean and neat appearance should be maintained. Clothing is inappropriate when it is sleeveless, strapless, backless, or revealing, has slits above the knee, or is form fitting. Dresses, skirts, and shorts must cover the knee. Hairstyles should be clean and neat, avoiding extremes in styles and color. Excessive ear piercing and all other body piercing are not appropriate. For safety, footwear must be worn in all public places.

1.8.4 Accommodating Special Needs

Brigham Young University–Hawai‘i (BYUH) is committed to providing a working and learning atmosphere, which reasonably accommodates qualified persons with disabilities. If you have a disability and need accommodations, you may wish to self-identify by contacting:

```
Services for Students with Special Needs
McKay Building, Room 181
Phone: 808-675-3518 or 808-675-3999
Email address: aunal@byuh.edu
```

The Coordinator for Students with Special Needs is Leilani A‘una.

Students with disabilities who are registered with the Special Needs Services should schedule an appointment with the instructor to discuss accommodations. If the student does not initiate this meeting, it is assumed no accommodations or modifications will be necessary to meet the requirements of this course. After registering with Services for Students with Special Needs, and with permission of the student, Letters of Accommodation will be sent to instructors.

If you need assistance or if you feel you have been unlawfully discriminated against on the basis of disability, you may seek resolution through established grievance policy and procedures. You should contact the Human Resource Services at 808-780-8875.

1.9 Syllabus Summary

Brigham Young University–Hawai‘i (BYUH) has adopted certain requirements relating to the information that must be provided in syllabi. This section lists those requirements and for each item either provides the information directly or gives a link to where it is provided above.

Course Information: See section 1.2.1 (page 7).

Title: Discrete Mathematics I

Number: CIS 205

Semester/Year: Fall 2014

Credits: 3

Prerequisites: CIS 101

Location: GCB 140

Meeting Time: MWF 08:40 to 09:40

Faculty Information: See section 1.2.2 (page 7).

Name: Don Colton

Office Location: GCB 128

Office Hours: MWF 14:30 to 15:30.

Telephone: 808-675-3478

Email: doncolton2@gmail.com

Course Readings/Materials: See section 1.2.3 (page 8) for a list of textbooks, supplementary readings, and supplies required.

Course Description: See section 1.2.1 (page 7).

Expected Proficiencies:
See section 1.1.2 (page 6) for the proficiencies you should have before undertaking the course.

Course Goals and Student Learning Outcomes, including Alignment to Program (PLOs) and Institutional (ILOs) Learning Outcomes, and extent of coverage.

See section 1.7 (page 23) for learning outcomes, showing the content of the course and how it fits into the broader curriculum. A listing of the departmental learning outcomes is provided together with the ratings taken from department's matrix assessment document representing the degree to which the course addresses each outcome.

Instructional Methods: See section 1.5 (page 16).

Learning Management System:
`https://dcquiz.byuh.edu/` is the learning management system for my courses.

Framework for Student Learning:
See section 1.5.1 (page 17) for a discussion of the student learning framework and how I use it.

Course Calendar: See section 1.3 (page 8) for the calendar in general.

Here are some items of particular interest:

First Day of Instruction: Mon, Sep 08

Last Day to Withdraw: Fri, Oct 31

Last Day of Instruction: Mon, Dec 08

Final Exam: Wed, Dec 10, 07:00 to 09:50

Final Exam Location: GCB 140

Course Policies: See section 1.6 (page 19).

Attendance: See section 1.4.2 (page 11).

Tardiness: See section 1.4.2 (page 10).

Class Participation: See section 1.5.1 (page 18).

Make-Up Exams: See section 1.6.2 (page 20).

Plagiarism: See section 1.8.1 (page 29).

Academic Integrity: See section 1.8.1 (page 28).

Evaluation (Grading): See section 1.4 (page 9).

Academic Honesty: See section 1.8.1 (page 28).

Sexual Harassment and Misconduct: See section 1.8.2 (page 30).

Grievances: The university grievance policy states that the policies listed on the syllabus can act as a contract and will be considered if a student complains about the faculty.

Services for Students with Special Needs: See section 1.8.4 (page 32).

Chapter 2

DCQuiz: My Learning Management System

Contents

2.1 Grade Book **37**
2.2 Daily Update **37**
2.2.1 Study Time 37
2.2.2 Comment 37
2.2.3 Genuine Questions 38
2.3 Exams **38**
2.3.1 Taking Exams 39
2.3.2 Practice Tests 40
2.3.3 ezCalc 40
2.3.4 Reviewing Exams 41
2.4 Other Features **42**

I developed my own learning management system (LMS) which we will use for this course. Other LMS examples include BlackBoard, Canvas, and Moodle. (I did not write them.) I currently do not use them.

`https://dcquiz.byuh.edu/` is the DCQuiz URL.

Since I wrote it myself, I am also responsible for any bugs that may be in its programming. If you notice any bugs, I hope you will let me know so I can get them fixed.

I can also make improvements when I think of them. I like that.

2.1 Grade Book

The most important place you will see DCQuiz is the grade book.

I use DCQuiz to manage my grade book for this class. You will be able to see the categories in which points are earned, and how many points are credited to you.

You will also be able to see how many points are credited to other students, but you will not be able to see which students they are.

This gives you the ability to see where you stand in the class, on a category-by-category basis, and in terms of overall points. Are you the top student? Are you the bottom? Are you comfortable with your standing?

2.2 Daily Update

Another place you are likely to see DCQuiz is the daily update.

Typically in class I start with a quiz called the Daily Update. It usually runs the first five minutes of class, and is followed by the opening prayer.

By having you log in and take the daily update quiz, I also get to see who is in class, in case I need a roll sheet and I did not take roll in some other way.

2.2.1 Study Time

Generally I give you the opportunity to tell me how much study time you have accumulated since the last reporting. Normally this is reported on the first class of the week, and covers the prior week (Sunday through Saturday).

2.2.2 Comment

Generally I also give you an opportunity to make an anonymous comment. This can be anything you want to say. It might include announcements, such as birthdays or concerts. It might include questions. It can be a simple greeting.

Comments provide a chance for each student to say something without the embarrassment of everyone else knowing who said it. You can say how unfair you think I am for something. You can ask about something you find confusing.

I introduce it something like this:

If you wish, you can type in a comment, question, announcement, or other statement at the start of class for us to consider. Or you can leave this blank.

This is a good opportunity to ask about something you find confusing.

The identity of the questioner (you) will not be disclosed to the class, and normally I will not check (although I could). My goal is for this to be anonymous.

2.2.3 Genuine Questions

I may include genuine questions in the daily update, and these can be graded. It's kind of unpredictable.

2.3 Exams

DCQuiz was originally developed for giving tests. My problem was handwriting, actually. Students would take tests on paper and sometimes I could not read what they had written.

So I cobbled together an early version of DCQuiz to present the questions and collect the answers.

I got a couple of additional wonderful benefits, almost immediately.

First, I got the ability to grade students anonymously. All I was seeing was their answer. Not their handwriting. Not the color of their ink. Not their name at the top of the paper. It was wonderful. I could grade things without so much worry about whether every student was being treated fairly.

Second, I got the ability to share my grading results with every student in the class. Each student can see, not only the scores earned by other students, but the actual answers that other students put to each question. This gives students the ability to learn from each other.

Third, it gave students a way to verify that they were being graded fairly compared to their fellow students. If you can see your own answer, and see that everyone with higher points gave a better answer, that is a good thing. If you think your answer is better, it gives you a reason to come and see the teacher so you can argue for more points, or you can be taught the reasons for their answers getting more points.

Fourth, it gave me the convenience of grading anywhere without carrying a stack of papers. I could grade on vacation. (Wait. Doesn't that make it a not-vacation?) I could grade in class, or in my office, or at home.

Fifth, although I never did this, it theoretically has the ability for me to let other people be graders. But I never did this.

2.3.1 Taking Exams

As it currently operates, DCQuiz lets you, the student, log in and see a list of quizzes. (The grade book is actually just another quiz, but it is one where I enter grades that you earned some other way.)

Quizzes typically have starting and ending times. Before the quiz starts, there is a note telling when it will start. As the quiz gets closer, like within an hour or two, an actual count-down clock will appear telling you how long until the quiz is available.

Once you start the quiz, if it has an ending time, you will be able to see a count-down timer telling you how much time you have left.

As you take a quiz, you can see the main menu, the question menu, and the question page.

Main Menu: The main menu was already mentioned. That's where you see what quizzes are available.

Question Menu: The question menu shows you what questions are on this quiz. It lets you select a question to work on. It shows you which questions you have answered already. It shows you which answers have already been scored. It lets you say that you are done. It lets you cancel the quiz (if that is allowed).

Question Page: The question page shows a single question, and lets you type in your answer. Some questions only allow a single-line answer. When you press ENTER it takes you automatically to the next question. Other questions let you type in several lines.

2.3.2 Practice Tests

Some questions have exactly one right answer. For this class, that is usually the case. For those questions, grading is easy and automatic. When grading is automatic, I will often set up a practice test. It works like this.

1. The test usually has the word "prac" in its title.
2. It can be taken anywhere, any time.
3. As you complete each question, if you got the answer right, it will tell you immediately.
4. As you complete each question, if you got the answer wrong, it says nothing special, but it lets you go back and fix your answer.

2.3.3 ezCalc

ezCalc is actually part of the Question Page, but it is special enough that I decided to let it have its own section in this study guide.

Many times a question will call for a numeric answer. In the early days of DCQuiz I had to make the questions easy enough to do in your head, or on paper, or else I had to give people access to a calculator. I was not comfortable letting them use a calculator because it might already be programmed for the kind of question I was asking.

Eventually I found that too limiting, so I added a capability into DCQuiz to do simple calculations. I call it "ezCalc" and it can be used on single-line answers. Sometimes. If I decide to let it be used.

Say for instance that the answer to a question can be calculated as 7 x 6 x 5 x 4 x 3 x 2 x 1. (You probably recognize that as 7 factorial.) That may be easy to think about but hard to calculate. Beyond about 4 factorial, they are pretty hard to calculate without a calculator. But 4 factorial really limits the number of questions I can ask.

So I wrote ezCalc.

It works like this. There is a special blank provided on each question. It is labeled with the word ezCalc.

You key in a mathematical expression, like 7*6*5*4*3*2*1, and press the = key or press Enter. ezCalc will replace your expression with the answer, which is 5040 in this case.

ezCalc uses the “*” key for multiplication. It also has “+” for addition, “-” for subtraction, “/” for division, “%” for remainder (or modulus), and parentheses for grouping and controlling the order of operations.

The purpose of ezCalc is to make it possible for me to ask questions that require bigger numbers, but without letting big numbers get in your way. As long as you know how to do the calculation, ezCalc will do it for you.

ezCalc will not let you use variables or functions besides the ones I mentioned above. Sorry.

2.3.4 Reviewing Exams

When an exam is finished, DCQuiz lets me, the author of the exam, share it with you, the student who took the exam.

You can see reviewing opportunities on the main menu.

After selecting an exam to review, you will see a question menu similar to the one that was used for taking the exam. But instead of seeing your answer, you will see all the scores that were earned, with your score highlighted. If yours is the top score, it will appear first. If it is the bottom score, it will appear last.

You can select a question to drill down and see more details. Specifically, you can see each of the answers provided by each student that wrote an answer. And you can see the score it received. And you can see any notes the grader (me) may have made while grading.

This is intended to (a) let you teach yourself by seeing examples of work by other students, and (b) let you verify that you were graded fairly. (Every once in a while, maybe a few times per semester, a student will see that I entered their grade wrong, or I overlooked something. This is your chance to get errors fixed.)

Sometimes an exam is not open for review. The teacher gets to decide. But even if the exam is closed, you can still see the question menu (with the questions blanked out), and you can see your score and everyone else’s score. Questions and answers are not available, but scores are available, even long after you took the test.

Sometimes an exam is deleted or revised and reused. The teacher gets to decide. When an exam is deleted, all questions and answers and scores are also deleted. After that, there is no way to see anything about that exam.

I generally revise and reuse the daily update exams. This causes all answers and scores to be deleted, but I keep the questions and just modify them for the next class meeting.

2.4 Other Features

DCQuiz has other features, such as the ability to limit where a test is taken, or to require a special code to access a test. Those features will be explained in class if they are ever needed.

Chapter 3

GradeBot

GradeBot is my automated program grader.

Contents

3.1 Where Can I Find GradeBot? **44**
3.2 Source Code File **44**
3.3 Choice of Language **45**
3.4 Customizations **45**
3.5 Standard In, Standard Out **46**
3.6 The Grading Script **46**
3.7 Understanding GradeBot's Requirements **47**
3.8 Submitting Your Work **49**
3.9 Email Submission Rules **49**
3.9.1 To: Line 49
3.9.2 Subject Line 50
3.9.3 Body When Submitting a Program 50
3.9.4 How Grading Happens 52
3.10 Programming Style **52**

I normally grade programming activities based on the following three criteria.

Behavior: How does the program respond to various situations? This is always important.

Style: How clearly is the program written? My standards vary from course to course and from assignment to assignment.

Algorithm: What algorithms are used? Were they efficient? My standards vary from course to course and from assignment to assignment.

When I grade on style and algorithm, I usually rely on visual inspection of the program source code.

When I grade on behavior, I try to use GradeBot. GradeBot verifies that your program seems to be running correctly by giving your program test cases to solve, and then seeing whether your program returns the correct answer each time.

This particular version of GradeBot is sometimes called GradeBot Lite because it is descended from what used to be a huge system that was also called GradeBot.

3.1 Where Can I Find GradeBot?

`http://gradebot.tk/` is the web interface for GradeBot.

`http://gbot.is2.byuh.edu/` is an alternate URL that you can use in case the short URL does not work for you.

If you want to explore, press the `[List All Labs]` button. Then pick a lab from the list and press its button.

GradeBot will give you a “SAMPLE EXECUTION” to show the behavior it is expecting from your program.

3.2 Source Code File

To keep things simple, GradeBot requires you to submit a single file of source code. You are not permitted to write modules as separate files, and compile them separately and then link the results. Such abilities are present in Integrated Development Environments, but not in GradeBot. Everything must fit into a single file. In some languages, this can be an uncomfortable restriction.

GradeBot also allows you to type your program directly into the web interface. It also allows you to upload your file by selecting it on your local computer.

3.3 Choice of Language

GradeBot is able to accept programs in any of several different languages. The list includes C, C++, Java, Perl, Python, Ruby, and Tcl.

You must tell GradeBot what language you are using. Simply click on the appropriate radio button.

When grading your program, GradeBot will compile or interpret your program and then check its behavior. If your program has syntax errors, GradeBot will let you know. If there are no syntax errors, GradeBot will run your program.

Java: This popular language is perhaps the most disadvantaged with GradeBot for several reasons: (a) Java likes to have separate files; (b) Java takes a long time (half a second) to start, so 20 tests will take 10 seconds if you are lucky. The bottom line is that Java can be used, but it is a bit harder to make things work.

3.4 Customizations

GradeBot can customize its requirements for each student. This is done based on the first line of the program, which may be a required comment line.

If that first line includes a particular string, the whole line is used to seed a random number generator that creates the differing requirements for each student.

For example, the instructor may require each program to include a comment line that identifies the task and the student, something like this:

```
# cis101 g05 Colton, Don
```

GradeBot may be looking for the string `cis101 g05` as its trigger for customizing.

Because these customizations are intended to be different for every possible first line, the result is that every student may have to create a slightly different program.

Changing the first line can change the customizations, so after you start working on a program, you should be careful about changing the first line.

The other lines of the program do not affect the customizations in any way.

An example of a customization is the string the student must use to prompt for the first input. GradeBot might select from among the following options.

```
Please enter a number.
Kindly enter a number.
Please type in your number.
Type in a number please.
```

The purpose of this customization feature is to cut down on simple copying that might go on among students. Without customization, the same program could be submitted by every student. With customization every student would need to adjust the behavior of their program a bit to match the expectations of GradeBot.

3.5 Standard In, Standard Out

Rigorous testing of computer programs is actually very difficult. To make it possible, I have made some decisions to keep things as simple as possible.

Tasks are generally limited to programs that read input from STDIN and write output to STDOUT. Beyond that, it can provide input through command line arguments, and it can inspect the return value from the programs it is testing.

That means that GradeBot does not get involved with mouse-based input, or network-based input or output, or graphical outputs.

Generally GradeBot does not get involved with reading and writing files, or accessing databases.

GradeBot also limits the amount of time each program is allowed to run.

3.6 The Grading Script

GradeBot has a version of each program you are asked to write. It generates sample inputs (somewhat randomly), runs its own version of the program, and collects the outputs. That is used to make a script: say this to the program, wait for the program to say this, repeat.

Your program is tested by seeing whether it can follow the script. Your program must behave exactly the same as GradeBot's program did.

This puts some serious constraints on your program. You must get all the strings right. If GradeBot wants "Please enter a number: " then that's exactly what your program must print. You may find yourself squinting at the output where GradeBot says you missed something. Usually it is a minor typographical problem.

Once you get the first test right, GradeBot typically invents another test and has you run it. And another. And another. Eventually, you either make a mistake, or you get them all correct.

If you make a mistake, GradeBot will tell you what it was expecting, and what it got instead.

If you get everything correct, GradeBot will announce your success. That means your program's observable behavior is correct.

3.7 Understanding GradeBot's Requirements

GradeBot is very picky. That is because it is really hard to tell when something is almost right, or close enough. GradeBot's only real choice is to require absolute perfection. That means spelling and spacing must be exact.

GradeBot shows you exactly what it wants, and exactly what you provided. Sometimes it can tell you what is wrong, but often you have to compare things and figure it out yourself. Just do a careful side-by-side comparison and adjust your spelling and spacing to make GradeBot happy.

(GradeBot may actually do things wrong in some cases. Maybe GradeBot spells something wrong or uses the wrong grammar. If you run into a case like that, the simple solution is to play along and do it the way GradeBot wants. You can also let me know where GradeBot is wrong and I may be able to fix it.)

Standard Input

"`in>`" is shown to designate input that your program will be given (through the standard input channel).

Standard Output

Numbered lines are shown to designate output that your program must create.

Quotes are shown in the examples to delimit the contents of the input and output lines. The quotes themselves are not present in the input, nor should they be placed in the output.

Each line of output ends with a newline character unless specified otherwise.

"eof" stands for end of file and means that your program must terminate cleanly.

Command Line Inputs

GradeBot will start your program by saying this:

```
GradeBot started your program with this command line:
```

It will then present the command line that was used to start your program. Often there is no special command line input, and the command line simply starts your program. Sometimes there are command line arguments. Here is an example:

```
"./gcd 35 28"
```

In this example, `./gcd` is the name of your program that is being run. `35` is the first command line argument. `28` is the second command line argument.

In most languages, command line arguments are available as an array named "argv" or "args" or something similar.

Return Codes

When your program terminates, it must send back a return code of zero unless something else was specified in the requirements. Many languages do this automatically.

For C programs, remember to start your program with "int main" and end it with "return 0;"

For other languages, do something similar if necessary.

3.8 Submitting Your Work

When GradeBot thinks you have a working program, and you also think so, you can formally submit it for credit.

Follow the email rules in section 3.9 (page 49) as you construct your email to me.

It is okay to work ahead, but please do not submit your work until I ask for it. There may be special things about the program that we have not discussed.

Warning: You must submit exactly what you have tested, no more and no less. In particular, you are required to submit a program that starts with a specific comment line which identifies the course, the program, and the person submitting the program. This comment line might cause GradeBot to behave differently (require different strings for example). If you test without the comment line, and then submit with the comment line, you can run into trouble. Always submit exactly what you tested.

If customizations are not in effect, the background color of the GradeBot screen will be **light yellow.** However, if customizations are in effect, the background color of the GradeBot screen will turn **light green.**

3.9 Email Submission Rules

In some cases, I require you to submit your work by email. When email is involved, there are a few rules I need you to follow.

If your program violates the rules enough that grading becomes difficult, I will probably reply to your submission telling you what rules you violated and asking you to fix and resubmit.

3.9.1 To: Line

You can email to `doncolton2@gmail.com`. That is my preferred email address.

If you cannot use that, you are welcome to email to `don.colton@byuh.edu`. They both ultimately go the same place, so you do not need to send to both. Either one is fine.

