CPSC 413 Lecture Notes — Part I

Department of Computer Science

Fall, 1998

Contents

Part I

Introduction

Chapter 1

Introduction to CPSC 413

1.1 Overview

This chapter provides an introduction to the course. First, "two motivating problems" are defined and used to raise issues that will be considered throughout CPSC 413. This is followed by a longer description of the course, including an explanation of why the course is required for a BSc in Computer Science and what it is hoped that students will learn by taking it. Later sections include descriptions of other resources that are available to CPSC 413 students and suggestions of things to do in order to increase the likelihood that you'll do well in this course.

1.2 Two Motivating Problems

1.2.1 Definition and Computability of Problems

Consider the following problems in arithmetic, and how you might solve them using a computer.

1. *Integer Squaring* takes a positive integer n as input, and returns n^2 as output.
2. *Integer Primality* takes a positive integer $n \geq 2$ as input and returns the answer "Yes" if n is prime, or "No" if n is composite.

These problems are both "computable," as this was defined in CPSC 313: You could decide on reasonable encodings for inputs and outputs and then write Turing machine programs for each. As far as CPSC 313 is concerned, the problems are therefore "easy" in this sense.

However, from the perspective of CPSC 413 (which analyses algorithms and introduces the theory of computational *complexity*), these problems are quite different.

1.2.2 Representations

Of course, in order to define these problems completely enough to solve them, it's necessary to agree on a way to represent the problem's inputs and outputs. As in CPSC 313, we'll represent both as strings over some finite alphabet.

Several representations of the input (an integer n) are possible. Here are two:

1. A "unary" representation: A positive integer n is represented by a string of n ones, terminated by a blank (or by a zero).

2. A "decimal" representation, using an alphabet that includes the digits from 0 to 9 as well as a terminator symbol, such as a blank. An integer n would be given by its decimal representation, which has length approximately $\log_{10} n$.

From the perspective of CPSC 313 these two representation schemes are pretty much the same, in the sense that it's easy to convert from either one to the other using a deterministic Turing machine (or random access machine, etc).

From the perspective of CPSC 413, these two representations are completely different: The length of the first representation of n is *exponential* in the length of the second representation of n, so that the first representation scheme is "unacceptable" (while the second is perfectly OK).

This second representation is unacceptable because it's so much longer than the first. It's entirely possible that there's an algorithm to solve each problem using the first representation, when given some value n as input, in *far less time* than you'd need just to *read* the same input value n if the second representation had been used instead!

In CPSC 413, we won't always insist that the *shortest* possible representation of an input be used — but we'll *definitely* reject representations that are *exponentially* longer than necessary.

In particular, a binary representation of n, hexadecimal representation of n, or a representation using another base (provided that the base is greater than or equal to two), will also be acceptable for the purposes of CPSC 413: You can convert from any of *these* representations of n to a decimal representation, or vice-versa, deterministically in polynomial-time, and the *length* of any one of these representations of n is linear in the length of any of the others.

Now, representing outputs will be straightforward. Since the output of *Integer Squaring* is just another integer, we'll agree to use the same representation of outputs as for inputs in this case. The output of *Integer Primality* is always either "Yes" or "No;" we'll agree that "Yes" is represented by the symbol "1" and "No" by "0"

1.2.3 Two Straightforward Solutions and Their Properties

Let's suppose from now on that we're using a decimal representation of n. You know (and you've been using) a deterministic polynomial-time algorithm for the first problem, *Integer Squaring*: You just multiply n by itself, using the method for integer multiplication that you learned sometime in public school. (I'll leave it to you to recall what that algorithm *was*; it might help to multiply 1324 by itself on paper and then consider what you've done.)

Assuming that your model of computation is sufficiently powerful (even a deterministic Turing machine with at least two tapes), this is a "quadratic" algorithm: If k is the length of the input (a representation of some integer n), then the number of steps used by this algorithm is at most linear in k^2.

You've probably also seen a simple algorithm for *Integer Primality*: Given $n \geq 2$ you first check whether n is even (and say "No" if it is). Otherwise, you check whether n is divisible by 3, 5, 7, 9, and so on, checking all the odd numbers until either a proper factor of n is found (at which point you stop and say "No,"), or you've checked all the odd numbers that are less than or equal to $\sqrt{n}$ without finding a factor (at which point you stop and say "Yes").

While this uses a number of steps that is polynomial in n, the *size of the input* is actually the length of a decimal representation of n, and this length is logarithmic in n. Thus the algorithm described above uses time that is *exponential* in this size — and this is *not* an acceptable (that is, "asymptotically efficient") algorithm, by the standards of CPSC 413.

If you aren't convinced that this algorithm really is slow, I suggest that you try to use it to

check whether the integer

$$n = 3,558,458,718,976,746,753,830,538,032,062,222,085,722,974,125,723$$

is prime — but if you write a computer program to do this, make sure you have a way to stop it before it completes!

Let's assume that you wrote such a program (and that you didn't "cheat" by changing the algorithm that's been described). No current computer is able to perform anything close to one trillion (10^{12}) integer operations per second — and it isn't likely that computers will be able to operate this quickly any time soon — so let's be generous and assume that this is the number of operations that can be performed. Since n is "probably prime" (according to the computer algebra system MAPLE), the number of these operations that will be executed by this algorithm is "probably" approximately $\sqrt{n}/2$ — and you'd need just under **thirty thousand years** in order to complete this many operations, using the above estimate of the number of operations that can be performed per second. (Since the estimate *isn't* accurate, this amount of time might not be enough!)

Note that the number of digits you need to write n (that is, the size of the input) is less than fifty.

1.2.4 A Better Solution for the Second Problem

Indeed, there *is* an efficient algorithm for this problem; it's a "randomized algorithm" (it makes calls to a random number generator), can make a mistake with controllably small probability (that is, its output should be interpreted as either "definitely composite" or "probably prime," and the time it uses depends on the probability of this kind of failure that is allowed). MAPLE uses either this algorithm, or an algorithm like it, to perform primality tests, and it takes less than a second on my workstation to apply it, to report that the above integer n is (probably) prime.

We certainly won't get to this algorithm in CPSC 413. However, it *is* sometimes presented in the successor course, CPSC 517, and you *will* find it in at least one of the reference books for the course that are available in the library (Cormen, Leiserson, and Rivest's book, *Introduction to Algorithms* [3], in Chapter 33, "Number-Theoretic Algorithms").

There's no efficient *deterministic* algorithm problem that has been proven, unconditionally, to be correct. On the other hand, if a plausible number-theoretic conjecture (the "Extended Riemann Hypothesis") is true, then an efficient deterministic algorithm exists for this problem as well.

Things are even worse for a problem that's related to primality testing, *Factorization,* in which you're asked to produce a proper factor of a given integer n if n isn't prime. The randomized algorithm for primality testing mentioned above manages to prove that n is composite *without* finding a factor, and no deterministic or probabilistic polynomial-time algorithm for factorization is known at all.

You might be wondering by now whether the "complexity" of these problems is of any real importance. It is! The security of some modern cryptographic protocols depend on the widely held assumption that integer factorization is hard (requiring more than polynomial time) *on average*, as well as in the worst case. On the other hand, in order to *use* some of these protocols to encode data, it's necessary to find extremely large prime numbers — and one way to do this (which turns out to be reasonably efficient) is just to make random choices of integers of the length that you need, and then use an efficient algorithm for *Integer Primality* to check each one, until a prime number has been found. It can be shown (but it's beyond the scope of this course) that there are

sufficiently many large primes, so that you won't have to look too long before a prime number is found — provided that your "random number generator" is reliable.

Professor Cleve is the person in the department whose research areas include cryptography, so you should ask him about this if you're interested!

1.3 About This Course

1.3.1 Why CPSC 413 Exists

CPSC 413 is an introductory course on the design and analysis of algorithms. It's mostly concerned with the "asymptotic complexity" of algorithms — which concerns the way that time and space used by the algorithms tend to grow as the size of the input increases. (And, when the "asymptotic performance" or "asymptotic efficiency" of an algorithm is mentioned later on, this will refer to the same thing.)

CPSC 413 is also a "theory course," in the sense that it stresses mathematical analysis over experimentation. There will be more about this, later.

Asymptotic Performance

General interest in asymptotic performance began in the 1960's and 1970's, when computers had been available long enough to be used for large scientific and engineering problems, but before they were quite as powerful or widely available as today.