3.9.2 Subject Line

The subject line of the email must be as follows:

```
cis205 label lastname, firstname
```

The reason for this rule is to facilitate the grading of submissions and the recording of grades. When I receive your email, it may be in the midst of many other emails from other students. I need to keep things straight so that I can record your grade properly.

The "label" part is replaced by the grading label for that assignment. The "lastname" part is replaced by your own last name. The "firstname" part is replaced by your own first name.

Also, the name parts are the names that you asked me to use for you. I use that name in my grade book.

When I am ready to record your grade, I scan down my grade book, which is sorted by lastname and firstname. If I do not see the lastname and firstname that you provided, it requires extra steps for me to verify which person should receive credit. I would prefer to have you do those extra steps instead of me.

So, for example, if I were submitting task p1 and my lastname were Colton and my firstname were Don, I would use this subject line:

```
cis205 p1 Colton, Don
```

3.9.3 Body When Submitting a Program

If you are submitting a program, the remainder of the email should be that program, and nothing else unless the assignment specifically requires it.

Do not send your program as an attachment. Send it directly in-line so I will see it immediately when I open your email.

The first line of your program must be a comment line that repeats the required subject line. This is to facilitate the recording of grades.

In Perl, comment lines start with `#`. So, following the previous example from above, I would have this comment line as the first line of my program.

```
# cis205 p1 Colton, Don
```

In other languages, the comment may be introduced by `//` or surrounded by `/*` and `*/` or follow some other convention. Use whatever is appropriate for your chosen language.

The second line of your program must be a comment line that specifies which language you are using. This is to facilitate grading.

If you were writing in Perl, you could use a line like this:

```
# language is Perl
```

The remaining lines should be your program itself.

The email must be in plain-text form. It may not be in html form or rich text form or in the form of an attachment. In plain text, there is no coloring to the letters. There is no bold or italics.

The reason for this rule is to facilitate testing of your program. When I receive your email, I may need to test it by running it. I do this by doing a copy-paste from your email into GradeBot (for instance) or into an empty program file. Then I run your program.

I should be able to use copy-paste to make a copy of your program for testing.

I do not accept programs sent as attachments because of the extra work it requires on my end.

You must avoid having each line of your program start with `>` or `>>` as is common when you are replying. Having those characters makes it impossible to copy-paste and run your program.

If your program includes long-line comments, emailing your program could result in line-breaks being added by your email client, which could cause your program to fail. Please take proper precautions.

You should avoid having anything else in your email that might make it difficult for me to decide what you intend as your program. I want to be able to assume that your whole email is the program you are submitting.

Specifically avoid including any “reply” comments, but if they are obviously not part of the program, they will be okay.

Specifically avoid a “signature”, but if it are obviously not part of the program, it will be okay.

3.9.4 How Grading Happens

Your program must pass a visual inspection and run properly. If either aspect fails, I will refer it back to you to fix the errors.

To visually inspect your program, I will check the structure of your program. Is it constructed as required? Does it follow my style requirements if any?

To run your program, I will cut and paste it into GradeBot and run it there myself. It must pass all tests.

3.10 Programming Style

From my point of view, style is mostly about making your program easy to read and update. That is my main concern.

If I complain about your style, what it probably means is I had trouble understanding your program. It may run fine in the computer, but it was difficult for me.

Your code should be readable to a programmer even if they are unfamiliar with your particular programming language. Most programming languages are similar enough that any programmer can read them. (Writing, on the other hand, can often require memorization of syntax rules.)

Use comments to explain what you are trying to accomplish with the key paragraphs of your program. Use comments to explain anything that might be hard to understand in the future.

Variable names should helpfully describe the contents they hold. Subroutine names should helpfully describe what they do.

Indenting:

Indenting should correctly reveal the internal structure of your program. It should be consistent and reasonable. I personally like indenting by two spaces for each additional level of nesting. I find that many students use four space indenting, perhaps because it comes that way naturally from Notepad++.

GradeBot has a built-in indent checker. It uses these simple rules: (1) Each time it encounters an opening curly brace, the indent count goes up by one. (2) Each time it encounters a closing curly brace, the indent count goes down by one. (3) Each time a new line starts, the number of spaces at the front of the line must be equal to the indent count times the indent size (like two or four spaces). GradeBot will tell you exactly where your indenting does not match its expectations.

When your code is properly indented, it is much easier for me to read, and I think it is probably easier for you to read as well. This probably results in fewer bugs.

Chapter 4

Programs Assigned

The following programs are hereby assigned. They will also be discussed in class. The programs are worth 300 points, plus extra credit for additional programs.

In some cases a program might be useful for when you take a test. See section 6.2 (page 79) for details.

Contents

4.1 p01: (20) Factors 56

4.2 p02: (20) Perfect Numbers 57

4.3 p03: (30) Fibonacci 58

4.3.1 Version 1: Recursion 58

4.3.2 Version 2: Memo-ization 59

4.4 p04: (20) Greatest Common Divisor 60

4.5 p05: (20) Prime Factors 62

4.6 Counting Terminology 63

4.7 p11: (20) Multiplication Rule 64

4.8 p13: (20) OSWOR 65

4.9 p14: (25) Choose 66

4.10 p15: (25) OWII 67

4.11 p21: (50) BST: Binary Search Tree 68

4.12 p22: (50) Huffman Coding 70

4.13 p23: (50ec) MST: Minimum Spanning Tree . . 71

4.14 p31: (50ec) PQ: Priority Queue 72

4.15 p32: (50ec) LCS: Longest Common Subsequence 73

4.16 p33: (50ec) Zoo 74
4.17 p34: (50ec) Zoo DB 75
4.18 p35: (50ec) nKrypto 76

General Information

Each program will be discussed in class, and will have a due date, a deadline, and a grading label.

Discussed means the date we talked about it in class.

Due Date means the date by which you must complete the assignment for it to be "on time." After that it is late, but it is still accepted for credit until the deadline.

Deadline means the date by which you must complete the assignment to receive credit for your work. After that it is not accepted for credit.

Grading Label means a short label I use in my grade book to track this activity for grading purposes.

Difficulty means how hard a problem seems to be, or how much time it will take to write it and debug it. The levels are trivial, easy, intermediate, and hard.

Retrieving Inputs

Several of the programs require you to receive inputs from standard input. For Perl and Java, here are some pieces of sample code.

Perl: `$x = <STDIN>;`

Java: `Scanner s = new Scanner(System.in); x = s.nextInt();`

Several of the programs require you to receive inputs from the command line argument vector instead of standard input. For Perl and Java, here are some pieces of sample code.

Perl: `$x = $ARGV[0];`

Java: `x = Long.parseLong(args[0]);`

4.1 p01: (20) Factors

- Discussed: Mon, Sep 08
- 20pt Deadline: Wed, Sep 10, 23:59
- 15pt Deadline: Fri, Sep 12, 23:59
- 10pt Deadline: Tue, Dec 09, 23:59
- Grading Label: **p01**

Difficulty: trivial

This is our "getting-started" task. It is a very simple program.

You receive a number, "n". Then you examine each number from one to n and tell whether it is a factor of "n" or not.

This is a GradeBot task. The general rules in section 3.8 (page 49) apply, including email subject line and program comment line.

`cis205 p01 lastname, firstname` is the required email subject line.

For Perl `# cis205 p01 lastname, firstname` is the required comment. Test the program you will submit, and submit the program you tested. Do not make changes unless you test again.

Gradebot can be found at http://gradebot.tk/
and at http://gbot.is2.byuh.edu/.

Reminder: The second line of your program must be a comment that says what language you are using to write this program. In Perl that would be:

```
# language:  Perl
```

4.2 p02: (20) Perfect Numbers

- Discussed: Wed, Sep 10
- 20pt Deadline: Fri, Sep 12, 23:59
- 15pt Deadline: Mon, Sep 15, 23:59
- 10pt Deadline: Tue, Dec 09, 23:59
- Grading Label: **p02**

Difficulty: trivial

This is a GradeBot task. The general rules in section 3.8 (page 49) apply, including email subject line and program comment line.

`cis205 p02 lastname, firstname` is the required email subject line.

For Perl `# cis205 p02 lastname, firstname` is the required comment. Test the program you will submit, and submit the program you tested. Do not make changes unless you test again.

Gradebot can be found at `http://gradebot.tk/`
and at `http://gbot.is2.byuh.edu/`.

This is an exercise in finding factors and adding them up. We define a "perfect" number as one whose proper factors add up to the original number.

Example: 6

6 has the proper factors 1, 2, and 3. (6 itself is also a factor, but we don't count that one.)

When you add up 1 + 2 + 3, you get 6.

Therefore, 6 is perfect.

Another perfect number is 28.

28 has factors 1, 2, 4, 7, and 14. 1 + 2 + 4 + 7 + 14 = 28.

If the factors add up to less than the number, we say it is "deficient."

10 is deficient because 1 + 2 + 5 = 8, and 8 is less than 10.

If the factors add up to more than the number, we say it is "excessive."

12 is excessive because 1 + 2 + 3 + 4 + 6 = 16, and 16 is more than 12.

So, identify and add up the factors, and compare the tally with the original number. Easy.

4.3 p03: (30) Fibonacci

- Discussed: Mon, Sep 22
- 30pt Deadline: Wed, Sep 24, 23:59
- 25pt Deadline: Fri, Sep 26, 23:59
- 20pt Deadline: Tue, Dec 09, 23:59
- Grading Label: **p03**

Difficulty: medium

This is a GradeBot task. The general rules in section 3.8 (page 49) apply, including email subject line and program comment line.

`cis205 p03 lastname, firstname` is the required email subject line.

For Perl `# cis205 p03 lastname, firstname` is the required comment. Test the program you will submit, and submit the program you tested. Do not make changes unless you test again.

Gradebot can be found at `http://gradebot.tk/`
and at `http://gbot.is2.byuh.edu/`.

The **Fibonacci** assignment builds on your introductory abilities by introducing two concepts: recursion and memo-ization.

4.3.1 Version 1: Recursion

The definition of the **Fibonacci** number series is this:

- `fib(1)=1` (basis case)
- `fib(2)=1` (basis case)
- `fib(n)=fib(n-1)+fib(n-2)` for higher numbers

For this class we require a recursive solution. You should have a subroutine named `fib` that calls itself recursively.

Your main program should do all the input and output. It should have one call to the `fib` subroutine to calculate the result, and then it should print that result. Your fib subroutine should make recursive calls to itself, each time using a smaller number.

Remember to stop the **recursion** when you call fib(2) or fib(1). The recursion must "bottom out" when it reaches the **basis case**.

Write this program and test it with GradeBot.

It will time out. (The running time grows exponentially, so it will not be able to complete its run because it takes more time than GradeBot will allow.)

4.3.2 Version 2: Memo-ization

Due to the nature of this particular problem, it turns out that we are recalculating the same result many times. That's where memo-ization comes in. (Memoization looks like memorization, but without the r.)

Create a static private array named `_fib` with size 46 or larger. If your language does not support static private, you can use a global variable `_fib` instead. (`_fib` is your memo pad.)

The point of using static variables here is that each time the subroutine is entered, the values already in the array will still be there. (Global variables are always static, with possible really rare exceptions.)

Local / Dynamic Variables: All **local variables** in your subroutines are freshly allocated each time the subroutine is activated (entered). If you call fib(10) and you put some value into a local variable x, then later when you call fib(8) you will not find that value in x because the x for fib(8) is different than the x for fib(10).

Global / Static Variables: All static or **global variables** in your subroutines are pre-allocated before your subroutine is activated (entered). If you call fib(10) and you put some value into a static variable x, then later when you call fib(8) you **WILL** find that value in x because the x for fib(8) is the same as the x for fib(10).

To use memoization, your fib subroutine should check to see if the requested value has already been calculated, and if so, simply return it. If not, it should make recursive calls to itself as before.

You may need some way to initialize the `_fib` array. In Perl, you can tell whether something is defined or not, and then do the proper initialization if it was not yet defined.

```
if ( defined $_fib[$n] ) { ... }
```

4.4 p04: (20) Greatest Common Divisor

- Discussed: Wed, Sep 24
- 20pt Deadline: Fri, Sep 26, 23:59
- 15pt Deadline: Mon, Sep 29, 23:59
- 10pt Deadline: Tue, Dec 09, 23:59
- Grading Label: **p04**

Difficulty: easy

This is a GradeBot task. The general rules in section 3.8 (page 49) apply, including email subject line and program comment line.

`cis205 p04 lastname, firstname` is the required email subject line.

For Perl `# cis205 p04 lastname, firstname` is the required comment. Test the program you will submit, and submit the program you tested. Do not make changes unless you test again.

Gradebot can be found at `http://gradebot.tk/`
and at `http://gbot.is2.byuh.edu/`.

See Section 4 (page 55) for a hint on how to retrieve inputs from the command line argument vector.

Use Euclid's algorithm to calculate the Greatest Common Divisor of two numbers.

You are required to present a recursive solution, where the gcd is calculated using a subroutine named gcd or something similar, and that subroutine calls itself as needed to simplify the task.

Try to make your program as short as possible. (I don't mean short variable names and no comments. I mean the smallest number of statements, and the simplest statements.)

See:

`http://en.wikipedia.org/wiki/Euclidean_algorithm#Concrete_example`

Classic Euclid (using subtraction):

The classic solution is based on subtraction.

If we have two numbers x and y, the gcd(x,y) can be calculated as follows.

Basis: If x and y are equal, then that value is the gcd.

Recursion: If they are not equal, we can subtract the smaller from the larger. Say x is larger than y, then gcd(x,y) is equal to gcd(x-y,y), which is an easier problem.

Modern Euclid (using mod):

A faster and more modern solution is based on the mod function, assuming % can be calculated efficiently.

Recursion: We can replace the larger parameter with larger % smaller, effectively subtracting the smaller as many times as possible, thus speeding up the whole process.

Debugging:

It may be smart to insert a print statement at the start of your gcd subroutine. Print out the values of the parameters that you received. GradeBot will ignore any lines that it was not expecting, if those lines appear to be comment lines. Specifically, if you have a line that begins with a hashtag and ends with a newline, GradeBot will print it out and ignore it, for up to 50 debug lines.

4.5 p05: (20) Prime Factors

- Discussed: Mon, Oct 13
- 20pt Deadline: Wed, Oct 15, 23:59
- 15pt Deadline: Fri, Oct 17, 23:59
- 10pt Deadline: Tue, Dec 09, 23:59
- Grading Label: **p05**

Difficulty: trivial

This is a GradeBot task. The general rules in section 3.8 (page 49) apply, including email subject line and program comment line.

`cis205 p05 lastname, firstname` is the required email subject line.

For Perl `# cis205 p05 lastname, firstname` is the required comment. Test the program you will submit, and submit the program you tested. Do not make changes unless you test again.

Gradebot can be found at `http://gradebot.tk/`
and at `http://gbot.is2.byuh.edu/`.

Prime Factorization

Try to make your program as short as possible. (I don't mean short variable names and no comments. I mean the smallest number of statements, and the simplest statements.)

You are given a positive integer (at STDIN). Report its prime factorization, smallest to largest. The product of a factorization is the original number. A prime factorization uses only prime numbers. A prime number has no exact divisors but itself and 1. All inputs are integers greater than 1.

For example, if you are given 12, you would print something like this (depending on what GradeBot tells you it wants):

```
The prime factor(s) of 12 are 2 2 3.
```

4.6 Counting Terminology

As we describe some of the following tasks, we use words like **selection**, **universe**, **sample**, and **distinct**. We briefly explain those words here.

By **selection** we mean a choice of one item from among all possible items that are available. If the possible items are “a b c” then we have three possible choices. If we choose “b” then “b” is our selection.

By **universe** we mean the set (or multi-set) from which possible items can be selected.

By **sample** we mean some selection from among the items in the universe.

By **distinct** we mean different. Sometimes we have identical items. If we cannot tell them apart, and one cannot be distinguished from the other, then we consider them to be identical. In the list “a a a” if we select “a” we cannot tell whether it is the first, second, or third “a” that was selected. We have only one distinct choice. In the list “a b c” we have three distinct choices. In the list “a b a” we have two distinct choices.

By **identical** we mean the opposite of distinct.

By **ordered** we mean that the order matters: “a b c” is different than “b a c”.

By **unordered** we mean that the order does not matter: “a b c” is the same as “b a c”.

By **replacement** we mean that after an item is selected, it can be selected again. From among “a b c” if we select three times **with** replacement, we could get “a b b”. If we select three times **without** replacement, we could not get “a b b” because the second “b” would not be available for selection again.

4.7 p11: (20) Multiplication Rule

- Discussed: Wed, Oct 15
- 20pt Deadline: Fri, Oct 17, 23:59
- 15pt Deadline: Mon, Oct 20, 23:59
- 10pt Deadline: Tue, Dec 09, 23:59
- Grading Label: **p11**

Difficulty: trivial

See Section 4 (page 55) for a hint on how to retrieve inputs from the command line argument vector.

This is a GradeBot task. The general rules in section 3.8 (page 49) apply, including email subject line and program comment line.

`cis205 p11 lastname, firstname` is the required email subject line.

For Perl `# cis205 p11 lastname, firstname` is the required comment. Test the program you will submit, and submit the program you tested. Do not make changes unless you test again.

Gradebot can be found at `http://gradebot.tk/`
and at `http://gbot.is2.byuh.edu/`.

We count the number of distinct assignments possible.

The model is assigning paint colors to houses in a neighborhood. (Each house is painted all the same color.)

If you have four houses, and three colors, then there are 81 possible assignments.

There are 3 possible assignments for the first house.

There are 3 possible assignments for the second house.

There are 3 possible assignments for the third house.

There are 3 possible assignments for the fourth house.

3 * 3 * 3 * 3 = 81.

This is known as the multiplication rule.

4.8 p13: (20) OSWOR

- Discussed: Mon, Oct 20
- 20pt Deadline: Wed, Oct 22, 23:59
- 15pt Deadline: Fri, Oct 24, 23:59
- 10pt Deadline: Tue, Dec 09, 23:59
- Grading Label: **p13**

Difficulty: easy

See Section 4 (page 55) for a hint on how to retrieve inputs from the command line argument vector.

This is a GradeBot task. The general rules in section 3.8 (page 49) apply, including email subject line and program comment line.

`cis205 p13 lastname, firstname` is the required email subject line.

For Perl `# cis205 p13 lastname, firstname` is the required comment. Test the program you will submit, and submit the program you tested. Do not make changes unless you test again.

Gradebot can be found at `http://gradebot.tk/`
and at `http://gbot.is2.byuh.edu/`.

Ordered Selections WithOut Replacement

Given a universe (the number of distinct objects among which selection is made) and a sample (the number of objects selected), tell how many distinct results are possible. Order matters: a b c is not the same as b c a.

For 26 3, we have 26 choices for the first selection, 25 choices for the second, and 24 choices for the third.

So, the answer is 26 * 25 * 24 = 15600.

(26!) / (23!) = 15600.

4.9 p14: (25) Choose

- Discussed: Wed, Oct 22
- 25pt Deadline: Fri, Oct 24, 23:59
- 20pt Deadline: Mon, Oct 27, 23:59
- 15pt Deadline: Tue, Dec 09, 23:59
- Grading Label: **p14**

Difficulty: easy

This is a GradeBot task. The general rules in section 3.8 (page 49) apply, including email subject line and program comment line.

`cis205 p14 lastname, firstname` is the required email subject line.

For Perl `# cis205 p14 lastname, firstname` is the required comment. Test the program you will submit, and submit the program you tested. Do not make changes unless you test again.

Gradebot can be found at `http://gradebot.tk/`
and at `http://gbot.is2.byuh.edu/`.

Unordered Selections WithOut Replacement

Given a universe (the number of distinct objects among which selection is made) and a sample (the number of objects selected), tell how many distinct results are possible. Order does not matter: a b c is the same as b c a.

For 26 3, we have 26 choices for the first selection, 25 choices for the second, and 24 choices for the third. But there are six ways each resulting sample could have been drawn (3 * 2 * 1).

So, the answer is (26 * 25 * 24) / (3 * 2 * 1) = 2600.

(26!) / (23! * 3!) = 2600.