At around that time, computers were being used to solve *larger* problems than had been attacked before, and it began to be clear that some of the methods that had been good enough to solve small problems by hand didn't scale up — they were either using *far too much time* or *far too much space* on larger inputs to be useful, so that new "asymptotically efficient" techniques would be needed to deal with the larger problems that were of interest.

Another trend was becoming evident: While computers were becoming more powerful, and more widely available, the *demand* for computing was also growing. It wasn't just true that the *number* of problems to be solved was growing; the *size* of the problems (seen to be of interest) was growing too.

Because of this, difficulties caused by the lack of asymptotically efficient solutions for some problems didn't *decrease* over time — they tended to *increase*, instead!

Comparison of Algorithms

These days it's clear that there are other things that a developer should worry about besides the efficiency of a program. Indeed, there are quite a few other things that should often to be considered to be more important — including correctness, ease of use, and maintainability.

However, it *is* still the case that asymptotic performance is important in some application areas, including scientific and engineering computations. And, more generally, when other important requirements are satisfied, it's still reasonable to prefer a program that's efficient to one that isn't.

It's also true that there are other ways, besides performing a mathematical analysis, to assess the performance of an algorithm: You can also write the source code, and compile and profile it, and you can perform other kinds of experiments or tests as well.

A mathematical analysis can't really replace this kind of experimentation, because profiling and similar testing frequently reveal more about how a program will run *on a particular platform*,

and *when the system is under a particular load*, than you can easily discover using a mathematical analysis.

On the other hand, experimentation can't replace mathematical analysis, either: There's no way to perform a finite number of experiments in order to measure an algorithm's performance on an infinite number of *arbitrarily large* inputs.

So, it's (still) true that *at a minimum*, a software developer should know enough about the asymptotic performance and analysis of algorithms to be able to **make an intelligent choice between algorithms or programs, based on their asymptotic performance.** This includes the ability to **compare two asymptotic performance bounds, in order to decide which is better,** and to **take source code and derive an accurate asymptotic performance bound for it.**

By the way, it's reasonably common to find "asymptotic performance bounds" associated with algorithms or programs when these are published in conference papers or journal articles. It isn't quite as common to find this kind of performance guarantee associated with software that's part of a component library for a programming language or environment. This may be changing: The *Standard Template Library* is an example of such a component library for which these guarantees are given. This library has been included in the ANSI/ISO C++ draft standard, so it should eventually be widely known and used. And, it's to be expected that this kind of asymptotic performance guarantee will be provided for future libraries as well, now that this precedent has been set.

Design of Algorithms

A developer who's been in the business for a few years and is advancing through the ranks will be expected to do much more than produce source code based on a design that's been handed to him, or to develop software by choosing from "off the shelf" components.

Such a person might not be directly involved with software development at all — for example, he (or she) might be *managing* software development instead.

On the other hand, she (or he) might now be expected to design new solutions for problems. This might include **developing new algorithms** rather than using old ones. If performance on large inputs is a concern then she or he will need to know something about techniques that can be used to avoid things like "brute force" or "exhaustive" searches — techniques that are useful for designing algorithms that are asymptotically efficient (so that they scale up well).

Complexity Theory

Unfortunately, these techniques won't always be good enough — sometimes, because our knowledge of algorithm analysis and design is incomplete, but (worse yet) sometimes because a problem doesn't have an asymptotically efficient solution at all.

If you've taken CPSC 313 then you should know about some problems that are provably undecidable, so that it's *provably impossible* to solve them reliably by computer (on arbitrary inputs).

There is another class of problems that are decidable but that are also known to be *intractable*, so that they have no *asymptotically efficient* solutions.

There's also a much larger class of problems that are widely *believed* to be intractable and whose complexity has been *proved* to be related, so that if any one of them has an efficient solution then they all do.

Many of the problems that are provably undecidable, or provably intractable, seem to be quite artificial (though this certainly isn't always the case) — that is, they're only of interest because

of the properties they can be proved to possess, and not because very many people would ever be interested in solving them. Others are only encountered if you're trying to use a computer to analyze other computer programs.

Unfortunately, quite a few of the problems that are "widely believed" to be intractable (and whose complexities are related) are quite natural. These problems can be found in numerous application areas, including network design, scheduling, and program optimization (as well as computational number theory, algebra, and so on).

So it wouldn't be surprising at all if a senior developer was asked to produce an efficient solution for one of these problems.

Under the circumstances the developer almost certainly *won't* be able to do this, and shouldn't spend much time trying.

On the other hand, he'll want to keep his job.

So it's appropriate for the developer to recognize the fact that it's likely (or, sometimes, provable) that the problem doesn't have an asymptotically efficient solution; report this fact; and then help to deal with this, by looking for a special case of the problem (or a slightly different problem) that *does* have an efficient solution and can be used instead, or by identifying weaker performance guarantees for the original problem that can be met and that are acceptable (or, by doing something else along these lines). In order to do this much, the developer should **know enough "complexity theory" to realize what results in this area imply**, and should **recognize, or be able to find, problems that are already known or widely believed to be intractable.**

Unfortunately, this still might not be enough.

Computers are still being used in new application areas, so it's still the case that a developer might be asked to solve a problem that hasn't been considered before. The problem might be intractable, but this fact wouldn't have been recognized yet, because the problem is new.

This problem might be completely different from any problems whose status is already known; in this case, it might be quite difficult to show that it's intractable. On the other hand, it might be a somewhat "disguised" version of a problem that's known to be intractable (or "widely believed to be intractable") already; in this case, the job of classifying the problem is generally easier. Fortunately, this second case seems to be more common than the first.

Either way, the developer will want to "extend" our knowledge of complexity theory, by determining the status of the new problem. This will often involve the use of standard techniques for **proving hardness (or "probable intractability") of problems,** especially if the second (easier) case applies — but these techniques are quite different from just looking for the problem in a list of problems that are known to be hard already.

How CPSC 413 Relates to All This

CPSC 413 includes each of the following:

- Techniques to compare asymptotic performance bounds, and to obtain these bounds from source code. These mathematically-based techniques seem to be less interesting to many students than the rest of the course material, so it's unfortunate that this is the first material that will be studied. However, it's easier to assess the algorithms you design and prove results about problems if you know something about algorithm analysis first. That is, the rest of the course builds on this drier material, making it necessary to put it at the beginning.

- Some techniques to design algorithms that are asymptotically efficient — generally, ways to avoid brute force searches.

- An introduction to an important part of complexity theory — the theory of *NP*-completeness. This includes statements of some of the fundamental definitions and results in the area (useful when you need to try to apply the theory, as you encounter problems that are already known or widely believed to be intractable), and a technique for proving that a problem is *NP*-complete, which may be useful when dealing with problems that are new.

This material is mathematically based, emphasizes fundamental (and long lasting) material rather than the latest computer technology, and includes quite a few things that someone *at the very beginning of his career* might not apply — yet. Each of these might explain why you *don't* often find a course like CPSC 413 offered by a community college or institute of technology.

However, the Department of Computer Science is part of the University of Calgary, and it's appropriate that we offer courses that cover the theoretical aspects of our discipline, as well as material that's useful to a senior developer or researcher, who's trying to *extend* what's known already, or to *improve* industrial practices and standards, and not only to use what's known already or follow the practices and standards that are already in place.

Thus, it's completely appropriate that CPSC 413 exists as a university course in computer science. Similar courses are offered by computer science departments in numerous other universities, so it's a reasonably standard offering.

1.3.2 Why CPSC 413 Exists as a 400-Level Undergraduate Course

With the possible exception of the material on algorithm analysis at the beginning (which could form part of a 300-level course if there was room for it), all of the material in CPSC 413 is sufficiently advanced, or requires enough of a background in mathematics and computer science, for it to belong at the 400-level or above.

In particular, some of the material at the end of CPSC 413 builds directly on material that's included at the end of CPSC 313, so CPSC 313 really is a necessary 300-level prerequisite.

On the other hand, CPSC 413 *introduces* algorithm analysis, algorithm design, and complexity theory. There's lots of other material that an undergraduate student may wish to study that depends on material in CPSC 413.

For example (depending on who teaches it, and students' preferences), CPSC 517 may include material on the design and analysis of "randomized" algorithms, parallel and distributed algorithms, and cryptographic protocols, as well as other material building on the information about algorithm design and analysis included in CPSC 413.