Warning: The numbers can get pretty big, and most of the calculations cancel each other out. If you are careful, you can do the required calculations without doing the needless calculations. Also, there may be an “overflow” situation, where multiplying all the numbers together exceeds the “word size” (64 bits, for example) of the computer. So it may actually be necessary to avoid needless calculations.

4.10 p15: (25) OWII

- Discussed: Mon, Oct 27
- 25pt Deadline: Wed, Oct 29, 23:59
- 20pt Deadline: Fri, Oct 31, 23:59
- 15pt Deadline: Tue, Dec 09, 23:59
- Grading Label: **p15**

Difficulty: easy

See Section 4 (page 55) for a hint on how to retrieve inputs from the command line argument vector.

This is a GradeBot task. The general rules in section 3.8 (page 49) apply, including email subject line and program comment line.

`cis205 p15 lastname, firstname` is the required email subject line.

For Perl `# cis205 p15 lastname, firstname` is the required comment. Test the program you will submit, and submit the program you tested. Do not make changes unless you test again.

Gradebot can be found at `http://gradebot.tk/`
and at `http://gbot.is2.byuh.edu/`.

Orderings With Identical Items

Usage: "owii set1 set2 ..." where "set1" (etc) is the number of identical objects of type "1" among which selection is made.

For example, how many distinct ways can you arrange the letters a a b b c? This would be "owii 2 2 1" or 30.

Since there are five items, you have 5 * 4 * 3 * 2 * 1 ways to arrange them. But since two of those items are identical, you must divide by 2 * 1 to cancel out duplicates. And since two more are also identical, you must divide again.

(5 * 4 * 3 * 2 * 1) / (2 * 1 * 2 * 1 * 1) = 30.

(5!) / (2! * 2! * 1!) = 30.

4.11 p21: (50) BST: Binary Search Tree

- Discussed: Mon, Nov 03+
- 50pt Deadline: Wed, Nov 12, 23:59
- 45pt Deadline: Fri, Nov 14, 23:59
- 40pt Deadline: Tue, Dec 09, 23:59
- Grading Label: **p21**

Difficulty: hard

This is a GradeBot task. The general rules in section 3.8 (page 49) apply, including email subject line and program comment line.

`cis205 p21 lastname, firstname` is the required email subject line.

For Perl `# cis205 p21 lastname, firstname` is the required comment. Test the program you will submit, and submit the program you tested. Do not make changes unless you test again.

Gradebot can be found at `http://gradebot.tk/`
and at `http://gbot.is2.byuh.edu/`.

Additional requirement: Your program must have proper indenting as evaluated by GradeBot with the Indent Check flag turned on. See section 3.10 (page 52) for additional details.

Binary Search Trees are explained in Chapter 14 (page 138).

See `http://en.wikipedia.org/wiki/Binary_search_tree`

See `http://en.wikipedia.org/wiki/Tree_traversal`

Your program must build a Binary Search Tree using the elements given to it. Then, on demand, it must traverse that tree and report the elements of the tree in the order requested.

Commands are read from Standard Input.

The first command to handle is “add n” where n is a number to be added to the binary search tree.

Another command to handle is “find n” which should result in “found” if that object is found in the tree, and “not found” otherwise.

Another command to handle is “preorder” which should result in a **preorder** traversal of the tree, reporting each item as it is visited.

Another command to handle is “inorder” which should result in an **in-order** traversal of the tree, reporting each item as it is visited.

Another command to handle is “postorder” which should result in a **post-order** traversal of the tree, reporting each item as it is visited.

Another command to handle is “flush”, which means to throw away the current binary search tree and start fresh with an empty tree.

Another command to handle is “quit” which should result in termination of the program.

4.12 p22: (50) Huffman Coding

- Discussed: Wed, Nov 12+
- 50pt Deadline: Wed, Nov 26, 23:59
- 45pt Deadline: Mon, Dec 01, 23:59
- 40pt Deadline: Tue, Dec 09, 23:59
- Grading Label: **p22**

Difficulty: intermediate

This is a GradeBot task. The general rules in section 3.8 (page 49) apply, including email subject line and program comment line.

`cis205 p22 lastname, firstname` is the required email subject line.

For Perl `# cis205 p22 lastname, firstname` is the required comment. Test the program you will submit, and submit the program you tested. Do not make changes unless you test again.

Gradebot can be found at `http://gradebot.tk/`
and at `http://gbot.is2.byuh.edu/`.

Additional requirement: Your program must have proper indenting as evaluated by GradeBot with the Indent Check flag turned on. See section 3.10 (page 52) for additional details.

Huffman Coding is explained in Chapter 15 (page 144).

See `http://en.wikipedia.org/wiki/Huffman_coding`

Your program must read lines from Standard Input, one by one, until the line "`#`" is read.

Each line is of the form "(letter) occurs (count) times".

The required response is "Bits required for Huffman coding: (count)".

You are not asked to reveal the Huffman code you used, but all correct Huffman codes will have the same bit count.

4.13 p23: (50ec) MST: Minimum Spanning Tree

- Extra Credit
- 50pt Deadline: Tue, Dec 09, 23:59
- Grading Label: **p23**

Difficulty: hard

This is a GradeBot task. The general rules in section 3.8 (page 49) apply, including email subject line and program comment line.

`cis205 p23 lastname, firstname` is the required email subject line.

For Perl `# cis205 p23 lastname, firstname` is the required comment. Test the program you will submit, and submit the program you tested. Do not make changes unless you test again.

Gradebot can be found at `http://gradebot.tk/`
and at `http://gbot.is2.byuh.edu/`.

Minimum Spanning Trees are explained in Chapter 16 (page 148).

This task is an extra-credit opportunity.

See `http://en.wikipedia.org/wiki/Minimum_spanning_tree`

Your program must read lines from Standard Input, one by one, until the line "`##`" is read.

Each line represents an edge of a graph, and will be of the form "vertex1 vertex2 cost" where vertex1 and vertex2 are strings. Cost is an integer.

After all lines are read, you must discover a minimum spanning tree within the graph, and you must report its total cost.

4.14 p31: (50ec) PQ: Priority Queue

- Extra Credit
- 50pt Deadline: Tue, Dec 09, 23:59
- Grading Label: **p31**

Difficulty: easy (by list), intermediate (by heap)

This is a GradeBot task. The general rules in section 3.8 (page 49) apply, including email subject line and program comment line.

`cis205 p31 lastname, firstname` is the required email subject line.

For Perl `# cis205 p31 lastname, firstname` is the required comment. Test the program you will submit, and submit the program you tested. Do not make changes unless you test again.

Gradebot can be found at `http://gradebot.tk/`
and at `http://gbot.is2.byuh.edu/`.

Do these things: Add items to a priority queue. Tell which item is on top (maximum value). Remove the top item.

For CIS 205, you are allowed to do this by using a list, and sorting it frequently. The run-time for list insertion is O(n). The run-time for sorting is O(n lg n). This is less efficient than using a heap, but it is also much simpler.

For CIS 301, this must be done using a data structure known as a binary heap. The run-time for heap insertion is O(lg n), and for pop is O(lg n), and for peek is O(1).

4.15 p32: (50ec) LCS: Longest Common Subsequence

- Extra Credit
- 50pt Deadline: Tue, Dec 09, 23:59
- Grading Label: **p32**

Difficulty: hard

This is a GradeBot task. The general rules in section 3.8 (page 49) apply, including email subject line and program comment line.

`cis205 p32 lastname, firstname` is the required email subject line.

For Perl `# cis205 p32 lastname, firstname` is the required comment. Test the program you will submit, and submit the program you tested. Do not make changes unless you test again.

Gradebot can be found at `http://gradebot.tk/`
and at `http://gbot.is2.byuh.edu/`.

4.16 p33: (50ec) Zoo

- Extra Credit
- 50pt Deadline: Tue, Dec 09, 23:59
- Grading Label: **p33**

Difficulty: hard

This is a GradeBot task. The general rules in section 3.8 (page 49) apply, including email subject line and program comment line.

`cis205 p33 lastname, firstname` is the required email subject line.

For Perl `# cis205 p33 lastname, firstname` is the required comment. Test the program you will submit, and submit the program you tested. Do not make changes unless you test again.

Gradebot can be found at `http://gradebot.tk/`
and at `http://gbot.is2.byuh.edu/`.

4.17 p34: (50ec) Zoo DB

- Extra Credit
- 50pt Deadline: Tue, Dec 09, 23:59
- Grading Label: **p34**

Difficulty: hard

See Section 4 (page 55) for a hint on how to retrieve inputs from the command line argument vector.

This is a GradeBot task. The general rules in section 3.8 (page 49) apply, including email subject line and program comment line.

$\boxed{\texttt{cis205 p34 lastname, firstname}}$ is the required email subject line.

For Perl $\boxed{\texttt{\# cis205 p34 lastname, firstname}}$ is the required comment. Test the program you will submit, and submit the program you tested. Do not make changes unless you test again.

Gradebot can be found at `http://gradebot.tk/`
and at `http://gbot.is2.byuh.edu/`.

4.18 p35: (50ec) nKrypto

- Extra Credit
- 50pt Deadline: Tue, Dec 09, 23:59
- Grading Label: **p35**

Difficulty: hard

See Section 4 (page 55) for a hint on how to retrieve inputs from the command line argument vector.

This is a GradeBot task. The general rules in section 3.8 (page 49) apply, including email subject line and program comment line.

`cis205 p35 lastname, firstname` is the required email subject line.

For Perl `# cis205 p35 lastname, firstname` is the required comment. Test the program you will submit, and submit the program you tested. Do not make changes unless you test again.

Gradebot can be found at `http://gradebot.tk/`
and at `http://gbot.is2.byuh.edu/`.

How would you write a computer program to solve Krypto challenges?

`http://en.wikipedia.org/wiki/Krypto_(game)` explains the game.

`http://www.dreamshire.com/krypto.php` is an online Krypto solver.

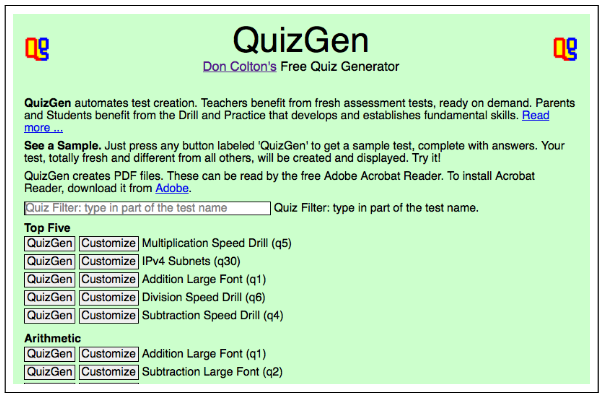
Chapter 5

QuizGen

QuizGen is a tool I created that I use to randomly generate exam problems for this course. If you wish, you can use it to generate sample problems (and answers) for practice.

This is an old version of the screen you see when you start QuizGen. You can find QuizGen at:

http://quizgen.tk/



QuizGen

Don Colton's Free Quiz Generator

QuizGen automates test creation. Teachers benefit from fresh assessment tests, ready on demand. Parents and Students benefit from the Drill and Practice that develops and establishes fundamental skills. Read more ...

See a Sample. Just press any button labeled 'QuizGen' to get a sample test, complete with answers. Your test, totally fresh and different from all others, will be created and displayed. Try it!

QuizGen creates PDF files. These can be read by the free Adobe Acrobat Reader. To install Acrobat Reader, download it from Adobe.

Quiz Filter: type in part of the test name. Quiz Filter: type in part of the test name.

Top Five

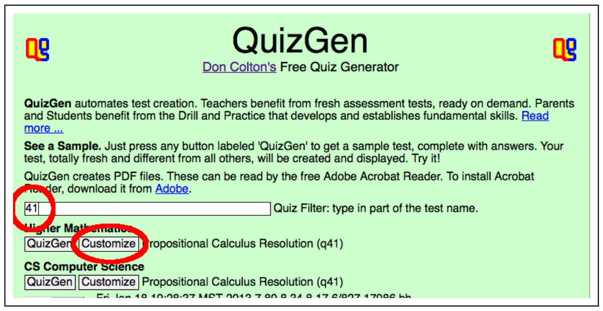
QuizGen Customize Multiplication Speed Drill (q5)
QuizGen Customize IPv4 Subnets (q30)
QuizGen Customize Addition Large Font (q1)
QuizGen Customize Division Speed Drill (q6)
QuizGen Customize Subtraction Speed Drill (q4)

Arithmetic

QuizGen Customize Addition Large Font (q1)
QuizGen Customize Subtraction Large Font (q2)

You can also find QuizGen at http://quizgen.doncolton.com/.

If, for example, you type 41 into the filter blank, it will reduce the menu until only those lines with 41 in them remain.



QuizGen

Don Colton's Free Quiz Generator

QuizGen automates test creation. Teachers benefit from fresh assessment tests, ready on demand. Parents and Students benefit from the Drill and Practice that develops and establishes fundamental skills. Read more ...

See a Sample. Just press any button labeled 'QuizGen' to get a sample test, complete with answers. Your test, totally fresh and different from all others, will be created and displayed. Try it!

QuizGen creates PDF files. These can be read by the free Adobe Acrobat Reader. To install Acrobat Reader, download it from Adobe.

41 Quiz Filter: type in part of the test name.

Higher Mathematics

QuizGen Customize Propositional Calculus Resolution (q41)

CS Computer Science

QuizGen Customize Propositional Calculus Resolution (q41)

Chapter 6

Skills Tests

Contents

6.1 Practice Tests 78
6.2 Write Your Own Program 79

There are seven skills tests. Each test is discussed in this study guide, and skills are taught to enable you to take and pass each test.

The skills tests are worth 400 points, as follows. Some tests have bonus problems worth additional points for extra credit.

S1, 60pt, Formal Logic Resolution, is covered in Chapter 8 (page 89).

S2, 50pt, Big Oh Analysis, is covered in Chapter 9 (page 96).

S3, 50pt, Counting, is covered in Chapter 11 (page 113).

S4, 60pt, Conditional Probability, is covered in Chapter 13 (page 122).

S5, 60pt, Binary Search Trees, is covered in Chapter 14 (page 138).

S6, 60pt, Huffman Coding, is covered in Chapter 15 (page 144).

S7, 60pt, Minimum Spanning Trees, is covered in Chapter 16 (page 148).

6.1 Practice Tests

I provide practice tests that you can take before the actual test.

These practice tests have the exact same kinds of problems as the real test does, but normally lots more of them.

These practice tests will tell you whether your answer is correct or not, right after you enter your answer.

These practice tests will let you do a “reset” to start over so you can practice again.

6.2 Write Your Own Program

The tests are intended to be easy enough that they can be taken without any computing aids, other than the built-in calculator (ezCalc, see section 2.3.3, page 40) that I provide for some of them.

But beyond knowing how to do something yourself, I find it even more impressive if you can write a program to do it. For that reason, I will allow you to use a program to assist you in taking any of my skills tests. You could, for instance, copy-paste some or all of my question into your program, and then copy-paste some or all of your answer back into my test.

The practice tests will give you a chance to debug your program before taking the real test.

If you decided to write your own program, here are the rules:

The Main Rule: Your program must have been totally written by you, without using any special code libraries. You are allowed to get ideas (but not code) from the web or from others.

Local: Your program must run locally from the command line (cmd.exe) on the MS Windows computer where you are taking the actual test. It may not use the web in any way.

Pre-Approval: Before the test starts, you must email me a source-code copy of your program. I will review it and reply giving my authorization for you to use that program during the test.

If you forget to email me in advance, you may still be able to show it to me on the day of the test and get my approval at that time.

Language: Your program must be written in a language I can understand. These are okay: Perl, Java, C, and C++. If you are thinking of using a different language, ask me in advance.

If these rules are preventing you from doing something that you think should be allowed, please confer with me. If your request seems reasonable, I will fix the rules.

Chapter 7

Tutorial on Formal Logic

Contents

7.1 Modus Ponens 82

7.2 Modus Tollens 82

7.3 Atomic Statements 82

7.4 Basic Operators 83

7.4.1 And 83

7.4.2 Or 84

7.4.3 Not 85

7.5 Additional Operators 85

7.5.1 Implies 85

7.5.2 Equivalence 86

7.6 Truth Tables and Proof 86

7.7 DeMorgan's Laws 87

7.8 Normal Forms 87

7.8.1 CNF: Conjunctive Normal Form 88

7.8.2 DNF: Disjunctive Normal Form 88

Formal Logic involves deductions that can be made from sets of statements.

Important words and concepts: and, or, not, xor, implies, equivalence, DeMorgan, tautology, contradiction, commutativity, associativity, wff (well-formed formula), truth tables, modus ponens, modus tollens.

7.1 Modus Ponens

Modus Ponens (MP) is the most famous example of Formal Logic. If you have two facts, A implies B, and A is True, then MP makes the deduction that B must be True.

For example, let's say we have two facts: (a) It is true that "I worked hard in this class" implies "I will receive a grade of A in this class," and (b) It is true that "I worked hard in this class."

We let A stand for "I worked hard in this class."

We let B stand for "I will receive a grade of A in this class."

We can restate our two facts as: (a) It is true that A implies B, and (b) A is true.

According to Modus Ponens, we can deduce that B is true. Specifically, "I will receive a grade of A in this class."

Modus Ponens is short for Modus Ponendo Ponens, which is Latin for "the way (modus) that affirms (ponendo) by affirming (ponens)"

`http://en.wikipedia.org/wiki/Modus_ponens` tells more.

7.2 Modus Tollens

Modus Tollens (MT) is the other famous example. If you have two facts, A implies B, and B is False, then MT makes the deduction that A must be False.

Modus Tollens is short for Modus Tollendo Tollens, which is Latin for "the way (modus) that denies (tollendo) by denying (tollens)"

`http://en.wikipedia.org/wiki/Modus_tollens` tells more.

7.3 Atomic Statements

The fundamental basis of Formal Logic is the statement. In Formal Logic, each statement is either True or False. In the real world, statements may not be so clear-cut. They may be ambiguous. They may be self-contradictory. Formal Logic describes a useful subset of the real world, but not the whole thing.

"This statement is False" is a classic example of a self-contradictory statement. It cannot be accurately represented in Formal Logic.

Atomic Statements are statements that **are not** composed of other statements. Atomic Variables are words or symbols each of which represent an Atomic Statement. The phrases "Atomic Statement", "Atomic Variable", and simply "Atomic" can be used interchangeably to mean the same thing.

Compound Statements are statements that **are** composed or constructed from other statements. This is done by using operators.

The truth value of a Compound Statement can be derived by looking at the truth values of the Atomic Statements of which it is constructed.

7.4 Basic Operators

Connecting the statements of Formal Logic, we have several well-known operators. The most important of these are And, Or, and Not.

The And and Or operators are **associative** and **commutative**, by which we mean for a given string of Ands, it does not matter what order they are in, or in what order they are evaluated. The same is true for Ors.

7.4.1 And

When two statements are combined by the word And, the resulting statement is True when both of the original statements are True. Otherwise, the resulting statement is False.

For any number of statements, And is presumed to be True unless one or more of the statements is False. Then And becomes False.

And is also called Conjunction.

And is often represented by the symbol $\wedge$, pronounced "and" or "wedge."

We can express the meaning of And by showing a Truth Table. We list all possible assignments of True and False to A and B. There are four of them, as shown in the truth table.

A	B	$A \wedge B$
T	T	**T**
T	F	**F**
F	T	**F**
F	F	**F**

7.4.2 Or

When two statements are combined by the word Or, the resulting statement is True when either one of the original statements is True. When both of them are False, the resulting statement is False.

For any number of statements, Or is presumed to be False unless one or more of the statements is True. Then Or becomes True.

Or is also called Disjunction.

Or is often represented by the symbol $\vee$, pronounced "or" or "vee."

We can express the meaning of Or by showing a Truth Table.

A	B	$A \vee B$
T	T	**T**
T	F	**T**
F	T	**T**
F	F	**F**

Xor

The above version of Or is sometimes called and/or, meaning either or both.

There is another version of Or, called Xor, for Exclusive Or, meaning either but not both.

For any number of statements, Xor is True if the number of true statements is odd, and False if the number of true statements is even.

Xor is also called Parity.

We can express the meaning of Xor by showing a Truth Table.