CPSC 511 continues the presentation of complexity theory started with CPSC 413's introduction to *NP*-completeness.

A few other courses build on CPSC 413 in a less direct way. For example, CPSC 513 ("Artificial Intelligence") sometimes includes material on pruning searches — techniques that are useful for game playing. This includes algorithms and heuristics that don't reduce the cost of searching quite as dramatically as CPSC 413's techniques can, when the techniques from CPSC 413 are applicable, but these other algorithms and heuristics are applicable in cases when techniques covered in CPSC 413 aren't. It is useful to know about the techniques covered in CPSC 413 ahead of time in order to appreciate the techniques covered in other courses.

So, CPSC 413 is a 400-level course because its material *can* be introduced at this level, and because there are other courses that build on material this course presents and that should be offered as undergraduate courses as well.

1.3.3 Why CPSC 413 is Required

As discussed in Section 1.3.1, the material covered in CPSC 413 really can be of value for researchers in computer science and for developers as they rise through the ranks.

CPSC 413 is also one of a very small number of "theory" courses that students are required to take in order to earn a BSc in Computer Science, and it is frequently the *only* course they take in which they're required to apply both calculus and discrete mathematics in order to solve problems in computer science, and in which students are required to read and write nontrivial proofs that include both written English *and* mathematical notation.

Thus, this course requires students to apply mathematics and exercise their communications skills in ways that most other required courses don't. Because of *this*, it's also considered to be valuable by researchers and developers who aren't very interested in "algorithm design and analysis" at all.

Finally, a course in the design and analysis of algorithms is now a fairly common requirement for university-level computer science programs; requiring CPSC 413 helps to make our program consistent with others.

1.4 About These Notes

These notes have been produced by Professor Wayne Eberly and are based on lecture material prepared for CPSC 413, mostly in 1996-97, as well as earlier material contributed by Professors Lisa Higham and Richard Cleve, who've also taught the course in recent years. Cormen, Leiserson, and Rivest's book, *Introduction to Algorithms* [3], has been used as a textbook for this course in previous years and has undoubtedly been an influence. This book would probably be a useful for reference for students who don't find these notes to be sufficient.

These notes are intended to *supplement* but not *replace* the lectures.

In particular, this course includes *lots* of material that must be covered. In past years, some students have been so busy copying material down during lectures that they haven't had time to listen to the presentation of material, or think or ask questions about it.

Because of this, these notes are *deliberately* very similar to the material that has been (and will) be presented in the lectures — they're supposed to replace most of the *notes* that students take during the lectures, so that students can participate more effectively during the classes. Students who expect the lectures to be *substantially* different from the notes will be disappointed!

On the other hand, the notes can't include or respond to questions that students may ask *during* those lectures. Furthermore, while *many* of the examples that will be taken up in class are also included in these notes, the instructor may give additional examples during the lectures as well, when this seems to be desirable.

Thus, it's the instructor's intention that students will be able to read these notes after — or, preferably, *before* — they attend lectures in which the same material is presented, so that they can get more out of the lectures themselves.

1.5 Other Resources for CPSC 413 Students

1.5.1 Math Review During Block Week

As in 1997, there will be a non-fee non-credit block course, held during the week before Fall lectures begin, which reviews material from the courses in mathematics that are prerequisites for CPSC 413.

Lecture notes for this block course are included in these notes; additional material may be provided on the course web page (which is mentioned below).

1.5.2 Access to Instructor

In Fall 1998, the instructor for CPSC 413 will be Professor Wayne Eberly (Office: MS 632; Telephone: 220-5073; Email: `eberly@cpsc.ucalgary.ca`).

The instructor's office hours will be announced on the course web page and listed on the course information sheet distributed during the first week of class.

1.5.3 Course Web Page

See `http://www.cpsc.ucalgary.ca/~eberly/Courses/CPSC413/` for the instructor's web page for this course. This will include much of the material that's in these notes, as well as additional online tests and solutions, some course news, and a mechanism for making comments about the course online.

1.5.4 Course Newsgroup

The course newsgroup, `cpsc.courses.cpsc413`, will be used to post course announcements and answers to students' questions that are of general interest. This is a "local" newsgroup, so you'll have to use a computer account provided to you by the Department of Computer Science at the University of Calgary in order to read messages on or post messages to this newsgroup.

Note that, while you certainly *can* post messages to this newsgroup, this is not an ideal way to make comments about any aspect of the course; the facilities that are provided on the course web page should be considered first!

1.5.5 Library Material

Several reference books have been placed on reserve for CPSC 413 in the University Library.

For additional information about how to read and produce proofs, see Solow's book, *How to Read and Do Proofs* [15]. While Polya's book, *How to Solve It* [11], isn't on reserve, it's widely available and would also be a useful reference for this topic.

Several references provide more information about discrete and combinatorial mathematics and the applications of this material to analyze algorithms. See, in particular, Graham, Knuth, and Patashnik's *Concrete Mathematics* [4], Grimaldi's *Discrete and Combinatorial Mathematics* [5], and Rawlins' *Compared to What?* [12]. Sedgewick's *Algorithms in C* [13] also includes a chapter on algorithm analysis, as well as material on data structures that you may have seen in CPSC 331.

Several references on the design and analysis of algorithms could be used as the textbook for a course like CPSC 413, including Aho, Hopcroft, and Ullman's *The Design and Analysis of Computer Algorithms* [1], Brassard and Bratley's *Fundamentals of Algorithmics* [2], Cormen, Leiserson, and Rivest's *Introduction to Algorithms* [3], Horowitz, Sahni, and Rajasekaran's *Computer Algorithms/C++* [6], and Neapolitan and Naimipour's *Fundamentals of Algorithms using C++ Pseudocode* [10].

Finally, Knuth's *The Art of Computer Programming* [7, 8, 9] includes more detailed information about many of the topics covered in both CPSC 331 and 413.

1.6 How to Succeed in This Course

Many (but not all) students find CPSC 413 to be a difficult course. Here are some suggestions of things you can do to increase the likelihood that you'll do well in it.

Of course, it takes time to follow each of these suggestions, and there *will* be other things that you'll need to do during the term. It's up to you to decide how much time you can afford to spend on this course.

It's also important to pay attention to this course *early on*, because so much of the material in this course depends on earlier material (so that you generally need to understand the earlier material and know how to solve the earlier problems, in order to be able to deal later with the later ones), and since the early term work counts for quite a bit of your final grade.

1.6.1 Attend the Block Course (If You Need To)

Much of the material in the prerequisite mathematics courses will be applied during the first month of term. Therefore, it's to your advantage to attend the math review course, that will be held during block week before the beginning of term, if you've forgotten the material in these prerequisites — you might not have time to review this material after term begins!

1.6.2 Prepare for and Attend the Lectures

As mentioned above, these lecture notes are intended to *supplement* the lectures instead of replacing them — if you skip the lectures because these notes are available then (I think) you're decreasing the chance that you'll do well in the course.

As time permits I would suggest that you try to read material in these lecture notes *before* the same material is covered in these lectures and then prepare questions to ask about the material during the lectures, when the material is being presented.

Even if you don't have time to read the notes ahead of time, I'd suggest that you try *not* to spend the lectures writing the same material down all over again! If you do, then you won't have time to think about the material, as it's being presented.

Finally, these notes are a work in progress; I'd be happy to hear about changes that could be made, in order to make them more useful the next time around (but, on the other hand, the time I have to work on them is limited, so that it might not be possible to make all changes, even if they really would improve the notes).

1.6.3 Prepare for and Attend the Labs

In this course you will have several chances to learn how to solve each of the kinds of problem that will be covered and then prove that you've learned to do so.

In particular, a "problem set" will always be made available that includes exercises that will discussed in labs, and this will happen several days before these exercises are discussed — *in hopes that you attempt the exercises before the labs.*

The teaching assistants will have been supplied with the instructor's solutions for the same exercises and will use these during the labs. Thus, if you solved the problems ahead of time, then you'll have an opportunity to compare your solutions with the instructor's before being tested.

On the other hand, if you attempted the problems but were *unable* to solve them, then you'll be able to focus on the steps you couldn't complete.

Either way, you'll be better off if you attempted the problems before attending the labs then you would have been if you simply attended the labs without thinking about the problems first: It's easier to understand someone else's solution of a problem than it is to solve it on your own. Thus, the fact that you *did* understand the instructor's solution of a problem doesn't imply that you'll be able to solve a similar problem later on, and you don't want to discover that you *can't* solve this kind of problem on your own in the middle of a test!

Finally, skipping the labs completely would be even worse than attending them without having prepared for them first.

1.6.4 Study Effectively for the Tests

It takes time for some of this material to sink in; cramming shortly before the tests isn't going to work for CPSC 413.

Note, by the way, that there are some things you can assume *will* be tested. If the instructor has taken the trouble to highlight a definition of a technical term in class, then there's a good chance that you'll be asked to *give* (or, sometimes, explain) the same definition on a test, later on. If the instructor takes the trouble to specify a set of steps that you can follow in order to solve a given problem, then it's to be expected that you will be able to use the same set of steps to solve a problem on a test — and that you'll be able to earn part marks by correctly *describing* these steps, even if you aren't able to apply them (provided that they really *are* the right set of steps to apply, for the problem that's been assigned).

1.6.5 Read and Write *Carefully* and *Precisely*

Part of the purpose of this course is for you to exercise skills in technical reading and writing. It's therefore important for you to read and write material carefully and precisely on assignments (if there are any) and tests.

Students occasionally waste time and lose marks on tests in this course because they haven't read a question carefully and end up answering a question that's different from the one that was asked. It's generally worthwhile to read questions carefully and completely before beginning to answer them, even if the test seems long.

It's also important to try to *plan* how you'll solve a question before you start to write out a solution. In particular, most of the questions on tests and exams in CPSC 413 can be solved by applying one or more techniques introduced in class. There will be frequently be more than one technique that could be applied, and the techniques won't always be easy to apply. Under these circumstances, you'll need to think about the question and choose an appropriate technique to apply, before you start to write your answer down. It'll generally be a good idea to explicitly *identify* the technique you chose at the beginning of your answer. If you get stuck, so that you can't complete your answer, then (as mentioned above) it will generally be worthwhile to describe the steps you'd planned to follow in order to finish, even though you couldn't complete them, so that a marker can verify that you had a plan in mind (and give you credit for it).

You will not be penalized on tests for errors you make in spelling or grammar, unless these are *so* serious that it isn't clear what you're trying to say.

On the other hand, you will *not* be allowed to mistake one technical term for another, if their meanings are different (that is, you'll be penalized for doing this when marking assignments and tests — even if the marker realizes that this is what you've done!).

The markers will always *mark what you've written down*, even if they suspect that it isn't really what you meant — and they won't give marks back again, later, if you tell them that you really

meant something else, after material has been marked and returned.

On the other hand, of course, if they really misread something, then you should point this out (and expect a correction of your mark)!

Part II

Math Review for CPSC 413

Chapter 2

Proof Techniques (Review of Math 271)

2.1 Overview

This chapter reviews proof techniques that were probably introduced in Math 271 and that may also have been used in a different way in Phil 279. The first subsection introduces standard proof techniques that will be used repeatedly in proofs presented in CPSC 413. The second subsection presents what will probably be the most important proof technique for this course — mathematical induction. It includes a "standard" form of mathematical induction, as well as several variations that will be needed for algorithm analysis.

It's expected that you've seen *all* the proof techniques included in Section 2.2, below, although you might not have seen the names that have been given to these techniques here. It's also expected that you've seen at least the "standard" (simplest) form of mathematical induction presented in Section 2.3.

On the other hand, some of the other versions of mathematical induction might be new to you! They're discussed here because they will be used, repeatedly, in CPSC 413, so it's worthwhile to study them now.

2.2 Some Common Proof Techniques

2.2.1 General Problem

In general, you're asked to establish that some assertion P is true. In a course like Philosophy 279, P is typically a statement expressed using a formal logical system, possibly propositional logic or first-order logic. In Computer Science 413 this is more frequently a statement about numbers, functions, data structures, or even computer programs written in (one hopes, simple and clear) English, or in a combination of English and mathematical notation.

When proving P you'll generally be allowed to use a set of *axioms* — statements that you may assume to be true without having to prove them — and you'll be able to use a set of *deduction rules* or *rules of inference* in order to prove or "deduce" some assertions from others. A few of these rules of inference, and methods for constructing proofs using them, are given in the rest of this section.

2.2.2 Direct Proofs

In a "direct proof" (or "forward proof"), you start by working with what can been assumed — namely, the axioms you've been given — and you apply deduction rules to list other statements that are *logically implied by* the ones you've listed already, until you've managed to include the desired assertion P. One deduction rule that's frequently useful here is *modus ponens*, which allows you to deduce P from Q and $Q \to P$, for any (other) assertion Q.

This "direct" method (of working forward from assertions to the desired conclusion) is often *not* the way you *discover* a proof — sometimes you start with P and work backwards, and sometimes you do a bit of both (working forward while it's clear how to proceed, and also working backward from desired conclusions in order to list a few other things that *imply* the conclusion and that might be easier to prove first).

Even though this repeated forward and backward reasoning may be the way you *develop* a proof, it's often best if you *write it up* purely as a "forward" proof: That is, write it as if you started with the axioms and applied deduction rules to establish additional things that were implied, including and using axioms as needed, until P was included (and as if you never worked "backwards" at all). When you do this you'll be listing things in an order that's different from the one in which you originally considered things, and you'll prune out "false starts," et cetera, but the result will generally be more readable (and seem better organized) than it would have been otherwise.

This does have the disadvantage, sometimes, of making it less clear *how* the proof was discovered, that information isn't strictly necessary, even though it might be interesting, and including it can make the resulting proof harder to follow.

2.2.3 Introducing and Discharging Assumptions

You'll frequently want to prove a conditional statement — that is, an assertion P that has the form

$$A \to B$$

for simpler assertions A and B.

In this case, you can proceed by

1. Adding A to your set of axioms that you'll consider to be true, by saying, "Assume A" (or, perhaps, "Suppose that A");
2. Proving B;
3. Noting that since you only assumed A in order to do this, you've proved $A \to B$ ("unconditionally" — that is, assuming only the original set of axioms *without* A).

This gives you one more "axiom" to work with and leaves you with a simpler assertion (B) to prove, so this technique will be used virtually every time you need to prove a conditional.

2.2.4 Proof of Contrapositive

The claim

$$A \to B$$

is logically equivalent to the claim

$$\neg B \to \neg A$$

In other words,

$$(A \to B) \text{ if and only if } (\neg B \to \neg A).$$

Sometimes, it's easier to try to prove the claim in the form $\neg B \to \neg A$ (that is, using $\neg B$ as a new assumption and $\neg A$ as the desired conclusion) than it is to prove $A \to B$ more directly.

By the way, in case you're used to seeing a different symbol for this: $\neg$ stands for "not," so that $\neg A$ is the negation of A.

Note, though, that $A \to B$ is *not* generally equivalent to $B \to A$! It's a common *mistake* to "argue in the wrong direction," by trying to prove "$A \to B$" by assuming B and then deducing A (instead of assuming A and deducing B).

2.2.5 Proof by Contradiction

Another (correct) way to proceed is to prove $A \to B$ is to assume both that the assumption A is true, but that the desired conclusion B is *false*, and then "prove" a contradictory statement ("establish a contradiction") without introducing any new assumptions. That is, if you can "prove" that

$$(A \wedge \neg B) \to (0 = 1)$$

or, perhaps

$$(A \wedge \neg B) \to (C \wedge \neg C)$$

for some other proposition C, then you can safely conclude that

$$A \to B$$

as desired.

2.2.6 Proof by (Counter)Example

If the desired conclusion P has the form

$$\exists x Q(x),$$

so that $Q(x)$ is a statement about some unknown value "x," and the claim is that there's some value v for x such that $Q(v)$ is true, then you can prove this by establishing the conclusion

$$Q(u)$$

instead, for any value u that you care to choose.

Similarly, one way to prove that

$$\forall x Q(x)$$

is *false* is to prove that

$$Q(u)$$

is false, for any value u you choose (and, for which this can be proved).

2.2.7 Proof by Universal Generalization

The previous technique (establishing $\exists x Q(x)$ by proving $Q(u)$) may seem obvious; the next one probably won't, but you'll frequently need to use it too.

Suppose you wish to prove

$$\forall x Q(x).$$

One way to do this is to prove

$$Q(u)$$

where the element u is **arbitrarily chosen** — meaning that the element "u" doesn't appear in any assumptions or axioms that you're allowed to use in order to establish the desired result, and that there are no (known) properties it has that other elements of the domain don't have as well. Sometimes (in Phil 279) such an element u is said to be "new."