A	B	A xor B
T	T	**F**
T	F	**T**
F	T	**T**
F	F	**F**

7.4.3 Not

For exactly one statement, Not changes the truth value from True to False, or from False to True. If A is True, then Not A is False. If A is False, then Not A is True. This is also called Dichotomy, which means exactly two options (True and False). As mentioned above, the Real World does not always bless us with Dichotomy, but it happens often enough to be useful.

Not is also called Negation.

Not is represented several different ways: A', $-A$, and $\bar{A}$.

We can express the meaning of Not by showing a Truth Table. Because only one variable is involved, we only need two rows.

A	$-A$
T	**F**
F	**T**

7.5 Additional Operators

There are other commonly-used operators. These include Implies and Equivalent.

7.5.1 Implies

When we say that A Implies B, we mean that any time A is True, B is also True. When A is False, we know nothing about B.

Implies is often represented by the symbol $\rightarrow$ (as in $A \rightarrow B$) or by the symbol $\Rightarrow$.

We can express the meaning of Implies by showing a Truth Table.

A	B	$A \to B$
T	T	**T**
T	F	**F**
F	T	**T**
F	F	**T**

People often find it confusing that when A is False, Implies is always True. But, by convention, that is the way Implies is defined.

The mere fact that A Implies B does not mean that A "causes" B. They may both be caused by some other thing, "C". Or maybe causation is not involved at all.

7.5.2 Equivalence

Equivalence: When two statements always have the same Truth value, we say that they are equivalent.

Equivalence is often represented by the symbol $\equiv$.

We can express the meaning of Equivalence by showing a Truth Table.

A	B	$A \equiv B$
T	T	**T**
T	F	**F**
F	T	**F**
F	F	**T**

7.6 Truth Tables and Proof

An equivalent way to write A Implies B is: Not A Or B.

$(A \to B) \equiv (-A \vee B)$

Let's prove it.

To verify that two statements are equivalent, we can construct a Truth Table listing all possible assignments of True and False to the atomic variables.

If there are N atomic statements, then there are 2^N possible assignments of True and False to those atomic statements.

A Truth Table will have 2^N rows, with each row representing one assignment of truth values to the atomic variables.

To prove $(A \rightarrow B) \equiv (-A \vee B)$, we can construct the following Truth Table:

A	B	$A \rightarrow B$	$-A$	$-A \vee B$
T	T	**T**	F	**T**
T	F	**F**	F	**F**
F	T	**T**	T	**T**
F	F	**T**	T	**T**

Since the third column $(A \rightarrow B)$ and the fifth column $(-A \vee B)$ have the same truth value (TFTT), the statements are equivalent.

7.7 DeMorgan's Laws

DeMorgan's laws deal with how a "not" outside of parentheses can be distributed across the pieces that are inside the parentheses.

Let's look at "not (A and B)".

DeMorgan changes this into "not A or not B".

Let's look at "not (A or B)".

DeMorgan changes this into "not A and not B".

DeMorgan's laws can easily be proved using truth tables.

Exam Question 1 (p.155)**:**
Use Truth Tables to Prove DeMorgan's Laws.

Acceptable Answer:
Construct the appropriate truth table.

7.8 Normal Forms

Taken as a whole, all the statements we know to be True are called our Knowledge Base, sometimes abbreviated KB.

Because there are equivalent ways of writing many things, things that look different from each other can actually still have the same ultimate truth value and meaning.

To grapple with that variability, several ways of writing things have been singled out as Normal Forms. Any KB can be restated in any of these Normal Forms.

Normal Forms are also called Canonical Forms.

It turns out that all Formal Logic operators (like Implies) can be restated using just the three main operators, And, Or, and Not.

7.8.1 CNF: Conjunctive Normal Form

CNF: Conjunctive Normal Form is when we state our knowledge as a series of conjunctions.

$A \wedge B \wedge C \wedge D \wedge E \wedge F \wedge G \wedge H ...$

Each part of the conjunction is a statement asserted to be true.

Within each of those parts, typically there are alternatives, at least one of which must be true. Each part is therefore a disjunction.

$(A1 \vee A2 \vee A3) \wedge (B1 \vee B2) \wedge C \wedge ...$

Thus, CNF is a conjunction of disjunctions.

http://en.wikipedia.org/wiki/Conjunctive_normal_form tells more.

7.8.2 DNF: Disjunctive Normal Form

DNF: Disjunctive Normal Form is when we state our knowledge as a series of disjunctions.

CNF is generally more useful, at least for our purposes.

Chapter 8

S1: Resolution

- Points: 60
- Exam 1: Fri, Sep 19, 09:00 to 09:40 (40m)
- Exam 2: Fri, Sep 26, 09:00 to 09:40 (40m)
- Exam 3: Wed, Dec 10, 07:10 to 09:50 (160m)
- Grading Label: **S1**

Contents

8.1 Basics 90
8.2 CNF Notation 91
8.3 Clause Simplification 91
8.4 Reduction 91
8.5 A or Not A 92
8.6 Resolution 92
8.7 Common Mistake 92
8.8 Procedure 93
8.9 Example 93
8.10 Required Format for Grading 93
8.11 Typical Exam 94
8.12 Resolution Solver 94

This chapter prepares you for the S1 (Skills 1) exam, Formal Logic and Resolution.

QuizGen (chapter 5, page 77) quiz q41 provides exam material for **S1**.

Easy: We do eight resolutions using three variables. Hard: We do three resolutions using four variables. There is 10% extra credit available.

Modus Ponens and Modus Tollens are classic rules of Formal Logic. There are many.

http://en.wikipedia.org/wiki/History_of_logic has a history of formal logic.

Fortunately, rather than memorize all those classic rules, there is a single method that always works in every case. It was invented by John Alan Robinson, a mathematician, in 1965 and is called Resolution.

http://en.wikipedia.org/wiki/Resolution_(logic) talks more about resolution.

http://en.wikipedia.org/wiki/J._Alan_Robinson tells about Dr. Robinson. As of 2012 he is still alive.

The resolution exam is introduced as follows.

Given a list of TRUE statements, simplify it as much as possible by using logic.

8.1 Basics

We will express our knowledge in CNF, conjunctive normal form.

The list of TRUE statements is called the knowledge base (the KB). We will add and delete statements to improve the list.

Each statement is called a (disjunctive) clause, and consists of an or-list of simple propositions. If it is in the KB, it is supposed to be TRUE.

Each simple proposition is either TRUE or FALSE. Put another way, for every proposition “p”, the clause “p or not p” must be TRUE.

“I am dry” is a simple proposition, and “I have an umbrella” is a simple proposition. Typically we abbreviate propositions down to letters or short words for convenience. The statement “I am dry, or I don’t have an umbrella” would be written as “dry or not umbrella” or “dry -umb” or even (d -u). Each such statement is called a clause. As we said, every clause in the KB is asserted to be TRUE.

8.2 CNF Notation

We will express our Knowledge Base like this:

(-a b) (a b) (a c)

By this we mean there are three clauses. The first is (-a b). The second is (a b). The third is (a c). They are implicitly joined together by $\wedge$, with this meaning:

(-a b) $\wedge$ (a b) $\wedge$ (a c)

Each of the clauses is meant as a disjunction. Thus, the KB has this meaning:

(-a $\vee$ b) $\wedge$ (a $\vee$ b) $\wedge$ (a $\vee$ c)

The minus signs mean Not.

8.3 Clause Simplification

We can drop any proposition we know to be FALSE from a clause because the whole clause must be TRUE, and the FALSE part clearly isn't helping. This is exactly like regular math where adding zero or multiplying by one does not change anything. If x+0=5 we can drop the +0 and be left with x=5. If (a b c) is TRUE, and we find out somehow that (-a) is TRUE, meaning that "a" is FALSE, then we can drop the "a" part, leaving (b c).

8.4 Reduction

Adding more propositions to a disjunctive clause (an or-list) cannot change it from TRUE to FALSE. That is the nature of OR. If we know (b) is TRUE, then (b c) is also TRUE, no matter what "c" is. As a direct result, if we have two clauses in the KB, and the little one is an exact subset of the big one, we can throw away the big one without losing any information. For example, if we know (I can drive), it does not help to also say (I can drive) or (I am rich). It provides no information about my riches.

8.5 A or Not A

If we have a clause that includes "(a) or (not a)" within it, we can delete it from the KB. The clause provides no information, since we already know that (a -a) is always TRUE.

8.6 Resolution

New clauses can be created from old clauses. This part is a little tricky, but it is very powerful. It is the heart of the Resolution method.

If we have the clause (a x), and we also have the clause (-a y), we can combine them to create a new clause. Notice that one includes (a) in its or-list, and the other includes the opposite, (-a).

By basic rule 1, "a" is either TRUE or FALSE. Here is the tricky part. IF "a" is TRUE, then "not a" must be FALSE, so the second clause simplifies into (y) is TRUE. On the other hand, if "a" is FALSE, then the first clause simplifies into (x) is TRUE.

We can splice the original clauses together to say (x y). More generally, if we have (lots of things) or "a", and (other things) or "not a", we can combine them into (lots of things) or (other things). That's resolution.

8.7 Common Mistake

For resolution, we need exactly one statement that is true in one clause and false in the other clause.

Sometimes we have two such statements.

Say we have (a b c d e) (-a -b c d e).

It is tempting, but incorrect, to try to "splice" on (a b) and deduce (c d e).

The opposite of (a) is (-a), but the opposite of (a b) is not (-a -b).

The correct splicing would be as follows:

(a b c d e) (-a -b c d e) can be spliced along "a".

The first part gives us (b c d e). The second part gives us (-b c d e). The combination gives us (-b b c d e).

Sadly, the (-b b) part makes this a useless statement.

(a b c d e) (-a -b c d e) can also be spliced along (b), but the result is similarly useless.

Bottom line: if there are two (or more) opposites, the result will not be helpful.

8.8 Procedure

To simplify the KB, we look at pairs of clauses. If one is a subset of the other, we delete the bigger one. If they inversely share a proposition (one has “a” and the other has “-a”) we use resolution to generate a new clause and insert it into the KB (unless the new clause was already there). We continue looking at all pairs of clauses until no more insertions or deletions can be made. Careful ordering can make the job faster. But whatever order you do things, you will always get the same result in the end.

8.9 Example

Given (-a b) (a b) (a c), resolve.

Solution: We know that (-a b) and (a b) resolve to (b).

Then, we know that (b) renders (anything b) useless.

Therefore, the solution is: (a c) (b).

8.10 Required Format for Grading

We call our required format **sorted CNF**.

All questions will be given to you in sorted CNF order.

Sorted CNF means conjunctive normal form where each clause is also internally sorted. The sorting order is - then [a-z]. Also the whole clauses are further sorted into alphabetical order.

For ease of grading, we require that your answers be given back to us in sorted CNF order.

The browser will verify that you have a single space between clauses. The browser will verify that you have a single space before and after each proposition.

The browser will **not** verify that you sorted things properly.

As a result, every correct answer will look exactly like the answer we calculated, and can be graded automatically by exact match.

8.11 Typical Exam

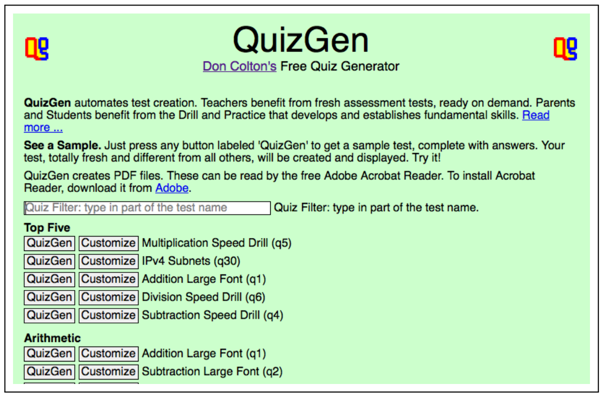
The S1 exam will probably consist of 11 questions. You are expected to get 10 correct. 8 will use three variables (a b c), and 3 will use four variables (a b c d).

It commonly takes about 4 minutes per problem to complete the exam.

8.12 Resolution Solver

QuizGen (chapter 5, page 77) is the tool that I use to generate Resolution problems. The Resolution generator also has a feature that allows it to solve a problem provided by you.

This is the screen you see when you start QuizGen.



QuizGen

Don Colton's Free Quiz Generator

QuizGen automates test creation. Teachers benefit from fresh assessment tests, ready on demand. Parents and Students benefit from the Drill and Practice that develops and establishes fundamental skills. Read more ...

See a Sample. Just press any button labeled 'QuizGen' to get a sample test, complete with answers. Your test, totally fresh and different from all others, will be created and displayed. Try it!

QuizGen creates PDF files. These can be read by the free Adobe Acrobat Reader. To install Acrobat Reader, download it from Adobe.

Quiz Filter: type in part of the test name Quiz Filter: type in part of the test name.

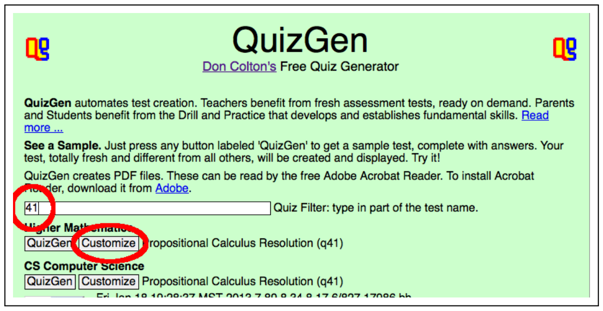
Top Five

QuizGen Customize Multiplication Speed Drill (q5)
QuizGen Customize IPv4 Subnets (q30)
QuizGen Customize Addition Large Font (q1)
QuizGen Customize Division Speed Drill (q6)
QuizGen Customize Subtraction Speed Drill (q4)

Arithmetic

QuizGen Customize Addition Large Font (q1)
QuizGen Customize Subtraction Large Font (q2)

If you type 41 into the filter blank, it will reduce the menu until only those lines with 41 in them remain. Select the Customize button.



QuizGen
Don Colton's Free Quiz Generator

QuizGen automates test creation. Teachers benefit from fresh assessment tests, ready on demand. Parents and Students benefit from the Drill and Practice that develops and establishes fundamental skills. Read more ...

See a Sample. Just press any button labeled 'QuizGen' to get a sample test, complete with answers. Your test, totally fresh and different from all others, will be created and displayed. Try it!

QuizGen creates PDF files. These can be read by the free Adobe Acrobat Reader. To install Acrobat Reader, download it from Adobe.

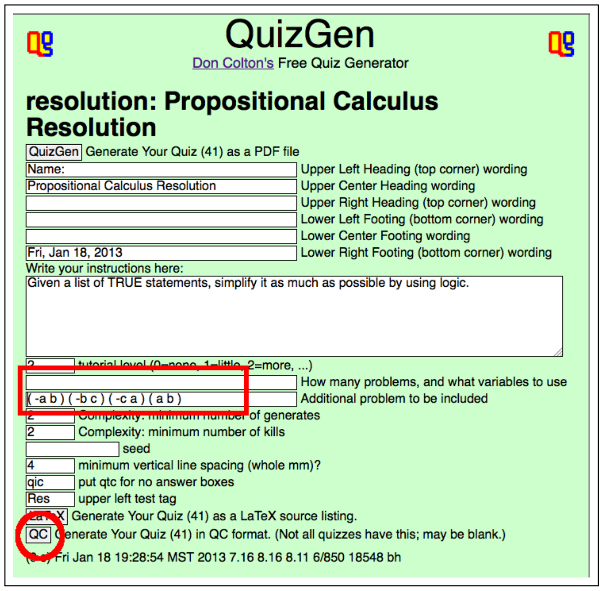
41 Quiz Filter: type in part of the test name.

Higher Mathematics
QuizGen Customize Propositional Calculus Resolution (q41)

CS Computer Science
QuizGen Customize Propositional Calculus Resolution (q41)

Fri Jan 18 19:28:37 MST 2013 7.80 8.24 8.17 6/827 17086 bh

Clear out the blank for how many problems, and what variables to use. Fill in the blank for Additional problems to be included. Then press the QC button.



QuizGen
Don Colton's Free Quiz Generator

resolution: Propositional Calculus Resolution

QuizGen Generate Your Quiz (41) as a PDF file

Name:	Upper Left Heading (top corner) wording
Propositional Calculus Resolution	Upper Center Heading wording
	Upper Right Heading (top corner) wording
	Lower Left Footing (bottom corner) wording
	Lower Center Footing wording
Fri, Jan 18, 2013	Lower Right Footing (bottom corner) wording

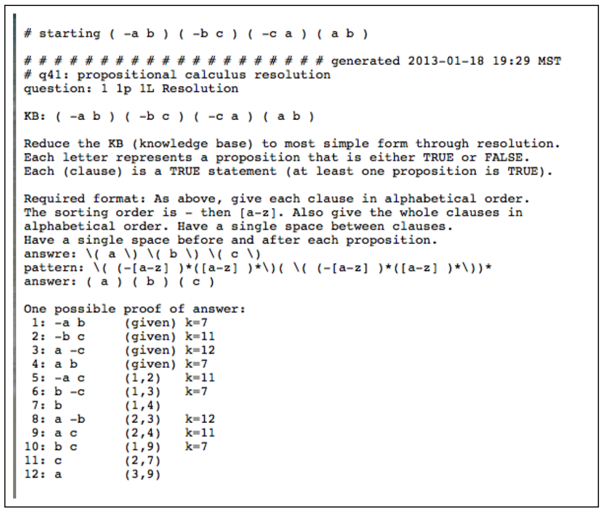
Write your instructions here:
Given a list of TRUE statements, simplify it as much as possible by using logic.

2	tutorial level (0=none, 1=little, 2=more, ...)
	How many problems, and what variables to use
(-a b) (-b c) (-c a) (a b)	Additional problem to be included
2	Complexity: minimum number of generates
2	Complexity: minimum number of kills
	seed
4	minimum vertical line spacing (whole mm)?
qic	put qtc for no answer boxes
Res	upper left test tag

LaTeX Generate Your Quiz (41) as a LaTeX source listing.
QC Generate Your Quiz (41) in QC format. (Not all quizzes have this; may be blank.)

(c-c) Fri Jan 18 19:28:54 MST 2013 7.16 8.16 8.11 6/850 18548 bh

A quiz will be generated using your problem. The nice thing, however, is that the quiz also includes an answer, and proof of that answer. This lets you solve problems, or see how to solve them.



```
# starting ( -a b ) ( -b c ) ( -c a ) ( a b )

# # # # # # # # # # # # # # # # # # # # generated 2013-01-18 19:29 MST
# q41: propositional calculus resolution
question: 1 1p 1L Resolution

KB: ( -a b ) ( -b c ) ( -c a ) ( a b )

Reduce the KB (knowledge base) to most simple form through resolution.
Each letter represents a proposition that is either TRUE or FALSE.
Each (clause) is a TRUE statement (at least one proposition is TRUE).

Required format: As above, give each clause in alphabetical order.
The sorting order is - then [a-z]. Also give the whole clauses in
alphabetical order. Have a single space between clauses.
Have a single space before and after each proposition.
answre: \( a \) \( b \) \( c \)
pattern: \( (-[a-z] )*([a-z] )*\)( \( (-[a-z] )*([a-z] )*\))*
answer: ( a ) ( b ) ( c )

One possible proof of answer:
 1: -a b      (given) k=7
 2: -b c      (given) k=11
 3: a -c      (given) k=12
 4: a b       (given) k=7
 5: -a c      (1,2)   k=11
 6: b -c      (1,3)   k=7
 7: b         (1,4)
 8: a -b      (2,3)   k=12
 9: a c       (2,4)   k=11
10: b c       (1,9)   k=7
11: c         (2,7)
12: a         (3,9)
```

Chapter 9

S2: Big Oh Analysis

- Points: 50
- Exam 1: Wed, Oct 01, 09:10 to 09:40 (30m)
- Exam 2: Fri, Oct 03, 09:10 to 09:40 (30m)
- Exam 3: Wed, Dec 10, 07:10 to 09:50 (160m)
- Grading Label: **S2**

Contents

9.1 What Are We Trying To Do? 97
9.2 Introduction 98
9.3 n Times As Much Input 98
9.3.1 Typical Algorithms 98
9.3.2 Constant-Time Algorithms 98
9.3.3 Linear-Time Algorithms 99
9.3.4 Logarithmic-Time Algorithms 99
9.3.5 Root-n-Time Algorithms 100
9.3.6 Exponential-Time Algorithms 100
9.4 Loops 101
9.4.1 Counting Up to n 101
9.4.2 Counting Down from n 101
9.4.3 Add/Subtract any Constant 101
9.4.4 Multiplying or Dividing 102
9.4.5 Unusual Limits 102
9.5 Combinations of Algorithms 102

9.5.1 Sequences of Statements 103
9.5.2 If-Else Constructs 103
9.5.3 Worst Case Running Time 103
9.5.4 Nested Blocks 103
9.5.5 Recursive Subroutines 104
9.6 And the Winner Is 104
9.7 Typical Exam 105

This chapter prepares you for the S2 (Skills 2) exam, Big Oh Analysis, which introduces the analysis of algorithms to determine their running time.