Proving something about a specific "generic" element u is conceptually easier than trying to prove the same thing about all elements at once. Here's an informal justification for the correctness of this technique: If you managed to prove $Q(u)$ when u is arbitrarily chosen, and if v is *any other element,* then you should be able to replace every occurrence of u in your proof by an occurrence of v, and end up with a proof of $Q(v)$ instead. Since you can do this for *any* element v, $\forall x Q(x)$ must be true.

Note that u really must be "arbitrarily chosen," or "new," if this is to work: If, instead, u *was* mentioned in axioms or prior assumptions, then replacing u by v in one of these might turn a true (and known) statement into a false (or unknown) one, so that you might turn a correct proof into an incorrect one. The fact that u is "arbitrarily chosen" keeps this from happening.

For example, suppose you'd established (or was given as an axiom) that 2 is a prime number. Then you couldn't prove that "every integer greater than or equal to one is prime" by choosing "u" to be an "arbitrary" integer, "say, $u = 2$," using the previous proof or axiom to deduce that u is then prime, and then concluding that since "u is arbitrarily chosen and is prime, all integers that are greater than or equal to one are prime." The problem with this "proof" is that (since you've set it to be equal to 2), u isn't an "arbitrarily chosen integer that is greater than one" at all. While it's probably obvious that this "proof" is incorrect, you should make sure you don't ever make the same mistake in a more subtle way: Once you've declared u to be an "arbitrarily chosen" value, you can't really assume that u has any properties at all, unless you've proved already that *every* value in the given range has them.

2.2.8 Proof by Case Analysis

Suppose you've managed to establish that at least one of a fixed number of *conditions* or *cases* must always hold (but that you don't necessarily know which one holds at any time). Then in order to prove an assertion P, it's sufficient to take *each* case and to prove P assuming that the case holds. Since your cases are *exhaustive* — at least one holds — you can conclude that P is always true, once you've done this.

This method is helpful because it allows you to assume a bit more — namely, that a given case holds — as you begin each proof of P, so that you have a bit more to work with. Of course, it has the disadvantage of forcing you to develop several proofs (one per case) instead of a single one.

Because of this, it's sometimes worthwhile to look back at your proofs for all the cases when you're done, to make sure that each *uses* the assumption that the given case holds. If you discover

that any of them *doesn't* use this assumption, then this is an "unconditional" proof of P — meaning that it's valid in all cases, so that you can throw away the proofs for the other cases and use this single proof, without a case analysis, to establish P.

2.2.9 A First Example

This proof isn't particularly interesting, but it does use the last two proof methods.

It also requires the following "axiom," which we'll assume to be true without proof.

Axiom 2.1 (Division Theorem). If a and b are positive integers then there exists an integer *quotient* q and an integer *remainder* r such that $0 \leq r \leq b-1$ and

$$a = bq + r.$$

Now, here's what we want to prove:

Claim. For every positive integer x, either x is even or $x+1$ is even (or both). That is, either $x = 2k$ for some integer k or $x+1 = 2k$ for some integer k, or both.

That is, we want to prove

$$\forall x(E(x) \vee E(x+1))$$

where E is the predicate "is even," and the quantification ranges over all positive integers.

Proof. Let n be an arbitrarily chosen positive integer.

By the Division Theorem, since n and 2 are positive integers, there exists an integer quotient q and an integer remainder r such that $0 \leq r \leq 2-1 = 1$ and

$$n = 2q + r. \tag{2.1}$$

Since $0 \leq r \leq 1$ and r is an integer, either $r = 0$ or $r = 1$. We will prove that "either n is even or $n+1$ is even" in each of these two cases.

In the first case ($r = 0$) it follows by equation (2.1) that $n = 2q$ and, since q is an integer, n is even. This clearly implies that "either n is even or $n+1$ is even."

In the second case ($r = 1$), equation (2.1) implies that $n = 2q+1$, so that

$$n + 1 = 2q + 2 = 2(q+1).$$

Since q is an integer, $q+1$ is an integer as well, so $n+1$ is even. Again, this clearly implies that either n is even or $n+1$ is even."

It follows by the above case analysis that "either n is even or $n+1$ is even" is (always) true.

Finally, since n is arbitrarily chosen, it follows (by universal generalization) that either x is even or $x+1$ is even, for every positive integer x. □

You should make sure you understand why this proof by "universal generalization"establishes the desired result.

The "Division Theorem" was used here to identify an exhaustive set of two cases, essentially that either n is congruent to 0 modulo 2 or n is congruent to 1 modulo 2. Once these cases were identified, a "proof by case analysis" could be used to establish the desired property for the "arbitrarily chosen" integer n.

2.3 Mathematical Induction

Recall that the *natural numbers* are the integers that are greater than or equal to zero — that is, the elements of the set $\{0, 1, 2, \dots\}$.

It'll often be true in CPSC 413 that we wish to prove that "$P(n)$" is true for every natural number n, where $P(x)$ is a some property of an unknown value x. "Mathematical induction" is a technique for proving things of this form, and it will be used in CPSC 413 over and over again.

2.3.1 Standard Form

Definition

The standard form of a proof that $P(n)$ is true for every natural number n has two pieces:

1. *Basis:* A proof that $P(0)$ is true;
2. *Inductive Step:* A proof that $P(n) \rightarrow P(n+1)$ for every natural number n.

The assumption "$P(n)$" included in the second piece of the proof (the "inductive step") is called the *inductive hypothesis*.

Justification

One way to convince yourself that this is a valid proof technique is to convince yourself that, if you've successfully proved that $P(0)$ is true and also completed the inductive step, then you can prove that $P(c)$ is true, for any *fixed* natural number c, using a longer proof (whose length depends on c) that doesn't use mathematical induction at all. At least, it doesn't use mathematical induction at the top "level:" you might have used induction *again* in either the basis or the inductive step of your original proof.

For example, "$P(0)$" is proved in the basis, so this works for $c = 0$. To prove $P(1)$, use the basis to conclude that $P(0)$ is true, and then use the inductive step to conclude that

$$P(0) \rightarrow P(1).$$

Then deduce (using "modus ponens") that $P(1)$ is true as well. Thus, this also works for $c = 1$.

To prove "$P(2)$," first prove $P(1)$ as above and then use the inductive step, again, to conclude that

$$P(1) \rightarrow P(2).$$

Use modus ponens to deduce that $P(2)$ is also true. Thus this works for $c = 2$.

In general, you'll need to start by using the basis and then use the inductive step $c - 1$ times (for $n = 0, 1, \dots, c-1$), in order to establish $P(c)$ by this technique.

Example

Suppose now that we wish to prove the following.

Claim.

$$\sum_{i=1}^{n} (2i - 1) = n^2$$

for every integer $n \geq 0$.

That is, $P(n)$ is the property $\sum_{i=1}^{n}(2i-1) = n^2$.

Proof. This will be established using induction on n.
Basis: To prove $P(0)$ it is necessary and sufficient to establish the identity

$$\sum_{i=1}^{0}(2i-1) = 0^2.$$

The left hand side is an "empty sum" (a sum of zero terms), and this defined by convention to be 0.

The right hand side is 0^2, which is also equal to 0, so that the identity is correct.
Inductive Step: Now we wish to show that "$P(n) \rightarrow P(n+1)$" for every natural number n. That is, we wish to show that

$$\left(\sum_{i=1}^{n}(2i-1) = n^2\right) \rightarrow \left(\sum_{i=1}^{n+1}(2i-1) = (n+1)^2\right)$$

for every integer $n \geq 0$.

Let m be an arbitrarily chosen integer that is greater than or equal to zero. Suppose (for now) that

$$\sum_{i=1}^{m}(2i-1) = m^2;$$

that is, assume that the "inductive hypothesis" is correct.

Then

$$\begin{aligned}
\sum_{i=1}^{m+1}(2i-1) &= \left(\sum_{i=1}^{m}(2i-1)\right) + (2(m+1)-1) \\
&= m^2 + (2m+2-1) && \text{by the inductive hypothesis} \\
&= m^2 + 2m + 1 \\
&= (m+1)^2 && \text{as desired.}
\end{aligned}$$

Since we only assumed that $\sum_{i=1}^{m}(2i-1) = m^2$, this implies *unconditionally* that

$$\left(\sum_{i=1}^{m}(2i-1) = m^2\right) \rightarrow \left(\sum_{i=1}^{m+1}(2i-1) = (m+1)^2\right).$$

Now, since m was an "arbitrarily chosen" integer that is greater than or equal to zero, it follows (by "universal generalization") that

$$\left(\sum_{i=1}^{n}(2i-1) = n^2\right) \rightarrow \left(\sum_{i=1}^{n+1}(2i-1) = (n+1)^2\right)$$

for all $n \geq 0$, as is needed to complete the inductive step.

Conclusion:

It follows by induction on n that

$$\sum_{i=1}^{n}(2i-1) = n^2$$

for all $n \geq 0$. □

Comments on This Proof: Note the use of "universal generalization" in the inductive step.

It will *always* be the case that universal generalization is needed at this point in the proof. From now on this won't be used quite so "explicitly" as it is here — in particular, this will be the last time that the variable is renamed (to call it m instead of n) in order to highlight the fact that universal generalization is being used.

The inductive step used another trick that is often useful when proving something about a summation (especially in the inductive step for a proof by induction): we split the sum into two pieces — one that we knew something about (the inductive hypothesis), and another that was easy to compute.

The inductive step (specifically, the chain of equations near the middle of it) is part of the proof that might have been discovered differently than it was written up. One might have started the sum and got stuck, then worked "backwards" from the desired final result, $(m+1)^2$, hoping to meet somewhere in the middle. After managing to use algebraic manipulation to equate both the left hand side and the right hand side to some third expression (in this case, most likely, "m^2+2m+1"), one could reverse the sequence of equations relating the middle value to the right hand side and then concatenate the two sequences of equations together — obtaining the sequence of equations shown above, as the form of the proof to be written up.

2.3.2 Variation: A Different Starting Point

Sometimes you don't want to prove something for every integer $n \geq 0$, but you want to prove it for every integer $n \geq k$ instead, for some fixed integer k.

Definition

A proof that $P(n)$ is true for every integer $n \geq k$ also has two parts:

1. *Basis:* A proof that $P(k)$ is true;
2. *Inductive Step:* A proof that $P(n) \rightarrow P(n+1)$ for every integer $n \geq k$.

Note that this is identical to the "standard" form of mathematical induction in the case that $k = 0$.

Justification

One could argue that this works in essentially the same way that one can argue that the standard form of mathematical induction works (as described in the "Justification" in the previous section).

Alternatively, you could argue more formally that this new proof method must be valid if the standard form of mathematical induction is, by "defining" a new property $Q(n)$ of integers as

$$Q(n) \equiv P(n+k).$$

Now, note that since $P(k) \equiv Q(0)$, the "Basis" described above is identical to the "Basis" in a proof that $Q(n)$ is true for all $n \geq 0$ that uses the standard form of mathematical induction. Similarly, the "Inductive Step" described above is pretty much identical to the "Inductive Step" in a proof that $Q(n)$ is true for all $n \geq 0$ that uses the standard form of mathematical induction as well. That is, the two things to be established in the two "Inductive Steps" are obviously logically equivalent, and it's trivial to "transform" a proof needed for the one "inductive step" into a proof that can be used to perform the other.

That is, if you prove both of the pieces given above (for $P(n)$), then you can conclude using "standard" mathematical induction on n that $Q(n)$ is true for all $n \geq 0$.

Finally, you can note that, *by definition,* $Q(n)$ is true for all $n \geq 0$ if and only if $P(n)$ is true for all $n \geq k$.

Example

Let's make things easy (this is going to be long enough as it is) and suppose you'd like to prove the following.

Claim.

$$\sum_{i=1}^{n}(2i-1) = n^2$$

for every integer n such that $n \geq 2$.

That is, the only difference between this and the first example is the different starting point, $k = 2$.

Sketch of Proof. The basis include a proof of the identity

$$\sum_{i=1}^{2}(2i-1) = 2^2,$$

and it's easy to verify that both the left hand side and the right hand side of this identity are equal to 4 (after all, the left hand side is just $1 + 3$).

The inductive step is also exactly the same as the inductive step for the first example, except that you wish to prove that

$$\left(\sum_{i=1}^{n}(2i-1) = n^2\right) \rightarrow \left(\sum_{i=1}^{n+1}(2i-1) = (n+1)^2\right)$$

for every integer n such that $n \geq 2$ (instead of, such that $n \geq 0$). So, you suppose that n is an arbitrarily chosen integer that is greater than or equal to 2 (instead of, greater than equal to 0), proceed exactly as in the proof until you're ready to draw conclusions. Then you observe that "since n was an arbitrarily chosen integer greater than or equal to 2 (instead of 0), the result you've just established must be true by universal generalization for *every* integer $n \geq 2$ (instead of, for every integer $n \geq 0$). Finally, as before, you finish off by observing that you've now proved what you'd hoped to, by induction on n. □

If you're confused about why the variable is called "n" throughout this argument, instead of "n" some of the time and "m" the rest, see the "Comments on This Proof" that followed the proof this is based on.

Finally, here's a tedious but (I hope) straightforward exercise: Write out the proof that was described above, just to make sure you know what it's supposed to be.

2.3.3 Another Variation: Multiple Cases in Basis and Inductive Step

Sometimes it's quite difficult to establish "$P(n+1)$" if you can only assume "$P(n)$," but it's easy to establish $P(n+1)$ assuming $P(n)$ *and* $P(n-1)$ — or more generally, that $P(m)$ is true for the previous "k" values of the integer m, for some positive constant k.

Definition

Suppose k is an integer that is greater than or equal to 1. In order to prove that $P(n)$ is true for every integer n greater than or equal to 0 by yet another form of mathematical induction, it is sufficient to prove the following two things:

1. *Basis:* $P(i)$ is true for every integer i between 0 and $k-1$ (inclusive);
2. *Inductive Step:* For every integer $n \geq 0$,

$$(P(n) \wedge P(n+1) \wedge P(n+2) \wedge \cdots \wedge P(n+k-1)) \rightarrow P(n+k);$$

that is,

$$\left(\bigwedge_{i=0}^{k-1} P(n+i) \right) \rightarrow P(n+k).$$

Note that this is identical to the standard form of mathematical induction in the special case that $k = 1$.

Justification

This time, you could convince yourself that the above rule is valid if and only if the standard form of mathematical induction is, by starting with $P(n)$ and defining $Q(n)$ as

$$Q(n) \equiv \bigwedge_{i=0}^{k-1} P(n+i).$$

Once again, if you were to complete both of the proofs listed above (involving $P(n)$) then you could use these to prove that $Q(n)$ is true for every integer $n \geq 0$, using the standard form of mathematical induction applied to Q. Clearly (if $k \geq 1$),

$$Q(n) \rightarrow P(n)$$

for every integer $n \geq 0$, so that $P(n)$ is true for every integer $n \geq 0$ if $Q(n)$ is.

Example

Definition 2.2. If $n \geq 0$ then the n^{th} *Fibonacci number*, F_n, is defined as follows:

$$F_n = \begin{cases} 0 & \text{if } n = 0, \\ 1 & \text{if } n = 1, \\ F_{n-1} + F_{n-2} & \text{if } n \geq 2. \end{cases}$$

Suppose we wish to prove the following bound.

Claim. If $n \geq 0$ then $F_n \leq 2^{n-1}$.

It will be difficult to prove this using the standard form of mathematical induction, because the recursive definition of F_n (for $n \geq 2$) involves both F_{n-1} and F_{n-2}. It will turn out to be easier to prove this using the form of mathematical induction given above, choosing $k = 2$.
Note: Something like this will generally be the case, whenever you want to prove something about recursively defined functions (recurrences) where the function f_n being defined depends on more than f_{n-1}. It will often be useful to choose "k" to be a positive integer such that f_n is defined (only) in terms of $f_{n-1}, f_{n-2}, \ldots, f_{n-k}$.

Proof. This result will be established using induction on n.
Basis: Since

$$F_0 = 0 \leq \frac{1}{2} = 2^{-1} = 2^{0-1}$$

and

$$F_1 = 1 \leq 1 = 2^0 = 2^{1-1}$$

the claim is true when $n = 0$ and when $n = 1$.

Inductive Step: Now we wish to show that

$$\Big((F_n \leq 2^{n-1}) \wedge (F_{n+1} \leq 2^{n+1-1})\Big) \rightarrow (F_{n+2} \leq 2^{n+2-1})$$

for every integer $n \geq 0$.