QuizGen (chapter 5, page 77) quiz q13 provides exam material for **S2**.

Easy: We do six programs that are each about a half page long. Hard: We do five programs that are each about a full page long. There is 10% extra credit available.

Welcome to **A Quick Guide to Big Oh.**

9.1 What Are We Trying To Do?

The amount of collected data in the world gets bigger every day. Credit card transactions. Log files. Web hits. Customer lists. Bigger. More.

One programming challenge is to build systems that do not fall to their knees under the weight of higher speeds and bigger transaction counts. We want systems that are robust, systems that degrade gracefully rather than collapse and melt down.

"Big Oh" analysis is that branch of computer science that measures program performance by simply looking at the program itself. There is a lot one can tell by looking at the code. We cannot easily measure the actual speed in seconds, but we **can** tell whether doubling the input will double the running time, or whether things will be worse or better.

There are some algorithms that are more work to program, but give a faster program. Programmers must choose the best approach to their task, given what they know of the future loads their program must support.

9.2 Introduction

"Big Oh" is the popular name for running-time analysis of algorithms. It is generally acknowledged that although you can buy more memory or a faster CPU chip, these things will not save you if you are running an inefficient algorithm. Computer Science students learn this material (and more) in their introductory courses. It is helpful for IS students to also have a grasp of the basic terminology and to have the ability to measure (in Big Oh fashion) the running time of various programs.

The words "Big Oh" have reference to "on the Order of," or "Order of magnitude." Specifically it is applied to running times of programs. As the input grows, what happens to the running time of the program?

The phrase "Big Oh" itself is used loosely here. Precisely it means that the algorithm runs at least that fast. Theta (Θ) is a more precise term used in Computer Science, but we will use somewhat less precise but much more familiar terminology, making "tight big oh" technically equivalent to "theta."

9.3 n Times As Much Input

We want to know what happens to the running time of our program if we get n times as much input data. For each algorithm, just by looking at the program code, we can come to some reliable conclusions. We imagine n to be really large. Thousands. Millions. Billions.

9.3.1 Typical Algorithms

In this section we talk about some typical algorithms and tell what their big oh running time would be.

9.3.2 Constant-Time Algorithms

An example of a constant-time algorithm would be one to pick the first number from a list. It does not matter how long the list is. In one step we can pick the first number and then stop.

If we have n times as much input, the running time does not change. Or, it "changes" by a factor of one. When the running time does not change, we say the algorithm is $\Theta(1)$, "theta one," "big oh one," or "order one," or constant.

A real-life example would be selecting a new employee by taking the first application on the pile. It would not matter how many applications were on the pile.

(On my Big-Oh quizzes, simple statements run in $\Theta(1)$ time. That is why they are called simple.)

Constant-time algorithms are the best possible algorithms, unless that time is very long.

9.3.3 Linear-Time Algorithms

Let's say we want to find the biggest number in a list, when the list is in unpredictable order. To find the biggest number, we must look at each entry in the list.

If we have n times as much input, the running time is n times longer. Such an algorithm is called $\Theta(n)$, "theta n," "order n," or "linear."

Linear algorithms are very common in IS programming, and are generally accepted as being efficient, unless there is a known way to do the job faster.

9.3.4 Logarithmic-Time Algorithms

Logarithmic algorithms have running times that grow more slowly than the size of the input. Double the input and the running time only gets a little longer. It does not double.

The classic example of a log-time algorithm is binary search. Take the (in)famous "guess my number" game. In this game, I think of a number and you must guess it. On each turn, you make a guess and I tell you whether you are too high, too low, or just right. If my number is between 1 and 100, your first guess may be 50. By guessing 50, you cut in half the number of remaining possibilities. Say my number is 78, but you don't know that. You say 50. I say higher. You say 75. I say higher. You say 88. I say lower. You say 81. I say lower. Each step you narrow the possibilities by half (roughly).

In this game, if we were to double the initial range, making it between 1 and 200, would it take you twice as many guesses? No. One extra guess at the front would determine whether it was above or below 100. From there, we are back to the same original challenge.

If we have n times as much input (meaning n times as many numbers to search), it will take us $\log_2 n$ steps before we get down to the original input size. Algorithms that run in log time are said to have a running time of $\Theta(\lg n)$, "theta log n," "big oh log n," or simply "log n."

9.3.5 Root-n-Time Algorithms

Root-n algorithms run in time proportional to the square root of n (the input size). An example would be finding whether a number n is prime. To be prime, a number must not be the mathematical result of multiplying too smaller numbers. To find if a number is prime, we can test all the smaller numbers to see if they divide exactly into n. But there is a trick. If n is 101, we can stop when we have tested 10, because if 11 goes in, then 101 = 11 * a, and a must be smaller than 11. But since we have tested all the numbers smaller than 11, we can quit. Without even trying it, we know 11 could not work. Such an algorithm would have running time $\Theta(\sqrt{n})$, or "root n."

9.3.6 Exponential-Time Algorithms

If you are just preparing for the quiz, you can skip this section. There are no exponential-time algorithms on the quiz.

Just as logarithmic algorithms are not much affected by a doubling of the input, there are other algorithms that may work well up to a point, but then the running time seems to explode.

The classical example of an exponential algorithm is the "Traveling Salesman Problem" (TSP). This problem is much studied in theoretical computer science. The task is simple. Given n cities, a traveling salesman must visit each exactly once before returning home. The goal is to do it the fastest possible way (or cheapest or shortest). Under the most general assumptions, the only way known to reliably solve the problem is to look at every possible route and then pick the best one. There is no known way to eliminate a meaningful proportion of the routes without checking each one.

How many routes are there? n factorial. That is, n possibilities for the first visit, and $n - 1$ for the second visit, until eventually there is just one city left for the last visit.

We like to stay away from algorithms that are exponential. Instead we invent "heuristics" which are shortcuts that tend to give good results but are not guaranteed to be the absolute best. A heuristic for TSP might be: go next to the nearest unvisited city. Or, link up the closest pair of cities. Then link the next closest pair of cities. Good heuristics can be rather tricky, but the payoff is a programming solution that you can use before the salesman dies of old age.

We say that exponential algorithms run in $\Theta(e^x)$ or exponential time. There are substantial differences between exponential algorithms, but we will leave that discussion for the CS students.

9.4 Loops

The running time of a simple loop (nothing but simple statements inside it) depends on how many times the loop will execute. We will look at several simple cases.

9.4.1 Counting Up to n

The most common case is a loop whose index starts at one (or zero) and counts by ones up to some limit n. This is a $\Theta(n)$ loop, the most common type of loop.

9.4.2 Counting Down from n

Another common case is a loop whose index starts at n and counts by ones down to some set limit, usually one or zero. This is also a $\Theta(n)$ loop, (still) the most common type of loop.

9.4.3 Add/Subtract any Constant

Whether you count up (add) or down (subtract), and whether you count by ones or fives or tens, the result is still the same. Those factors do not affect the running time of the loop. It is still a $\Theta(n)$ loop.

9.4.4 Multiplying or Dividing

If you multiply by a constant greater than one, your running time will be $\Theta(\lg n)$. That is, your index starts at one, then doubles each time until you reach or exceed n. It does not matter whether you double each time, or multiply by three each time (or four or ten or one hundred). The running time is still log n.

Similarly, if you start at n and count down by dividing by two or three or ten at each step, stopping when you reach one (or ten or one hundred), the running time is also log n.

9.4.5 Unusual Limits

Watch especially for this one variation on the limit: $i * i < n$. In this case, we are running a loop where i starts at one, for instance, and steps up by a constant while $i * i < n$. This loop will not run the full n times, but will stop when i reaches $\sqrt{n}$. Thus, it becomes a root-n loop, written $\Theta(\sqrt{n})$.

If we step up or down by multiples, then the $i * i < n$ limit has no special effect. It would theoretically be $\Theta(\lg \sqrt{n})$, but mathematically this is still the same as $\Theta(\lg n)$.

9.5 Combinations of Algorithms

When we have a $\Theta(n)$ loop (block) buried inside another $\Theta(n)$ loop (block), the effects are multiplied. The total running time becomes $\Theta(n^2)$, or "n squared." An example would be comparing two unsorted lists to see if the same item is present in each list. We might take the first item from list one and compare it to each item in list two. That would take order n time. Then we repeat for the next item in list one. As we go through all n items in list one, we have $n \times n$ or n^2 comparisons. If we double the inputs, it takes us four times as long to complete the task.

(If the lists are sorted, we can do it in $\Theta(n)$ time.)

9.5.1 Sequences of Statements

For a sequence of statements (including possibly whole blocks of statements), we take the worst case running time among the statements.

For instance, a $\log n$ block followed by a linear block would have an overall running time that is linear. The effects of the $\log n$ loop just vanish. They are too small to worry about. There is an old saying in English: Take care of the dollars and the pennies will take care of themselves.

Simple statements run in $\Theta(1)$ time. That is why they are called simple. A series of however many simple statements still runs in $\Theta(1)$ time.

9.5.2 If-Else Constructs

For if-else constructs, we always assume the worst case when we are not sure what will happen. The worst case for an if-else construct is that it will do either the if side, or the else side, whichever one is worse. For practical purposes, this behaves the same as if we did both sides (see "sequences of statements" above).

9.5.3 Worst Case Running Time

In selecting the worst case running time, we can follow two simple rules.

(1) If the running time includes a power of n, like n^2 or $n^{\frac{1}{2}}$ (which equals $\sqrt{n}$), then the block with the higher power of n is worse.

(2) If the powers of n are the same (or there are no powers of n), the block with more logs is worse.

For example, in comparing $n\sqrt{n}\lg n$ to $\lg^3 n$, the first has a power of n of 1.5 (one for n, a half for $\sqrt{n}$). The second has a power of n of zero. So the first is worse.

9.5.4 Nested Blocks

When the blocks (typically loops) are nested, we multiply their running times to get the overall running time.

9.5.5 Recursive Subroutines

If you are just preparing for the quiz, you can skip this section. There are no recursive subroutines on the quiz.

There is a more elaborate analysis that goes on for recursive subroutines. These are subroutines (or functions, or procedures) that call themselves. They are typical of a divide-and-conquer programming approach, where a function foo divides its input into smaller sets and calls itself, foo, on each of those sets.

This topic is covered in Computer Science courses. You can look it up under the name of "Master Method," also called the Master Theorem, or the Master Method for Solving Recurrences. It is covered in QuizGen's q14, Big Oh: Recurrence Relations.

Master theorem (Cormen 2/e, p.73) Let $a \geq 1$ and $b > 1$ be constants, let $f(n)$ be a function, and let $T(n)$ be defined on the nonnegative integers by the recurrence $T(n) = aT(n/b) + f(n)$, where we interpret n/b to mean either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Then $T(n)$ can be bounded asymptotically as follows.

1. If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.

2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \lg n)$.

3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$.

Typical problems look like this:

Given: $T(n) = 5T(n/2) + n^2\sqrt{n}$

The solution is: $\Theta =$ ____________________

9.6 And the Winner Is . . .

In the long run, a program with better running time is a better program. Some programs are not meant to live a long life. They run once or a few times and are permanently retired. For these programs, it does not matter much which algorithm you use (except exponential, which may not even finish once in your lifetime).

For any program that will run possibly many times, maybe for years and years, it is generally worth the extra effort to use the best possible algorithm.

A simple algorithm is fast to program and takes longer to run. A more complex algorithm costs more to program, but then it runs faster forever.

It is important for every programmer to be able to do simple kinds of running-time analysis, such as those presented here. It is important to be able to identify a better algorithm by seeing that it has a faster running time.

Beyond that, for more information one should consult a basic book on Computer Science, or take a course in analysis of algorithms, where other aspects of Big Oh analysis are more fully explored.

9.7 Typical Exam

The S2 exam will probably consist of 11 questions. You are expected to get 10 correct. Six will be short, half-page programs. The other five will be full-page programs. The programs are written in C.

It commonly takes about 60 seconds per problem to complete the exam.

Chapter 10

Tutorial on Sets

Contents

10.1 Finite Sets 107
10.2 Operations on Sets 107
10.2.1 Membership 107
10.2.2 Union 108
10.2.3 Intersection 108
10.2.4 Subtraction 109
10.2.5 Universe 109
10.2.6 Complement 109
10.2.7 Specially Named Sets 109
10.2.8 Disjoint Sets 110
10.2.9 Subset 110
10.2.10 Proper Subset 110
10.2.11 Power Set 110
10.2.12 Set Generators 110
10.3 Infinite Sets 111
10.3.1 Infinite Cardinality 111
10.3.2 Infinite Set Generators 112

It is important for students of computing to know some things about sets. Particularly, it is important to know some terminology and how to perform a few operations.

A set is an unordered collection of things.

10.1 Finite Sets

The easiest thing (for me) to think about is a nice, small, finite set.

Let's take the days of the week. This is a set with seven items in it. Normally we consider it to be an ordered set, as follows:

Sun, Mon, Tue, Wed, Thu, Fri, Sat.

But we could do it in alphabetical order:

Fri, Mon, Sat, Sun, Thu, Tue, Wed.

It is still the same set. Order does not matter. (When order matters, we call it a list.)

We can define a set by listing its members inside curly braces {...}.

Days = { Sun, Mon, Tue, Wed, Thu, Fri, Sat }.

Days = { Fri, Mon, Sat, Sun, Thu, Tue, Wed }.

In both cases, the set has the same members. Order does not matter.

10.2 Operations on Sets

There are two really important operations on sets, but a few more that are also important. The really important ones are Union and Intersection. The others include Subtraction and Complement and Universe.

10.2.1 Membership

Say $X = \{ 1, 2, 3 \}$.

The elements of X are 1, 2, and 3.

We write $1 \in X$ to say that 1 is an element of X.

The symbol $\in$ is read "is an element of."

Notice that $\in$ looks like E at the front of "element."

10.2.2 Union

The union of two (or more) sets is a set that includes all the members that are in any of the original sets.

Say $X = \{ 1, 2, 3 \}$ and $Y = \{ 3, 4, 5 \}$.

The union, written $X \cup Y$, is $\{ 1, 2, 3, 4, 5 \}$.

The symbol $\cup$ is read “union.”

Notice that $\cup$ looks like U at the front of “union.”

Notice that X had three members, and Y had three members, and the union of X and Y had five members. The number 3 appears in each set. But duplicates count the same as singletons.

The number of members in a set is called its **cardinality**.

Exam Question 2 (p.155)**:**
What do we call the number of members in a set?

Acceptable Answer:
cardinality

Exam Question 3 (p.155)**:**
What does cardinality mean?

Acceptable Answer:
It is the number of members in a set.

10.2.3 Intersection

The intersection of two (or more) sets is a set that includes all the members that are in every one of the original sets.

Say $X = \{ 1, 2, 3 \}$ and $Y = \{ 3, 4, 5 \}$.

The intersection, written $X \cap Y$, is $\{ 3 \}$.

The symbol $\cap$ is read “intersection.”

Notice that $\cap$ looks like an upside down $\cup$, and that intersections are kind of the opposite of unions.

10.2.4 Subtraction

The subtraction of one set from another is a set that includes all the members of the first set, except those that are also members of the second set.

Say $X = \{ 1, 2, 3 \}$ and $Y = \{ 3, 4, 5 \}$.

The subtraction, written $X - Y$, is $\{ 1, 2 \}$.

10.2.5 Universe

The universe, written U, is the set of all members of all sets in this category.

If we are talking about dinner foods, U would be all possible dinner foods.

10.2.6 Complement

The complement of a set X, written X', is just $U - X$, everything from the universe that is not in the original set.

10.2.7 Specially Named Sets

Some sets are special enough that we give them a special name.

$\{ \}$ represents the empty set, the set with no members.

$\varnothing$ represents the empty set, the set with no members.

$\emptyset$ represents the empty set, the set with no members.

$\mathbb{N}$ represents the natural numbers starting from one.

$\mathbb{N}^0$ represents the natural numbers starting from zero.

$\mathbb{N}^1$ represents the natural numbers starting from one.

$\mathbb{N}^*$ represents the natural numbers starting from one.

$\mathbb{R}$ represents the rational numbers, each of which can be formed by a ratio of two integers.

$\mathbb{Z}$ represents the integers (positive and negative natural numbers, including zero).

10.2.8 Disjoint Sets

When the intersection of two sets, X and Y, is the empty set $\varnothing$, then they have nothing in common, and we say that X and Y are disjoint.

10.2.9 Subset

For any sets X and Y, if $X - Y = \varnothing$, it means that starting with X, and then removing everything that is also in Y, we are left with nothing. X is then recognized as a subset of Y.

The union of X and Y is Y, because X adds nothing new to Y.

The intersection of X and Y is X, because everything in X is also in Y.

$\varnothing$ is a subset of every set.

10.2.10 Proper Subset

If X is not equal to Y, then we say that X is a proper subset of Y.

$\varnothing$ is a proper subset of every set except itself.

10.2.11 Power Set

The power set of a set X is the set of all subsets of X.

For example, if $X = \{$ 1, 2, 3 $\}$, then the power set of X is { { }, { 1 }, { 2 }, { 3 }, { 1, 2 }, { 1, 3 }, { 2, 3 }, { 1, 2, 3 } },

In this case, X has a cardinality of 3. The power set of X has a cardinality of 8. $8 = 2^3$. The cardinality of the power set is always 2 raised to the power of the cardinality of the original set. That is because for each member of the original set, it can be either present or absent in a given subset. Two choices, taken across the N items in the set.

10.2.12 Set Generators

We can generate sets by describing them. We don't have to list them explicitly. For example, we could say:

$X = \{x | x \in \mathbb{N}^1, x^2 < 20\}$

In this case, X consists of all elements, we will call each of them x, such that x is a member of the set $\mathbb{N}^1$, and x^2 is less than 20.

That leaves us with $X = \{1, 2, 3, 4\}$

10.3 Infinite Sets

Most sets I talk about are finite. The members can be listed, even if there are a lot of them. But they don't go on forever.

However, there are some really important sets that do go on forever.

The first of these is the set of natural numbers: 1, 2, 3, ...

http://en.wikipedia.org/wiki/Natural_numbers tells more.

Together with zero and the negatives of the natural numbers, we have the set of integers.

http://en.wikipedia.org/wiki/Integer tells more.

$\mathbb{Z}$ is used as a symbol to represent the integers.

10.3.1 Infinite Cardinality

Two sets have the same cardinality, which means they have the same number of members, if you can construct a one-to-one correspondence between them. That's the definition of same cardinality.

We can show that $\mathbb{N}^0$ and $\mathbb{N}^1$ have the same number of members. This is strange because clearly $\mathbb{N}^0$ includes zero as well as all the members of $\mathbb{N}^1$. So how can they be the same?

We merely have to construct a one-to-one correspondence, so that each and every member of $\mathbb{N}^0$ is paired with a unique member of $\mathbb{N}^1$, and each and every member of $\mathbb{N}^1$ is paired with a unique member of $\mathbb{N}^0$,

That correspondence is "add one". We can take any member of $\mathbb{N}^0$ and add one to it, giving us a member of $\mathbb{N}^1$. The correspondence is complete. Given any member of $\mathbb{N}^0$, we can tell you which member it matches in $\mathbb{N}^1$, and vice versa. The correspondence is complete. That is our proof, a proof by construction.

In other words, infinity plus one equals infinity.