Suppose, then, that n is an arbitrary integer that is greater than or equal to 0, and suppose the inductive hypothesis holds — that is, suppose $F_n \leq 2^{n-1}$, and $F_{n+1} \leq 2^{n+1-1}$.

Since $n \geq 0$, $n + 2 \geq 2$, and

$$\begin{aligned}
F_{n+2} &= F_{n+1} + F_n && \text{by definition of } F_n \\
&\leq 2^{n+1-1} + 2^{n-1} && \text{by the inductive hypothesis} \\
&= 2^{n-1}(2+1) \\
&\leq 2^{n-1} \cdot 4 \\
&= 2^{n+1} = 2^{n+2-1}.
\end{aligned}$$

This implies unconditionally that if $F_n \leq 2^{n-1}$ and $F_{n+1} \leq 2^{n+1-1}$ then $F_{n+2} \leq 2^{n+2-1}$.

Since n was an arbitrarily chosen nonnegative integer, *this* implies that if $F_n \leq 2^{n-1}$ and $F_{n+1} \leq 2^{n+1-1}$ then $F_{n+2} \leq 2^{n+2-1}$, for *every* natural number n, as is required to complete the inductive step.

It therefore follows by induction on n that $F_n \leq 2^{n-1}$ for every natural number n. □

2.3.4 A Final Variation: A Much Stronger Inductive Hypothesis

Finally, it's sometimes difficult to prove $P(n+1)$ unless you can assume that $P(i)$ is true, *for every integer* i that is between 0 and n, inclusive. Fortunately, it's possible to assume all this and still prove correctly that $P(n)$ is true for all $n \geq 0$ using a form of mathematical induction.

Definition

A proof that $P(n)$ is true for every integer $n \geq 0$, using this final form of mathematical induction, has two pieces:

1. *Basis:* A proof that $P(0)$ is true;
2. *Inductive Step:* A proof that, for every integer $n \geq 0$, if $P(i)$ is true for every integer i such that $0 \leq i \leq n$, then $P(n+1)$ is true as well — that is, a proof that, for every integer $n \geq 0$,

$$\left(\bigwedge_{i=0}^{n} P(i) \right) \rightarrow P(n+1).$$

Justification

This time you could relate this to the standard form of mathematical induction by defining a property $Q(n)$ of n to be

$$Q(n) \equiv \bigwedge_{i=0}^{n} P(i).$$

If you complete both of the proofs (or "pieces") described above for $P(n)$ then you can use these to produce the basis and inductive step in a proof, using the standard form of mathematical induction, that $Q(n)$ is true for every integer $n \geq 0$. It should be clear that $Q(n) \rightarrow P(n)$ for every n, so this implies that $P(n)$ is true for every integer $n \geq 0$, as well.

Example

Recall the definition of a *binary tree* as it was given in CPSC 331 (and, perhaps, in Math 271 before that). Every node of the tree is either a *leaf* (if it does not have any children) or an *internal node* (if it has at least one child). The *depth* of a binary tree is the length of the longest path (that is, the number of edges) from the root of the tree to any leaf, while the *size* of a tree of the number of nodes in it. By convention, an empty tree — one which has size 0 — is considered to have depth -1.

Suppose now that we wish to prove the following.

Claim. For every integer $d \geq 0$, every binary tree of depth d has size at least $d+1$ and at most $2^{d+1} - 1$.

Notation: In order to shorten the following proof, let $P(d)$ be the assertion that "every binary tree of depth d has size between $d+1$ and $2^{d+1}-1$, inclusive," for any nonnegative integer d. Then the claim asserts that $P(d)$ is true for every nonnegative integer d.

Proof. This will be proved using induction on d.
Basis: If $d = 0$ then there is only one binary tree of depth d: This is a tree with a single node — its root — and this tree has size 1.

Since

$$1 \geq 1 = 0 + 1 = d + 1$$

and

$$1 \leq 1 = 2 - 1 = 2^1 - 1 = 2^{0+1} - 1 = 2^{d+1} - 1,$$

it is true that every tree of depth d has size at least $d+1$ and at most $2^{d+1} - 1$, when $d = 0$. (That is, $P(0)$ is true.)

Inductive Step: We now wish to prove, for every integer $d \geq 0$, that if $P(e)$ is true for every integer e between 0 and d (inclusive) then $P(d+1)$ is true as well.

Suppose now that d is an (arbitrary) integer that is greater than or equal to 0. Next, suppose that $P(e)$ is true for every integer e between 0 and d.

Let T be an arbitrarily chosen binary tree of depth $d+1$. Since $d \geq 0$, $d+1 \geq 0$ as well, and T is nonempty — that is, it has at least one node (its root).

It is therefore possible to split T into three pieces — with each of the last two possibly being empty:

1. the root, u, of T (and any edges connected to the root);
2. the left subtree of T (the subtree whose root is the left child of u, if this child exists); this subtree is empty if u has no left child. We will call this subtree T_L;
3. the right subtree of T (the subtree whose root is the right child of u, if this child exists); this subtree is empty if u has not right child. We will call this subtree T_R.

Since $d \geq 0$, $d+1 \geq 1$ and the root of T cannot be a leaf — otherwise T would have depth at most $0 \leq d$. Therefore T_L and T_R can't *both* be nonempty at once, and u is not a leaf of T.

Let d_L and s_L be the depth and size, respectively, of T_L, and let d_R and s_R be the depth and size, respectively, of T_R. Then

$$s_L, s_R \geq 0$$

since the size of any binary tree is a nonnegative integer, and

$$d_L, d_R \geq -1$$

since the depth of any binary tree is at least -1.

It is clear that the size, s, of the entire tree T is exactly

$$s = s_L + s_R + 1$$

since every node in T is either the root, a node in T_L, or a node in T_R (but it can't be more than one of these at the same time).

It's also true that the depth of T is exactly

$$\max(1 + d_L, 1 + d_R) = 1 + \max(d_L, d_R),$$

since any leaf of in T is either a leaf in T_L or in T_R, and its distance from the root of T is one plus its distance from the root of the subtree (T_L or T_R) to which it belongs.

On the other hand, T was chosen to be a binary tree of depth $d+1$. Therefore (by comparison of the two formulas for the depth of T that have been identified), $\max(d_L, d_R) = d$, so

$$-1 \leq d_L, d_R \leq d \text{ and either } d_L = d \text{ or } d_R = d \text{ (or both)}.$$

Now, there are two cases to be considered — either $d_L = d$ or $d_R = d$ (or both). These cases aren't mutually exclusive — both could be applicable at once — but they are "exhaustive." That is, at least one case must always apply, so that it's sufficient to prove that the desired inequalities are correct in each of these cases in order to complete this part of the proof.

In the first case ($d_L = d$), it follows by the inductive hypothesis that

$$d + 1 \leq s_L \leq 2^{d+1} - 1.$$

Now, since $s_R \geq 0$,

$$s = s_L + s_R + 1 \geq s_L + 1 \geq (d + 1) + 1$$

as desired, and all that is left for this case is to prove the desired upper bound on s as well.

Either T_R is empty, or it is not. If T_R is empty then $d_R = -1$ and $s_R = 0$, so the size s of T is exactly $s_L + 1$. Thus

$$\begin{aligned}
s &= s_L + 1 \\
&\leq 2^{d+1} - 1 + 1 && \text{by the previous inequality} \\
&= (2^{d+1} + 1) - 1 && \text{reordering terms} \\
&\leq 2^{d+2} - 1 && \text{since } 2^{d+1} + 1 \leq 2^{d+2} \text{ if } d \geq 0 \\
&= 2^{(d+1)+1} - 1.
\end{aligned}$$

On the other hand, if T_R is nonempty, then $0 \leq d_R \leq d$, and it follows by the inductive hypothesis that

$$d_R + 1 \leq s_R \leq 2^{d_R+1} - 1.$$

Thus

$$\begin{aligned}
s &= s_L + s_R + 1 \\
&\leq (2^{d+1} - 1) + (2^{d_R+1} - 1) + 1 \\
&\leq (2^{d+1} - 1) + (2^{d+1} - 1) + 1 && \text{since } d_R \leq d, \text{ and } 2^{d_R+1} \leq 2^{d+1} \\
&= 2 \cdot 2^{d+1} - 1 \\
&= 2^{(d+1)+1} - 1,
\end{aligned}$$

as desired. Therefore the desired inequality holds in this first case.