We can also show that $\mathbb{N}$ has the same cardinality as the even numbers. It has the same cardinality as $\mathbb{Z}$. It has the same cardinality as $\mathbb{R}$, the rational numbers.

The cardinality of $\mathbb{N}$ is a countable infinity.

There is a bigger infinity, an uncountable infinity. The cardinality of the real numbers is one such infinity.

10.3.2 Infinite Set Generators

We can specify infinite sets by describing them. We can never list them explicitly. For example, we could say:

$X = \{x^2 | x \in \mathbb{N}^0\}$

In this case, X consists of all elements, we will call each of them x^2, such that x is an element of the set $\mathbb{N}^0$, the natural numbers starting with zero.

That leaves us with $X = \{0, 1, 4, 9, 16, ...\}$, the set of square numbers.

Chapter 11

S3: Counting

- Points: 50
- Exam 1: Wed, Oct 08, 09:10 to 09:40 (30m)
- Exam 2: Fri, Oct 10, 09:10 to 09:40 (30m)
- Exam 3: Wed, Dec 10, 07:10 to 09:50 (160m)
- Grading Label: **S3**

Contents

11.1 What Are We Trying To Do? **114**
11.2 Distinct Assignments **114**
11.3 Permutation **114**
11.4 Ordered Selection **115**
11.5 Unordered Selection **115**
11.6 Orderings With Identical Items **116**
11.7 Distributing Objects Into Bins **117**
11.8 Typical Exam **118**

This chapter prepares you for the S3 (Skills 3) exam, Counting (Combinations and Permutations).

QuizGen (chapter 5, page 77) quiz q31 provides exam material for **S3**.

We do five questions in each of six categories. There is no extra credit available.

We include here Combinations and Permutations.

11.1 What Are We Trying To Do?

It can be important to know approximately or exactly how many of a thing can exist, or how many ways a thing can be done.

In this chapter, we briefly look at six of those ways and show how to calculate the count for each.

11.2 Distinct Assignments

Say we have N houses and K colors of paint. Each house must be painted with one of those colors. (We keep it simple; there is no mixing of paint.)

How many ways can this be done?

The answer is that each house can be painted K different ways. Those ways do not influence each other in any way.

So, we have K ways for the first house, and K ways for the second house, and K ways for the third house, and so on.

We have K times K ..., for N Ks, or K^N, K to the Nth power.

This relies on the **multiplication rule**, where the total number of possibilities is calculated by multiplying the number of possibilities of each individual piece.

11.3 Permutation

Given N distinct (individual) objects, in how many different ways can they be listed (or arranged)?

The answer is $N!$, N factorial.

There are N choices for the item to be listed first. Then there are N-1 items from which we can choose the item to be listed second. Eventually, there is only one item that can be listed last.

$N!$ means $N \times (N-1) \times (N-2) \times (N-3) \times ... \times 3 \times 2 \times 1$.

The recursive definition is $N! = N \times (N-1)!$, and the basis case is $1! = 1$. (It is also generally agreed that $0! = 1$.)

Permutation is also called Ordered Full Set Without Replacement.

11.4 Ordered Selection

Short of a full permutation, we can stop after selecting K of the items. We then have an ordered selection of K items out of a total set of N items.

Just like with permutation, we calculate as follows:

There are N choices for the item to be listed first. Then there are N-1 items from which we can choose the item to be listed second. Eventually, there are N-K choices for the item that can be listed last.

$N \times (N-1) \times (N-2) \times (N-3) \times ... \times (N-K+1)$ which equals $N!/(N-K)!$.

If we want to list three items out of a set of seven possible items, the answer would be:

7! / 4! = 7 × 6 × 5 = 210

Pro tip: You may be able to avoid doing some of the multiplications by realizing that they cancel each other out. When the numbers start to get big, too many multiplies can exceed the limit of the largest number that the computer can handle.

Specifically, notice that it is not necessary to do eleven multiplications and one division to find this answer. You can avoid doing eight of the multiplications by realizing that they cancel each other out.

11.5 Unordered Selection

With an unordered selection, we don't care the order in which the items are selected. We can simply make an ordered selection, and then cancel out the different orderings in which the same items might appear.

This is also called "choose", as in "7 choose 3".

If we want an unordered group of three items out of a set of seven possible items, the answer would be:

7! / 4! = 7 × 6 × 5 = 210 for the ordered lists,

210 / 3 / 2 / 1 = 35 after noticing that there are 3 × 2 × 1 ways in which

each of the selections can appear.

7! / 4! / 3! = $7 \times 6 \times 5$ / 3 / 2 / 1 = $7 \times 5 = 35$

The formula is $N!/(N-K)!/K!$

Pro tip: You may be able to avoid doing some of the multiplications by realizing that they cancel each other out. When the numbers start to get big, too many multiplies can exceed the limit of the largest number that the computer can handle.

11.6 Orderings With Identical Items

In the above examples, the items were always distinctive. They could always be told apart from one another.

When we have identical items, we assume the items cannot be told apart. Another way to say this is to call them "unlabeled" items.

A classic example is ordering letters to form "words."

In how many distinct ways can the letters *aaabbc* be arranged?

Well, if the letters were distinct, we would have 6!. But since they are not distinct, we have to divide by the amount of duplication that exists.

Since there are three "a"s in the string, we divide by 6. The "a"s could have been arranged as $a_1a_2a_3$ or $a_1a_3a_2$ or four other ways.

Since there are two "b"s in the string, we divide by 2. The "b"s could have been arranged as b_1b_2 or b_2b_1.

Since there is only one "c" we divide by 1 (i.e., make no adjustment). The "c"s could have been arranged as c_1 only.

The answer will then be:

$6 \times 5 \times 4 \times 3 \times 2 \times 1$ / 3 / 2 / 1 / 2 / 1 / 1 = 60

That is, 6!/3!/2!/1! = 60

Pro tip: You may be able to avoid doing some of the multiplications by realizing that they cancel each other out. When the numbers start to get big, too many multiplies can exceed the limit of the largest number that the computer can handle.

11.7 Distributing Objects Into Bins

In this case, we have N bins. Each bin is distinct and identifiable. Each could represent a child that is receiving gifts.

The objects may include duplicates. For example, we may want to distribute 2 baseballs among 3 children. How many ways can this be done?

There are six ways, listed as follows, with each digit telling how many baseballs a certain child got. (ABC means A for the first child, B for the second child, and C for the third child.)

200, 110, 101, 020, 011, 002

Nobody said the dividing had to be fair.

The calculation for this is just about like Orderings With Identical Items. But not quite.

We will do something clever. We introduce a fake item, the boundary line between bins (children). We will use "`|`" (or "X" which is easier to type) to represent the boundary, and "b" to represent the baseball.

200 can be written as `bb||`.

110 can be written as `b|b|`.

002 can be written as `||bb`.

In short, we can restate the problem as how many different words can be constructed from the letters "`||bb`".

Notice that the number of pipes is one less than the number of kids. (The number of dividers is one less than the number of labeled bins.)

That would be $4!/2!/2! = 4 \times 3 \times 2 \times 1 / 2 / 1 / 2 / 1 = 4 \times 3 / 2 = 6$.

What if there are more items than just baseballs?

We use the **multiplication rule**, handling each of the items separately.

Say we have three children, two baseballs, and two bats.

We have 6 ways to divide the baseballs. We also have 6 ways to divide the bats. The total is 36.

Say we have 4 kids, 3 balls, 2 gloves, and 1 bat.

For the balls, we have `|||bbb` $= 6!/3!/3! = 20$.

For the gloves, we have `|||gg` = 5!/3!/2! = 10.

For the bat, we have `|||b` = 4!/3!/1! = 4. Well, that's kind of obvious when you think about it. One bat, four kids? Four ways.

Now we multiply those together, since the assignments are independent:

$20 \times 10 \times 4 = 800$

There are 800 ways to distribute 3 balls, 2 gloves, and 1 bat among 4 kids.

Pro tip: You may be able to avoid doing some of the multiplications by realizing that they cancel each other out. When the numbers start to get big, too many multiplies can exceed the limit of the largest number that the computer can handle.

Incorrect Shortcut: It may be tempting to cut out a step and try to calculate 3 balls, 2 gloves, and 1 bat among 4 kids as follows:

For everything we have `aaabbc|||` = 9!/3!/2!/1!/3! = 5040

The problem with this approach is that it counts as distinct quite a few options that actually result in each child getting the same things. A child that got, say, a ball and a bat could also have gotten a bat and a ball with the same end effect. It is hard to cancel that out.

11.8 Typical Exam

The S3 exam will probably consist of 30 questions. You are expected to answer each of them correctly.

- Five will count distinct assignments.
- Five will count distinct full orderings.
- Five will count distinct non-full orderings.
- Five will count distinct non-full subsets.
- Five will count orderings with identical items.
- Five will distribute unlabeled objects to labeled bins.

ezCalc: You will be given access to a simple calculator that you can use during the exam. You can read about ezCalc, in section 2.3.3 (page 40). ezCalc will allow you to type in something like this:

```
9 * 8 * 7 =
```

When you press the = sign, or press Enter, it will attempt to evaluate your expression. If it is successful, it will replace your typing with the answer. It allows you to use plus, minus, times, divide, and parentheses, but not things like factorial. You have to spell those things out in terms of the simple operations that it allows.

It commonly takes about 30 seconds per problem to complete the exam, so you should be done in about 15 minutes. We typically allow 30 minutes for the exam.

Chapter 12

Tutorial on Discrete Probability

Discrete probability is based on a set of cases, each of which is equally likely to occur.

An example would be the rolling of a six-sided die. There is one chance in six that the die will end with a 4 showing on top. We say the probability is 1/6 or one in six.

The probability of an even number (2, 4, or 6) showing after the roll is 1/6 + 1/6 + 1/6 = 1/2.

To calculate the probability of a specific outcome, we need to count the number of equally-likely ways that outcome could happen, and the number of equally-likely ways **any** outcome could happen. Then we divide.

When rolling two six-sided **dice**, the probability of the total being seven is calculated as follows.

It could happen as (1 6) or (2 5) or (3 4) or (4 3) or (5 2) or (6 1). That makes six equally-likely ways it could happen.

There are 36 total equally-likely outcomes: (1 1), (1 2), (1 3), ..., (6 6).

It may seem that (1 6) and (6 1) are really the same, and in one sense that is true. But consider the **dice** to be of different colors, say yellow and red. There is only one way to roll (6 6). The yellow die must be 6 and the red die must be 6. But there are two ways to roll a one and a six. Yellow could be 1 and red 6. Or yellow could be 6 and red 1.

We must be careful to measure equally-likely outcomes.

Example: What is the probability that a family with two children has one child of each gender (male / female)?

Wrong analysis: There are three possibilities: boy-boy, girl-girl, and one of each. Therefore the probability is 1/3.

Correct analysis: There are four equally-likely possibilities: boy-boy, boy-girl, girl-boy, and girl-girl. Therefore the probability is 2/4 or 1/2.

Independence

When two outcomes do not depend on each other, we call them independent.

Specifically, if we know one of the outcomes, and they are independent, then we do not know anything about the other outcome.

Consider yellow/red, as used in the example above. After rolling the **dice**, if we know that yellow came up 4, we know nothing about red.

Yellow/red is independent.

On the other hand, knowing one outcome can give us information about the other outcome sometimes. If so, they are not independent.

Let's say the outcomes are (smallest number) and (biggest number) from the roll of a pair of **dice**.

If we know the smallest number is 1, then we have no information about the other die.

But if we know the smallest number is 4, then we can conclude that the biggest number is either 4, 5, or 6.

And if we know the smallest number is 6, then we can conclude that the biggest number is also 6.

Smallest/biggest is not independent.

Chapter 13

S4: Conditional Probability

- Points: 60
- Exam 1: Fri, Oct 17, 09:10 to 09:40 (30m)
- Exam 2: Fri, Oct 24, 09:10 to 09:40 (30m)
- Exam 3: Wed, Dec 10, 07:10 to 09:50 (160m)
- Grading Label: **S4**

Contents

13.1 Strategy: Grid **124**
13.2 Strategy: Venn **128**
13.3 Notation **130**
13.4 Sample Problems **131**
13.5 Bayesian Probability **134**
13.5.1 The Product Rule 134
13.5.2 Detailed Example 135
13.5.3 Bayes' Rule and Bayesian Updating 136
13.6 Typical Exam **137**

This chapter prepares you for the S4 (Skills 4) exam, Conditional Probability.

QuizGen (chapter 5, page 77) quiz q45 provides exam material for **S4**.

Easy: We do 40 questions that are fairly easy. Hard: We do 15 questions that are a bit harder. There is 10% extra credit available.

Independent: Some events are independent. If we toss a fair coin, it comes up heads half the time. If we toss another fair coin, it also comes up heads half the time. But the results of the first coin toss has no influence on the second coin toss. We say the coin tosses are independent. They are not correlated.

Dependent: Some events are not independent. For example, there may be a 1/100 chance that any random person has the flu. But if we only look at people that have a headache and a runny nose, there may be a 1/10 chance that they have the flu. In this case, headache, runny nose, and flu are not independent. They are correlated.

With conditional probability, we assume the outcomes or events may not be independent. We assume that knowing something about one event may give us information about another event.

Bayes' Rule, discussed later in this chapter, is an important way of dealing with conditional probability.

For this test, you need to know about three kinds of probabilities: prior, joint, and conditional.

Prior: prior probabilities simply refer to a single attribute, A or Not A, for example. There is no sense of whether there are any other attributes that are interesting or influential.

Joint: joint probabilities refer to two (or more) attributes, A and B, for example. With two attributes, we have four joint probabilities: A and B, A and not B, not A and B, not A and not B.

Conditional: conditional probabilities refer to two (or more) attributes, A and B, for example. In this case, we are also given the value of one of the attributes and we have to determine the probability of the other attributes. For example: A given B, not A given B, A given not B, not A given not B.

If we know all the prior and joint probabilities, we can calculate the conditional probabilities. We will show you how to do this.

First we will look at probabilities using a probability grid.

Then we will look at probabilities using Venn diagrams.

13.1 Strategy: Grid

We can consider a grid of possibilities. We can have events that are in category A and other events that are not in category A. Similarly we can have events in B and events not in B.

Overall, each event must be equally likely. We must construct our sets of events so this is true.

We will use this notation in this section.

$p(a)$ means the probability that a randomly chosen event belongs to category A.

$p(a')$ means the probability that a randomly chosen event does not belong to category A. Instead, it belongs to "not A."

$p(a) + p(a') = 1$. Always, no matter what "a" is.

$p(ab)$ means the probability that a randomly chosen event belongs to both category A and category B. This is called the **joint probability** of A and B.

We can build a table to hold all the probabilities. Across the top, we will have two columns: a and a'. Down the side we will have two rows: b and b'. Beyond those we will have a total.

	a	a'	total
b	$p(ab)$	$p(a'b)$	$p(b)$
b'	$p(ab')$	$p(a'b')$	$p(b')$
total	$p(a)$	$p(a')$	1.0

Some of these probabilities are prior probabilities. Some of them are joint probabilities.

	a	a'	total
b	joint	joint	prior
b'	joint	joint	prior
total	prior	prior	1.0

Confusing? Okay. Let's start with something a little more concrete and specific. Imagine that I have a bag filled with objects. Each one is either a die (singular of dice) or a coin. And each one is either red or blue.

	die	coin	total
red	$p(rd)$	$p(rc)$	$p(r)$
blue	$p(bd)$	$p(bc)$	$p(b)$
total	$p(d)$	$p(c)$	1.0

Say we have 10 red dice, 7 red coins, 4 blue dice, and 6 blue coins. We have a total of 27 items in our bag. If we grab one at random, there are 10 chances it will be a red die, for a probability of 10/27. Similarly we can fill out the whole table.

	die	coin	total
red	10/27	7/27	17/27
blue	4/27	6/27	10/27
total	14/27	13/27	27/27

Now it turns out that for our purposes, we can multiply all the numbers by 27 and simplify the table.

	die	coin	total
red	10	7	17
blue	4	6	10
total	14	13	27

Red Coin: What is the probability that my randomly picked item is a red coin? There are 7 red coins. There are 27 total items. The probability of drawing a red coin is 7 out of 27: $p(rc) = 7/27$.

	die	coin	total
red	10	*7*	17
blue	4	6	10
total	14	13	*27*

Red Anything: What is the probability that my randomly picked item is a red object, either die or coin? There are 17 red objects. There are 27 total items. The probability of drawing a red object is 17 out of 27: $p(r) = 17/27$.

	die	coin	total
red	10	7	*17*
blue	4	6	10
total	14	13	*27*

Blue Die: What is the probability that my randomly picked item is a blue die? There are 4 blue dice. There are 27 total items. The probability of drawing a blue die is 4 out of 27: $p(bd) = 4/27$.

	die	coin	total
red	10	7	17
blue	*4*	6	10
total	14	13	*27*

We know the probability of randomly drawing a blue die is $p(bd)$=4/27, but what if we have additional information? What if the object has been drawn, and we are told it is blue. What is $p(bd)$ now?

Well, we don't actually call it $p(bd)$ now. Instead, we call it $p(d|b)$, the probability of it being a die given that we know it is blue.

Die, Given Blue: When I tell you that the object is blue, we say that you are "given" that the object is blue. What is the probability it is a die? Well, there are 10 blue objects. We know the object is blue so we can totally ignore all the red objects. Our total is 10, not 27 like it was before.

Among those 10 objects, 4 are dice and 6 are coins. Since we already know it is blue, and we only need to guess whether it is a die, the **conditional probability** $p(d|b)$=4/10=2/5.

	die	coin	total
red	10	7	17
blue	*4*	6	*10*
total	14	13	27

Solving a Problem: We may be asked to figure out the complete probability table from a limited amount of information. Say we are given this: $p(a)$=5/8, $p(b)$=5/16, and $p(ab)$=1/4. We know the total probability is always 1. We can fill in our table as follows.

	a	a'	total
b	1/4		5/16
b'			
total	5/8		1.0

Now, working with fractions is fun, maybe, but to keep things simple here let's convert them to whole numbers. We have fourths, eighths, and sixteenths. Lets convert everything to sixteenths. (It is called the "least common multiple." In this case, everything is a factor of 16.)

	a	a'	total
b	4/16		5/16
b'			
total	10/16		16/16

And let's multiply through by 16 to make these all into natural numbers.

	a	a'	total
b	4		5
b'			
total	10		16

Next, we just do some su-do-ku kind of thing, and fill in the blanks.

In the "a" column, we have 4, blank, 10. 4 plus 6 is 10.

In the "total" column, we have 5, blank, 16. 5 plus 11 is 16.

In the "b" row, we have 4, blank, 5. 4 plus 1 is 5.

In the "total" row, we have 10, blank, 16. 10 plus 6 is 16.

	a	a'	total
b	4	1	5
b'	6		11
total	10	6	16

Now we can fill in the middle. We can use 1, blank, 6, giving us 5. Or we can use 6, blank, 11, giving us 5. Either way, the middle cell is 5.

We could have done this all using fractions. We would have gotten this table.

	a	a'	total
b	4/16	1/16	5/16
b'	6/16	5/16	11/16
total	10/16	6/16	16/16

But natural numbers just seem easier to me.

	a	a'	total
b	4	1	5
b'	6	5	11
total	10	6	16

Armed with this table, we can answer all kinds of questions.

What is $p(ab)$? That is, what is the probability that an item belongs to category A and category B?

	a	a'	total
b	*4*	1	5
b'	6	5	11
total	10	6	*16*

There are 4 objects that belong to category A and also B. There are 16 total objects. So, $p(ab)$=4/16=1/4. (But we already knew that.)

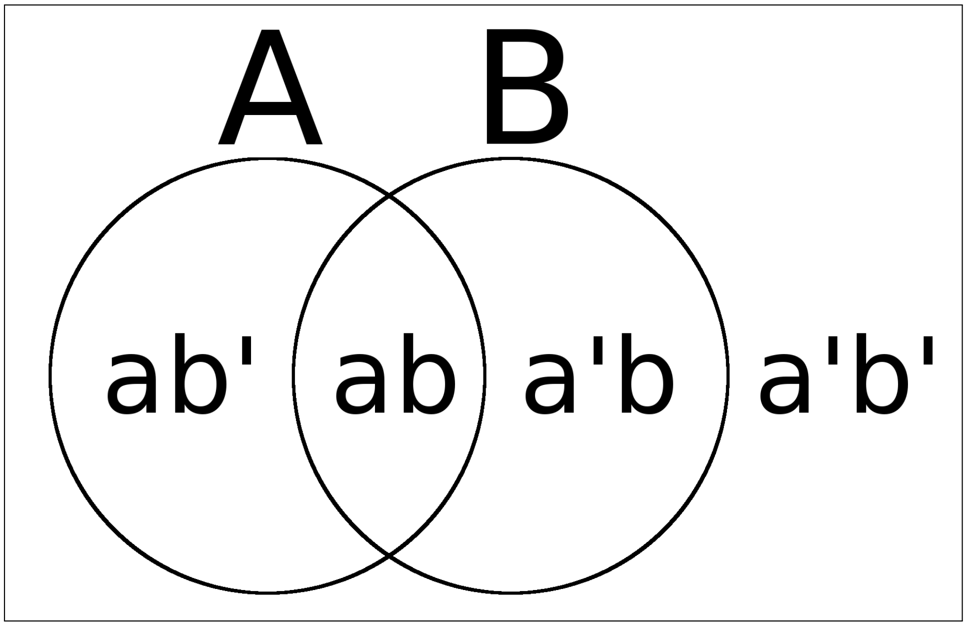
What is $p(a|b)$? That is, what is the probability that an item belongs to category A if we are given that it belongs to category B?

	a	a'	total
b	*4*	1	*5*
b'	6	5	11
total	10	6	16

There are 4 objects that belong to category A and also B. There are 5 objects that belong to category B. So, $p(a|b)$=4/5.

13.2 Strategy: Venn

Many students are familiar with the **Venn diagram**.



In this diagram, the circle on the left represents events in category A. The circle on the right represents events in category B.

The portion of circle A that is labeled ab' represents events that are in category A but not in category B.

The portion of circle B that is labeled a'b represents events that are in category B but not in category A.

The central area that is labeled ab represents events that are in both category A and category B. This is called the intersection of A and B.

The outside area that is labeled a'b' represents events that are neither in category A nor in category B.

http://en.wikipedia.org/wiki/Venn_diagram has a wonderful article that gives useful information. Even students familiar with Venn diagrams may learn something new.

The recommended approach to solving problems in conditional probability is to construct a Venn diagram using the facts at hand. Then, using the Venn diagram, determine the answer.

13.3 Notation

You should be familiar with the commonly used notations relating to probability. These are often written with mathematical symbols.

`http://en.wikipedia.org/wiki/Truth_table` has much more.

Primitives: These are some of the primitive wordings and operations used with probability.

$p(x)$: When you see p() read it as "the probability that ... is true".

$\cap$: When you see $\cap$ read it as the word "and". It can also be read as the word "intersection". It can be pronounced "cap". It is also called **conjunction**.

Notice that $\cap$ and $\wedge$ each represent the word "and". In the case of $\cap$ we are anding sets. In the case of $\wedge$ we are anding truth values. But they each mean "and."

$\cup$: When you see $\cup$ read it as the word "or". It can also be read as the word "union". It can be pronounced "cup". It is also called **disjunction**.

Notice that $\cup$ and $\vee$ each represent the word "or". In the case of $\cup$ we are oring sets. In the case of $\vee$ we are oring truth values. But they each mean "or."

$\overline{x}$: When you see a bar over something, read it as the word "not".

$|$: When you see $|$ read it as the word "given".

$\rightarrow$: When you see $\rightarrow$ read it as the word "implies". Note that "implies" is not the same as "causes".

Expressions: These are typical combinations of the five primitives into longer expressions.

$p(A)$ means "the probability that A is true".

$p(A)$=5/7 means that in the universe of possibilities, there are basically seven equally-likely groupings of things, and in five of them A is true.

$p(A \cap B)$ means the (joint) probability that both A and B are true. We may also write this as **p(A and B)**.

$p(\overline{B})$ means "the probability that not B is true", or in other words, "the probability that B is false". We may also write this as **p(not B)**.

$p(A \cap \overline{B})$ means the probability that A is true and B is false. We may also write this as **p(A and not B)**.

$p(A|B)$ means the probability that A is true if we already know that B is true. We may also write this as **p(A given B)**.

$p(A{\to}B)$ means the probability that if A is true, then B is also true. We may also write this as **p(A implies B)**.

13.4 Sample Problems

QuizGen (chapter 5, page 77) quiz q45 provides additional opportunities for you to learn and practice these skills.

Given $p(A)$=1/3, $p(B)$=7/18, $p(A{\cap}B)$=1/9.

You should first derive the following **Venn diagram**: (4(2)5)7.

The "(4(2)" part represents the A circle in the Venn diagram. It means that in four cases, A is true but B is not true. In two cases A is true and B is true. It does not say anything about when A is false.

The "(2)5)" part represents the B circle in the Venn diagram. It means that in two cases, B is true and A is also true. In five cases B is true but A is not true. It does not say anything about when B is false.

The "7" part represents the space outside the A and B circles. It means that in seven cases both A and B are false.

Each of these 4, 2, 5, and 7 cases are independent and equally likely, for a total of 18 possible cases.

Exam Question 4 (p.155)**:**
Find $p(A \cap \overline{B})$ given $p(A)$=1/3, $p(B)$=7/18, $p(A{\cap}B)$=1/9.

Acceptable Answer:
2/9

Exam Question 5 (p.155)**:**
Find $p(\overline{A}{\cap}B)$ given $p(A)$=1/3, $p(B)$=7/18, $p(A{\cap}B)$=1/9.

Acceptable Answer:
5/18

Exam Question 6 (p.155)**:**
Find $p(\overline{A} \cap \overline{B})$ given $p(A)$=1/3, $p(B)$=7/18, $p(A{\cap}B)$=1/9.

Acceptable Answer:

7/18

Exam Question 7 (p.155)**:**
Find $p(A|B)$ given $p(A)$=1/3, $p(B)$=7/18, $p(A∩B)$=1/9.

Acceptable Answer:
2/7

Exam Question 8 (p.155)**:**
Find $p(A|\overline{B})$ given $p(A)$=1/3, $p(B)$=7/18, $p(A∩B)$=1/9.

Acceptable Answer:
4/11

Exam Question 9 (p.155)**:**
Find $p(B|A)$ given $p(A)$=1/3, $p(B)$=7/18, $p(A∩B)$=1/9.

Acceptable Answer:
1/3

Exam Question 10 (p.155)**:**
Find $p(B|\overline{A})$ given $p(A)$=1/3, $p(B)$=7/18, $p(A∩B)$=1/9.

Acceptable Answer:
5/12

Given $p(A)$=5/6, $p(B)$=2/3, $p(A∩B)$=7/12, you should first derive the following Venn diagram: (3(7)1)1.

Exam Question 11 (p.155)**:**
Find $p(\overline{A}∩B)$ given $p(A)$=5/6, $p(B)$=2/3, $p(A∩B)$=7/12.

Acceptable Answer:
1/12

Exam Question 12 (p.155)**:**
Find $p(\overline{A} \cap \overline{B})$ given $p(A)$=5/6, $p(B)$=2/3, $p(A∩B)$=7/12.

Acceptable Answer:
1/12

Exam Question 13 (p.155)**:**
Find $p(A|B)$ given $p(A)$=5/6, $p(B)$=2/3, $p(A∩B)$=7/12.

Acceptable Answer:
7/8

Exam Question 14 (p.156)**:**
Find $p(A|\overline{B})$ given $p(A)$=5/6, $p(B)$=2/3, $p(A∩B)$=7/12.

Acceptable Answer:
3/4

Exam Question 15 (p.156)**:**
Find $p(B|A)$ given $p(A)$=5/6, $p(B)$=2/3, $p(A{\cap}B)$=7/12.

Acceptable Answer:
7/10

Exam Question 16 (p.156)**:**
Find $p(B|\overline{A})$ given $p(A)$=5/6, $p(B)$=2/3, $p(A{\cap}B)$=7/12.

Acceptable Answer:
1/2

Given $p(A)$=3/7, $p(B)$=3/7, $p(A{\cap}B)$=2/21, you should first derive the following Venn diagram: (7(2)7)5.

Exam Question 17 (p.156)**:**
Find $p(A \cap \overline{B})$ given $p(A)$=3/7, $p(B)$=3/7, $p(A{\cap}B)$=2/21.

Acceptable Answer:
1/3

Exam Question 18 (p.156)**:**
Find $p(\overline{A}{\cap}B)$ given $p(A)$=3/7, $p(B)$=3/7, $p(A{\cap}B)$=2/21.

Acceptable Answer:
1/3

Exam Question 19 (p.156)**:**
Find $p(\overline{A} \cap \overline{B})$ given $p(A)$=3/7, $p(B)$=3/7, $p(A{\cap}B)$=2/21.

Acceptable Answer:
5/21

Exam Question 20 (p.156)**:**
Find $p(A|B)$ given $p(A)$=3/7, $p(B)$=3/7, $p(A{\cap}B)$=2/21.

Acceptable Answer:
2/9

Exam Question 21 (p.156)**:**
Find $p(A|\overline{B})$ given $p(A)$=3/7, $p(B)$=3/7, $p(A{\cap}B)$=2/21.

Acceptable Answer:
7/12

Exam Question 22 (p.156)**:**

Find $p(B|A)$ given $p(A)$=3/7, $p(B)$=3/7, $p(A \cap B)$=2/21.

Acceptable Answer:
2/9

Exam Question 23 (p.156)**:**
Find $p(B|\overline{A})$ given $p(A)$=3/7, $p(B)$=3/7, $p(A \cap B)$=2/21.

Acceptable Answer:
7/12

13.5 Bayesian Probability

http://en.wikipedia.org/wiki/Thomas_Bayes has information about Thomas Bayes (1702-1761).

Bayes comes up a lot in artificial intelligence. It is surprisingly simple.

13.5.1 The Product Rule

The **product rule** is this:

$p(a \wedge b) = p(a|b)p(b)$ or $p(a \wedge b) = p(b|a)p(a)$

This can be easily verified by looking at a Venn diagram.

We will show an example of this shortly.

Combining these two forms, we get **Bayes' rule**, aka Bayes' law, aka Bayes' theorem, which is this:

$p(b|a) = p(a|b)p(b)/p(a)$

(For some reason, I have a hard time remembering the Bayes' rule, but a somewhat easier time remembering the product rule. Fortunately for me, it is easy to derive Bayes from product.)

Each of $p(a)$ and $p(b)$ are called **prior probabilities**, or **a priori probabilities**. They are the probabilities that something is true when you don't know anything else about anything.

$p(a|b)$ is the conditional probability that a is true given that you already know b is true.

Exam Question 24 (p.156)**:**
What is the product rule?

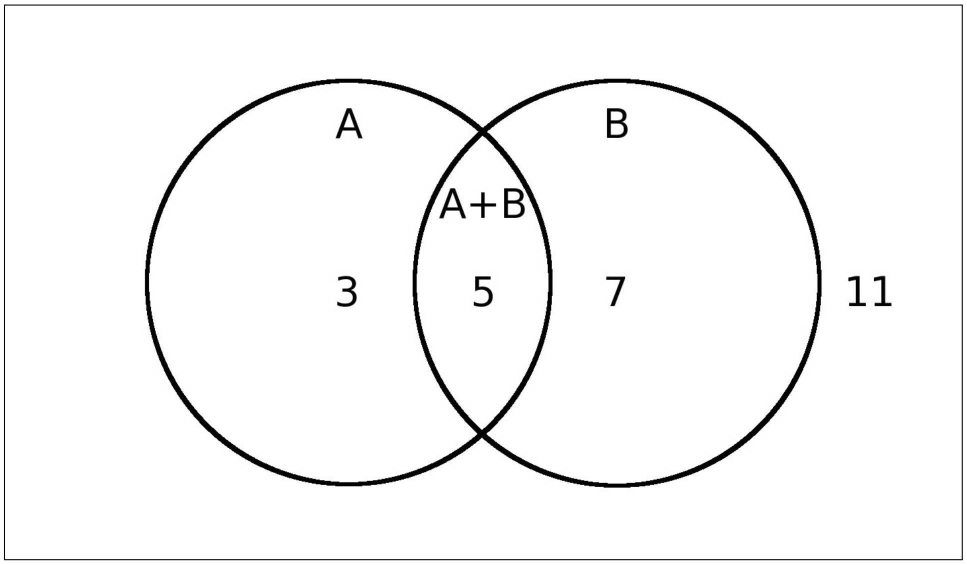
Acceptable Answer:
p(a and b) = p(a given b)p(b)

Exam Question 25 (p.156)**:**
What is Bayes' rule?

Acceptable Answer:
p(a given b) = p(b given a)p(a)/p(b)

13.5.2 Detailed Example

Let's look at what this means by way of a Venn diagram.



In this example, $p(A) = 8/26$, $p(A{\wedge}B) = 5/26$, and $p(B) = 12/26$.

Bayes' rule is used to calculate $p(b|a)$ when our basic facts include $p(a|b)$, $p(a)$, and $p(b)$. We do this as follows.

We know that $p(A{\wedge}B) = p(A|B)p(B)$. What does that mean? $p(A|B)$ is the probability of A given that B is known to be true.

In this case, we have 5+7=12 cases where B is true, and in 5 of them, A is also true. Hence, $p(A|B)$ is 5/12.

We now have our three facts: $p(A|B) = 5/12$, $p(A) = 8/26$, and $p(B) = 12/26$.

The product rule tells us that multiplying $p(A|B)$ by $p(B)$ we get $p(A{\wedge}B)$.

Specifically, 5/12 times 12/26 equals 5/26. Notice that the 12s cancel each other out.

The product rule tells us that dividing $p(A \wedge B)$ by $p(A)$ we get $p(B|A)$ which is our goal. 5/26 divided by 8/26 equals 5/8. Notice that the 26s cancel each other out.

We can check our work by counting up the chances directly. We see there are 8 chances for a to be true, and in 5 of them, b is also true. Therefore $p(b|a)$ is 5/8.

Bayes' rule just combines the two applications of the product rule.

$p(B|A) = p(A|B)p(B)/p(A)$

$p(B|A) = (5/12)(12/26)/(8/26)$

13.5.3 Bayes' Rule and Bayesian Updating

Let's look at two propositions.

A: The sound ah was uttered.

B: The computer thinks it heard the sound ah.

We want to know the probability that ah was uttered when the computer recognizes an ah: $p(u|r)$.

We can find the prior probability that ah was **uttered** by looking at a corpus of speech labeled by trained human transcribers. Let's make up a number and say that out of some corpus of speech, ah is uttered in one percent of the frames.

$p(u) = 0.01$

We can find the prior probability that ah was **recognized** by looking at a similar corpus of speech labeled by computer. Let's make up a number and say that in that corpus of speech, ah is recognized in two percent of the frames.

$p(r) = 0.02$

Due to confusion between similar phonemes, let's say that out of the times ah was actually uttered, it is recognized as the most likely phoneme 3/4 of the time.

$p(r|u) = 0.75$

From this we can calculate $p(u|r)$, the probability that ah was uttered given the computer recognized it.

$p(u|r) = p(r|u)p(u)/p(r)$ = 0.75 * 0.01 / 0.02 = 0.375.

I guess in this case our computer is not very accurate yet.

13.6 Typical Exam

The S4 exam will probably consist of 55 questions. You are expected to get 50 of them correct.

40 will be based on the given information of p(A), p(B), and p(A and B). You will be asked to calculate the conditional probabilities.

15 will be based on the given information of p(A), p(B), and one of the conditional probabilities. You will be asked to calculate the other conditional probabilities.

It commonly takes about 30 seconds per problem to complete the exam.

Chapter 14

S5: BST: Binary Search Trees

- Points: 60
- Exam 1: Fri, Oct 31, 08:50 to 09:40 (50m)
- Exam 2: Fri, Nov 07, 08:50 to 09:40 (50m)
- Exam 3: Wed, Dec 10, 07:10 to 09:50 (160m)
- Grading Label: **S5**

Contents

14.1 Tree Creation **139**
14.2 Tree Search (Look-up) **139**
14.3 Tree Traversal **140**
14.3.1 Pre-Order Traversal 141
14.3.2 In-Order Traversal 141
14.3.3 Post-Order Traversal 141
14.3.4 Breadth-First Traversal 141
14.4 The Exam **142**
14.5 Typical Exam **143**

This chapter prepares you for the S5 (Skills 5) exam, Binary Search Trees.

QuizGen (chapter 5, page 77) quiz q36 provides exam material for **S5**.

Easy: We do 6 sets of 4 questions with 7 numbers. Hard: We do 3 sets of 3 questions with 15 numbers. There is 10% extra credit available.

Binary trees are important because they can organize data for quick look-up of any item. They are easy to build and easy to use.

14.1 Tree Creation

Binary search trees are constructed by repeatedly inserting new numbers (or other sortable objects) into the tree, one by one. The objects in the tree are called nodes.

The first object becomes the root of the tree.

Each subsequent object is inserted into the tree by starting at the root. If it is an exact match, we stop. If it is less than the root, we follow the left branch of the tree. If it is greater than the root, we follow the right branch of the tree.

We repeat this process until we get an exact match or we run out of nodes.

If we get an exact match, the object is found and we are done. We do not modify the tree.

If we run out of nodes, the object is not found, and we create a new node at that location.

"Repeat this process" is called **recursion** when it is called from the midst of already running the process.

With random inputs, the tree is typically pretty well balanced, by which we mean the number of steps needed to reach any node is on the order of the log (base 2) of the number of nodes in the whole tree.

Thus, with a million nodes in a well-balanced tree, we will only wander through 20 nodes before finding a match or knowing that there is no match.

14.2 Tree Search (Look-up)

The main purpose of a binary search tree is search. By this we mean that an element is sought within the tree. It is either found or not found.

The procedure for search is the same as the procedure for insertion, up until the point that the item is either found or not found.

During tree search, the tree does not change in any way.

If the tree is well-balanced, the search runs in $O(\lg n)$ time.

14.3 Tree Traversal

Starting at the root of the tree, we can visit every node. There are two main ways to do this. **Depth-First Search** (**DFS**) uses a **stack** and **recursion** and requires very little extra space to keep track of our progress. Breadth-First Search (BFS) uses a **queue** and requires much more space but can provide answers faster in some cases.

As we "visit" each node, we do whatever processing we had in mind. This could simply mean comparing the node to some standard, or printing some attribute of the node.

Here is a sample recursive subroutine for visiting nodes using a depth-first search strategy. It is written in pseudo-code.

```
subroutine visit(node) {
  do pre-order processing, if any.
  visit(leftof(node)) // this is a recursive call
  do in-order processing, if any.
  visit(rightof(node)) // this is a recursive call
  do post-order processing, if any.
  return to the caller }
```

As we process each node, starting at the root, we go through the following steps.

Enter the node for the first time. At this time, we could do a pre-order visit if requested.

Visit the left sub-tree of that node by visiting the left child of this node. This is a recursive activity.

Return to the node for the second time. At this time, we could do an in-order visit if requested.

Visit the right sub-tree of that node by visiting the right child of this node. This is a recursive activity.

Return to the node for the third time. At this time, we could do a post-order visit if requested.

When is DFS good? It will normally use a much smaller amount of memory than BFS.

When is DFS bad? For many trees, the DFS method will "go down a rabbit hole" looking for a solution and will miss a solution that was on a nearby branch. It does not find the closest solution. Some trees are infinitely deep and branches, once bypassed, will never be visited.

14.3.1 Pre-Order Traversal

With pre-order traversal, when we enter a node, we visit it immediately. Then we recursively go down the left-hand side of the sub tree. Then we recursively go down the right-hand side of the sub tree.

The effect is to zig-zag up and down the tree, tracing its outline, starting at the root and ending when all nodes have been visited.

14.3.2 In-Order Traversal

With in-order traversal, when we enter a node, first we recursively go down the left-hand side of the sub tree. Then we visit the node itself. Then we recursively go down the right-hand side of the sub tree.

The effect is to zig-zag up and down the tree, tracing its outline, starting at the root and ending when all nodes have been visited.

The in-order traversal will always result in a sorted list of nodes.

14.3.3 Post-Order Traversal

With post-order traversal, when we enter a node, first we recursively go down the left-hand side of the sub tree. Then we recursively go down the right-hand side of the sub tree. Lastly we visit the node itself.

The effect is to zig-zag up and down the tree, tracing its outline, starting at the root and ending when all nodes have been visited.

14.3.4 Breadth-First Traversal

With breadth-first traversal, we start with an empty **queue** which is our to-do list. Then we add the root node to that list. Then we repeat the following steps.

We remove the first node from the list and visit it. Then, if it exists, we add its left-hand node to the end of our to-do list. Then, if it exists, we add its right-hand node to the end of our to-do list.

The effect is to go from left to right across the tree, row by row, starting at the root and ending when all nodes have been visited.

Here is a sample iterative procedure for visiting nodes using a breadth-first search strategy. It is written in pseudo-code. Here, **push** means add to the back end of the **queue** and **shift** means remove from the front end of the queue.

```
queue = empty;
push root onto end of queue;
while ( queue is not empty ) {
  shift first item off of the front of queue
  visit that item if it exists
  push its left child, if any, onto the end of queue
  push its right child, if any, onto the end of queue
  continue }
```

When is BFS good? It will find the closest solution in the tree.

When is BFS bad? For many trees, the BFS method will use lots more memory than a DFS method because the queue will grow to be as long as the tree is at its widest point.

14.4 The Exam

You will be given a list of numbers to insert into a new binary search tree. It typically takes about two minutes to draw a 15-node tree on a sheet of paper.

You will then be asked to traverse the tree and report the numbers as they are visited. It typically takes 30 seconds to traverse a 15-node tree, typing the numbers into the computer as you go.

For efficiency, you may be asked to traverse the same tree several times in different ways.

14.5 Typical Exam

The S5 exam will probably consist of 33 questions. You are expected to get 30 of them correct.

24 will be based on seven-number trees. There are six trees. For each tree, you will report the pre-order, in-order, post-order, and breadth-first traversal.

9 will be based on fifteen-number trees. There are three trees. For each tree, you will report the pre-order, post-order, and breadth-first traversal.

It commonly takes about 60 seconds per problem to complete the exam.

Chapter 15

S6: Huffman Coding

- Points: 60
- Exam 1: Fri, Nov 14, 08:50 to 09:40 (50m)
- Exam 2: Fri, Nov 21, 08:50 to 09:40 (50m)
- Exam 3: Wed, Dec 10, 07:10 to 09:50 (160m)
- Grading Label: **S6**

Contents

15.1 Background 144
15.2 Building The Tree 145
15.3 Building The Code 145
15.4 The Exam 146
15.5 Only One Correct Answer? 147
15.6 Typical Exam 147

This chapter prepares you for the S6 (Skills 6) exam, Huffman Coding.

QuizGen (chapter 5, page 77) quiz q35 provides exam material for **S6**.

Easy: We do 8 questions with 8 letters. Hard: We do 3 questions with 11 letters. There is 10% extra credit available.

15.1 Background

Huffman codes are also called prefix codes. Each letter (or other thing) is coded with a unique set of bits, such that no set of bits is a prefix to any other set of bits.

The bits essentially define a path through a decision tree, where the answers are always at the end, not buried in the middle like they can be for binary search trees. When you reach the end, you copy out that letter. Then start over at the root of the tree with the next bit.

15.2 Building The Tree

When constructing a Huffman code, you must build a decision tree. The procedure for doing this is simple.

Consider each letter to be a terminal (ending) node in the decision tree. Each node has a weight that is the frequency of that letter.

Find the two nodes with the lowest weight. Create another node that joins them, and give it a weight that is the sum of the two original weights.

Example: If the two smallest weights are (12 a) and (16 b), create a new node called (28 a b). When you reach that node in the ultimate decision tree, you will know the letter will be either a or b.

If there are more than two nodes with the same lowest weight, it does not matter which two you pick to combine.

Continue this process, combining the two smallest nodes and creating a new node, until you have combined everything. Then you are done. You have your tree.

15.3 Building The Code

Throughout the tree, for each non-terminal node, assign 0 to one branch and 1 to the other branch. It does not matter which way you assign them.

Now, starting at the root, follow the path to each letter, one by one. Copy down the 1s and 0s along the path to that letter. The result is the Huffman code for that letter.

Because of the way the codes are assigned, it is impossible for any two letters to have the same code. And it is impossible for any letter's code to be a "prefix" of another letter's code.

Example: If 11100 is a code, then 1110 cannot be a code, and 111 cannot be a code, and 11 cannot be a code, and 1 cannot be a code.

The reason is simple. When you receive 11100, it can be decoded by following the decision tree, starting at the base (root), and then taking the 1 branch, and then the 1 branch, and then the 1 branch, and then the 0 branch, and then the 0 branch. That will end at the letter which was encoded.

If any of the prefixes were a letter code, it would mean the tree had a letter that was not on the outer edge (frontier) of the tree. But because of the way the tree was built, we know that could not happen.

15.4 The Exam

You will be given a list of letters and frequencies.

Your task is to develop a set of codes for the letters.

However, the exam does not (currently) ask you for those codes.

Then, you multiply each frequency by the number of bits in its code. That becomes the cost for coding that letter that many times. Add it up across all the letters and you have the total.

The exam asks you for the total.

Example:

Using the specified letter.frequency pairs, develop a correct Huffman code. Report the total bits used (sum of length times frequency for each letter). Count carefully.

a.3 b.24 c.2 d.16 e.16 f.21 g.4 h.2 j.15 k.17 m.26

This means the letter "a" appears 3 times, the letter "b" appears 24 times, and so forth.

From this you might decide on the following Huffman code:

a=111010 b=110 c=1110110 d=010 e=011 f=101
g=11100 h=1110111 j=1111 k=100 m=00

(There are lots of correct answers. This is just one of them.)

Since a happens 3 times and requires 6 bits each time, we count 3*6 bits.

Since b happens 24 times are requires 3 bits each time, we count 24*3 bits.

The calculator capability will allow you to type in a mathematical expression like this:

3*6+24*3+2*7+16*3+16*3+21*3+4*5+2*7+15*4+17*3+26*2=

When the = sign is pressed, the expression will be replaced with the answer, which is 460. There is only one correct total. All correct Huffman codes have the same total bits.

15.5 Only One Correct Answer?

As you process the letters, you combine branches in the decision tree. You must always combine to two smallest-weight branches to guarantee a correct solution. (Sometimes another combination will result in the same total, but it cannot be guaranteed.)

If there are several choices with the lowest weight (frequency), it does not matter which way you combine. The result is still provably optimal.

It is possible to end up with two letters that have the same frequency and yet end up with a different number of bits.

In one example I saw, (6 x) had five bits, and (6 y) had four bits. If you swap them around (which you might have), each x would take four bits, thus saving six bits, but each y would take five bits, thus costing six more bits. The net gain would be zero.

The order that things are combined affects the code you create, and maybe the number of bits per letter, but it does not affect the total bits needed.

15.6 Typical Exam

The S6 exam will probably consist of 11 questions. You are expected to get 10 of them correct.

8 will be based on eight-letter alphabets.

3 will be based on eleven-letter alphabets.

It commonly takes about 2.5 minutes per problem to complete the exam.

Chapter 16

S7: MST: Minimum Spanning Trees

- Points: 60
- Exam 1: Mon, Dec 01, 08:50 to 09:40 (50m)
- Exam 2: Fri, Dec 05, 08:50 to 09:40 (50m)
- Exam 3: Wed, Dec 10, 07:10 to 09:50 (160m)
- Grading Label: **S7**

Contents

16.1 Background **149**
16.2 The Exam **149**
16.3 Typical Exam **150**

This chapter prepares you for the S7 (Skills 7) exam, Minimum Spanning Tree.

QuizGen (chapter 5, page 77) quiz q18 provides exam material for **S7**.

Easy: We use 6 nodes on 6 questions. Hard: We use 7 nodes on 5 questions. There is 10% extra credit available.

16.1 Background

A **graph** is a network of nodes, typically having many connections per node, and typically having **cycles** by which you can go from one node to another, eventually returning to your starting point.

Each connection between two nodes is called an **edge**.

A fully-connected graph that involves N nodes will have $N(N-1)/2$ edges.

A tree is a graph with no cycles, but where everything is connected. A tree that involves N nodes will have $N-1$ edges.

A **forest** is a graph with no cycles, but where everything may not be connected.

Sometimes the edges have weights (or costs).

A spanning tree is simply a tree that connects all the nodes of a graph.

A minimum spanning tree is a tree that connects all the nodes and has the smallest possible cost of any spanning tree.

We do not actually care how long any path is. It could be very long. The only thing we care about is that you can get from any point to any other point, and that the whole tree has minimal weight.

16.2 The Exam

You will be given a list of nodes (also called vertices). For example:

a b c d e f

In this example, there are six nodes and the first node is called "a".

You will be given a list of edges in vertex.vertex.weight format. For example:

a.b.17 a.c.7 a.e.14 a.f.14 b.d.18 b.e.10 b.f.25 c.d.10 c.e.29 c.f.13 d.e.8 d.f.17

In this example, the first edge is between nodes a and b, and has a weight of 17.

Your task is to discover a minimum spanning tree and report its total weight.

You will be provided with a calculator capability to help you add up the weights, but you must decide which weights to add up.

The typical approach to solving this problem is to draw a diagram that shows each of the nodes, probably arranged in a circle. Then each edge is added by drawing a line between its two nodes, and its weight is written on or near the edge.

Next, you select a node to be your starting point. The choice does not matter because every node has to be part of the spanning tree eventually. The MST is grown by adding **the smallest edge that will add a new node to the tree.** Edges that would create cycles are crossed out. Eventually all edges will either be added or crossed out.

In the example above, the first node selected might be (d).

Next we might add d.e.8, thus adding (e) to the tree. We have to select an edge that involves d, and d.e.8 is the one with the lowest weight.

Next we might add b.e.10, thus adding (b) to the tree. We have to select an edge that involves d or e, and we have two choices that are minimal.

Next we might add c.d.10, thus adding (c) to the tree. We have to select an edge that involves d, e, or b, and something not already in the tree.

Next we might add a.c.7, thus adding (a) to the tree. We have to select an edge that involves d, e, b, or c, and something not already in the tree.

Next we might add c.f.13, thus adding (f) to the tree.

Finally, the weights of the included edges are added up and the answer is typed into the blank provided on the test.

The calculator capability will allow you to type in a mathematical expression like this:

8+10+10+7+13=

When the = sign is pressed, the expression will be replaced with the answer: 48

16.3 Typical Exam

The S7 exam will probably consist of 11 questions. You are expected to get 10 of them correct.

6 will be six-node graphs.

5 will be seven-node graphs.

It commonly takes about 2.5 minutes per problem to complete the exam.

Appendix

Appendix A

Spelling Errors

I offer extra credit for reports of spelling and grammar errors in my formal communications, by which I mean written materials like syllabi, study guides, and text books as well as current portions of webpages. This is very helpful to me in correcting spelling mistakes. And it sometimes gets my students to read my materials carefully.

This has gotten to be sort of a game at times, which makes it fun. We can get into Grammar Nazi mode and be picky, picky, picky. Students will cut and paste my words into a document and then run a spelling checker or grammar checker. Or they will directly open the PDF in a spelling or grammar checker.

You are welcome to do this, but you should be aware that spelling and grammar checkers work by a simplified set of rules compared to real life. If there are two spellings for a word, the spelling checker will commonly only accept one and will reject the other. This does not make the other wrong.

The truth about English, and probably all languages, is that language changes over time. New words are created. New spellings are accepted. New grammar happens. And old grammar is resurrected.

I generally follow the accepted practices as shown in style guides such as the Chicago Manual of Style. But I take exception to certain things like those that are noted below. For things that I have considered and listed below, even though they may show up with a checker, I do not consider them to be incorrect.

My rules are (a) is it commonly done? (b) is it ambiguous? (c) is it pretty?

These are the same rules used by grammarians, but our decisions in any given case may be different.

Here is my list.

themself - Modern usage has tended away from gender-specific words like himself in favor of gender-neutral words. I have migrated from him and her to “singular” **them** as my solution of choice to the gender-neutral dictates of modern political correctness. Some dictionaries do not recognize themself as a word, and instead suggest themselves. For plural them, this would be correct, but for singular them, themself is correct and is documented to have been used as far back in time as the 1400s.

vs - Should it have a dot? The usage argument is that in British writing, abbreviations are dotted when the final letters have been dropped, but not when the intermediate letters have been dropped. Versus removes intermediate letters. American usage may differ. I do not put a dot after it. I don’t like how it looks with a dot. It is a conscious decision, not an error.

zeros - versus zeroes: Both are considered correct. Google says that zeros is more commonly used.

Ambiguous Plurals - The plural of 15 is 15s, not 15’s. Using an apostrophe generally indicates possession, but people do commonly (and incorrectly) use an apostrophe for plurals when without it the meaning seems less clear. My choice when making a plural that would look ambiguous is to quote the string being pluralized. So, for me, the plural of (a) is (“a”s) rather than (a’s) or (as).

Ambiguous Quoted Punctuation - When should punctuation that is not part of a quote be moved inside the quote marks? Typesetters traditionally float a period (full stop) inside a trailing quote mark because it looks better that way. In computing, quote marks typically delimit strings that have special meaning, and putting punctuation inside the marks changes the meaning of the string. I usually float punctuation if it does not change the meaning of the thing quoted. Otherwise not.

Series Comma - Some people write a list of three things as (a, b and c), but others write it as (a, b, and c). I write it the second way. This is not an error. Both usages are correct, but I find the first usage to be ambiguous, so I almost always use the second form.

Appendix B

Test Bank

Test Bank

1: (p.87) Use Truth Tables to Prove DeMorgan's Laws.

2: (p.108) What do we call the number of members in a set?

3: (p.108) What does cardinality mean?

4: (p.131) Find $p(A \cap \overline{B})$ given $p(A)$=1/3, $p(B)$=7/18, $p(A{\cap}B)$=1/9.

5: (p.131) Find $p(\overline{A}{\cap}B)$ given $p(A)$=1/3, $p(B)$=7/18, $p(A{\cap}B)$=1/9.

6: (p.131) Find $p(\overline{A} \cap \overline{B})$ given $p(A)$=1/3, $p(B)$=7/18, $p(A{\cap}B)$=1/9.

7: (p.132) Find $p(A|B)$ given $p(A)$=1/3, $p(B)$=7/18, $p(A{\cap}B)$=1/9.

8: (p.132) Find $p(A|\overline{B})$ given $p(A)$=1/3, $p(B)$=7/18, $p(A{\cap}B)$=1/9.

9: (p.132) Find $p(B|A)$ given $p(A)$=1/3, $p(B)$=7/18, $p(A{\cap}B)$=1/9.

10: (p.132) Find $p(B|\overline{A})$ given $p(A)$=1/3, $p(B)$=7/18, $p(A{\cap}B)$=1/9.

11: (p.132) Find $p(\overline{A}{\cap}B)$ given $p(A)$=5/6, $p(B)$=2/3, $p(A{\cap}B)$=7/12.

12: (p.132) Find $p(\overline{A} \cap \overline{B})$ given $p(A)$=5/6, $p(B)$=2/3, $p(A{\cap}B)$=7/12.

13: (p.132) Find $p(A|B)$ given $p(A)$=5/6, $p(B)$=2/3, $p(A{\cap}B)$=7/12.

14: (p.132) Find $p(A|\overline{B})$ given $p(A)$=5/6, $p(B)$=2/3, $p(A \cap B)$=7/12.

15: (p.133) Find $p(B|A)$ given $p(A)$=5/6, $p(B)$=2/3, $p(A \cap B)$=7/12.

16: (p.133) Find $p(B|\overline{A})$ given $p(A)$=5/6, $p(B)$=2/3, $p(A \cap B)$=7/12.

17: (p.133) Find $p(A \cap \overline{B})$ given $p(A)$=3/7, $p(B)$=3/7, $p(A \cap B)$=2/21.

18: (p.133) Find $p(\overline{A} \cap B)$ given $p(A)$=3/7, $p(B)$=3/7, $p(A \cap B)$=2/21.

19: (p.133) Find $p(\overline{A} \cap \overline{B})$ given $p(A)$=3/7, $p(B)$=3/7, $p(A \cap B)$=2/21.

20: (p.133) Find $p(A|B)$ given $p(A)$=3/7, $p(B)$=3/7, $p(A \cap B)$=2/21.

21: (p.133) Find $p(A|\overline{B})$ given $p(A)$=3/7, $p(B)$=3/7, $p(A \cap B)$=2/21.

22: (p.133) Find $p(B|A)$ given $p(A)$=3/7, $p(B)$=3/7, $p(A \cap B)$=2/21.

23: (p.134) Find $p(B|\overline{A})$ given $p(A)$=3/7, $p(B)$=3/7, $p(A \cap B)$=2/21.

24: (p.134) What is the product rule?

25: (p.135) What is Bayes' rule?

Index

2145, 7

a priori probabilities, 134
academic integrity, 28
Analysis of Algorithms, 96
and, **83**
associative, 83
atomic statement, **82**

balanced tree, 139
basis case, 58
Bayes' rule, **134**, 135, 136
Bayesian probability, **134**
Bayesian updating, **136**
Big Oh, 96
binary search tree, **138**
bins, 117
BST, **138**

calendar, 8
cardinality, **108**
causality, 85
CLO, 24
CNF, **88**
CNF notation, **91**
combinations, 113
commutative, 83
complement (set), 109
conditional probabilities, 123
conditional probability, 122, 126
conjunction, 130
conjunctive normal form, **88**
correlation, 123
counting, 113
cycles, 149

daily quiz, 11
daily update, 10, 37
DeMorgan's Laws, **87**
Depth-First Search, **140**
DFS, **140**
dice, 120, 121
discrete probability, 120
disjoint sets, 110
disjunction, 130
disjunctive normal form, 88
distinct, 63
distinct assignments, 114
distributing objects into bins, 117
DNF, 88
dress and grooming, 31
dynamic variables, 59

edge, **149**
email, 49
empty set, 109
equivalence, **86**
exams, 15

factorial, 114

Fibonacci, 58
forest, 149
formal communication, 21
formal logic, **81**
formal logic operators, 83, 85

global variables, 59
grading, 9
graph, **149**

houses, 114
Huffman coding, **144**

identical, 63
ILO, 23
implies, **85**
in-order, 69
independence, 118, **121**
informal communication, 21
integers, 109
intersection (sets), 108

joint probabilities, 123
joint probability, 124, **130**

learning framework, 17
learning outcomes, 23
local variables, 59

memoization, 59
minimum spanning tree, **148**
modus ponens, **82**
modus tollens, **82**
MST, **148**
multiplication rule, **114**, 117

natural numbers, 109
normal forms, 87
not, **85**

or, **84**
ordered, 63
ordered selection, 115
orderings with identical items, 116

p01, **56**
p02, **57**
p03, **58**
p04, **60**
p05, **62**
p11, **64**
p13, **65**
p14, **66**
p15, **67**
p21, **68**
p22, **70**
p23, **71**
p31, **72**
p32, **73**
p33, **74**
p34, **75**
p35, **76**
paint, 114
permutations, 113, **114**
plagiarism, 28
PLO, 24
post-order, 69
power set, 110
pre-order, 68
prior probabilities, 123, 134
probability, 120, 122
product rule, **134**
proper subset, 110
push, 142

queue, 140–142

rational numbers, 109
readings, 11
recursion, 58, 114, 139, 140
replacement, 63
resolution, **89**, 92

S1, **89**, 89
S2, **96**, 97
S3, **113**, 113
S4, **122**, 122
S5, **138**, 138
S6, **144**, 144
S7, **148**, 148
sample, 63
selection, 63
set intersection, 108
set membership, 107
set subtraction, 109
set union, 108
sets, 106
shift, 142
SLO, 24
sorted CNF, 93
special needs, 32
stack, 140
static variables, 59
study time, 12
style, 52
subset, 110
subtraction (sets), 109

truth table, 83–85, **86**, 86, 87
tutoring, 19

union (sets), 108
universe, 63
universe (set), 109
unordered, 63
unordered selection, 115

Venn diagram, **128**, 131
Version 1: Recursion, **58**
Version 2: Memo-ization, **59**

xor, **84**