The second case ($d_R = d$) is symmetric to the first: One can establish the same inequality by exchanging the subtrees T_L and T_R (and their depths and sizes) in the argument given above.

Since T is an arbitrarily chosen binary tree of depth $d + 1$, it follows that *every* binary tree of depth $d + 1$ has size between $(d + 1) + 1$ and $2^{(d+1)+1} - 1$, as desired — provided (as assumed in this part of the proof) that the size of every binary tree with depth e is between $e + 1$ and $2^{e+1} - 1$, for every integer e such that $0 \leq e \leq d$.

That is, it's been established that if $P(e)$ is true for every integer between 0 and d, then $P(d+1)$ is true as well.

Since d is an arbitrarily chosen integer that is greater than or equal to 0, *this* implies that if $P(e)$ is true for every integer between 0 and d then $P(d+1)$ is true as well, *for every nonnegative integer* d — completing the inductive step.

It therefore follows by induction on d that every binary tree of depth d has size between $d+1$ and $2^{d+1}-1$, for every integer $d \geq 0$, as desired. □

Comments on This Proof: The two cases in the inductive step ($d_L = d$ and $d_R = d$) are so similar, that it might also have been acceptable to write, immediately after establishing that $d_L = d$ or $d_R = d$, "Suppose, without loss of generality, that $d_L = d$ (otherwise, one can exchange the subtrees T_L and T_R);" and then just to give the proof for the first case, without mentioning the second case again.

This is not the only way that one could prove this result, and it isn't even the only way that one could prove this using induction on d.

Finally, it should be noted that this last form of mathematical induction might be quite useful whenever you're trying to prove something about a data structure (at least, a graph or a tree) if it's recursively defined, or if there's a way to split the structure into two or more pieces and examine the pieces independently — *and*, as above, when you can't relate the "size" (or "depth") of the pieces as closely to the "size" (or "depth") of the original as precisely as you'd need to in order to use one of the other forms of mathematical induction instead.

2.3.5 Additional Variations

If you've understood the rest of this section on mathematical induction, then you might want to try to figure out what some *correct* additional variations on mathematical induction might look like.

For example: What form of "mathematical induction" might you use if you needed to prove that something held for every integer $n \geq k$, for some integer k different from 0, when you needed to assume more than just the fact that the property held for the "previous" value, $n-1$? In other words, how could you correctly combine the first variation on mathematical induction that is described here, with the last one?

How could you correctly prove by (some form of) mathematical induction that a property holds for every *even* integer $n \geq 0$?

How could you prove using some form of mathematical induction that some property holds for every integer n that is *less* than or equal to 0?

2.4 One More Comment About Proofs

Each of the proofs given above is *slightly* longer than proofs I'd normally present, in order to make sure that each of the proof techniques being used is highlighted.

You should note that each proof — especially the final one — was essentially in the form of a sequence of paragraphs, and (after replacing a small amount of mathematical notation with the corresponding phrases in written English) each of these paragraphs was a set of complete, grammatically correct sentences that were as clear as I could make them.

Furthermore, the logical relationships between claims used in the proof, and their current status — whether they needed to be proved or had been proved already, whether one implied another, was implied by another, and so on — were also specified whenever this might have been in doubt.

I hope that this made the proofs more readable than they'd have been otherwise. Since someone else will be reading and marking) the proofs *you* develop for CPSC 413, these are things that you should also be true of *your* proofs, when time permits.

2.5 Exercises

1. Prove that either n, $n+1$, or $n+2$ is exactly divisible by 3 for every integer $n \geq 0$.

2. Prove that $n^3 - n$ is divisible by 6 for every integer $n \geq 0$.

3. Recall the Fibonacci numbers, as given in Definition 2.2 (in Subsection 2.3.3).

   Prove that $F_n \geq (3/2)^{n-2}$ for every integer $n \geq 2$.

4. In fact, there exist real numbers α and β such that

$$F_n = \alpha \left(\frac{1+\sqrt{5}}{2} \right)^n + \beta \left(\frac{1-\sqrt{5}}{2} \right)^n .$$

   Use the definition of F_0 and F_1 given above to figure out what α and β must be, and then prove that this formula is correct (for the values of α and β you've discovered) for all $n \geq 2$, as well.

5. Recall that a node of a binary tree is an *internal node* if it has at least one child, and that it is a *leaf* otherwise.

   The *size* of a binary tree is the number of nodes in the tree, which is the same as the sum of the number of internal nodes and the number of leaves.

   A binary tree is a *complete* binary tree if every internal nodes has exactly two children — that is, if there are no internal nodes having only one child each.

   Prove that every nonempty complete binary tree has exactly one more leaf than it has internal nodes.

You'll find hints for Exercises 2–5 in Section 2.6, below, and you'll find a solution for Question 2 in Section 10.1. I recommend that you try to solve these exercises before you look at their hints or solutions!

2.6 Hints for Selected Exercises

Exercise #2: You might find it to be helpful for part of this proof to consider the cases "n is even" and "n is odd" (for some integer n) separately.

Exercise #3: This problem can be solved by developing a proof whose structure is similar to the proof involving Fibonacci numbers in Section 2.3.3.

Exercise #4: You'll need to start by figuring out what α and β are. You can do this by comparing the values for F_0 and F_1 given in the definition of the Fibonacci numbers, to the values (involving α and β) you get using the expression in the question. You should find that you end up with a system of two linear equations in the two unknowns α and β, and that this system has a unique solution.

Once you've done that, use yet another proof by induction whose structure is similar to the one given above.

You might get stuck in the middle of the inductive step, but you may be able to "recover" by working backwards from what you know that you need at the end, at this point.

Here's a final hint: The two bases of powers in the expression in the question,

$$\frac{1+\sqrt{5}}{2} \quad \text{and} \quad \frac{1-\sqrt{5}}{2},$$

turn out to be the roots of the polynomial $x^2 - x - 1$, as you can verify by evaluating at the polynomial using these values for x, or by using the binomial theorem. Of course this isn't obvious by inspection of these values, but this will turn out to be useful (and, this is the missing thing you might "discover" that you need by working backwards at the end, as is suggested above).

Exercise #5: This problem can be solved using a proof by mathematical induction with the same structure as the proof involving binary trees included in Section 2.3.4.

When you complete the inductive step, it may be useful to observe that any complete binary tree of size greater than one has a left and right subtree of the root that are *both* nonempty.

Chapter 3

Limits and Derivatives (Review of Math 249 or 251)

3.1 Overview

This is the first of two chapters reviewing material from calculus; limits and derivatives are discussed in this chapter, and integrals will be discussed in the next.

While formal definitions of limits and derivatives are included, for the sake of completeness, these *won't* be used very much in CPSC 413; you should concentrate more on the rules for *computing* limits and derivatives that this chapter includes.

3.2 Limits

3.2.1 Informal Introduction

You should remember from Math 249 or 251 that "limits reveal the behaviour of a function near a point."

Consider, for example, the function $f(x) = x^2 + 1$. A graph of the function near $x = 1$, and the function values $f(0.9) = 1.81$, $f(0.99) = 1.9801$, $f(0.999) = 1.998001$, as well as $f(1.1) = 2.21$, $f(1.01) = 2.0201$, and $f(1.001) = 2.002001$, should both suggest that the limit of the value of this function as x approaches 1 is 2. That is, these should suggest that

$$\lim_{x \to 1} f(x) = 2.$$

Of course, this is $f(1)$ in this case, so that for this example the limit of the value of the function as x approaches 1 is just $f(1)$.

This isn't always true. For example,

- If $f(x) = \frac{1}{x}$ then there is no limit of the value $f(x)$ as x approaches 0: The value diverges to $+\infty$ as x approaches 0 "from the right" (that is, as x takes on progressively smaller positive values), and it diverges to $-\infty$ as x approaches 0 "from the left" (that is, as x takes on progressively smaller *negative* values).

- If $f(x) = \frac{\sqrt{x}}{x}$ then the limit of $f(x)$ does not exist as x approaches 0. This time, though, the value approaches 1 from the right and -1 from the left (and, again $f(0)$ is not defined).

- If $f(x) = \frac{\sqrt{x^2}}{|x|}$, then the limit of $f(x)$ exists as x approaches 0, and the limit equals 1, but $f(0)$ is not defined. In fact, $f(x) = 1$ for all x *except* when $x = 0$.
- Finally, if

$$f(x) = \begin{cases} 1 & \text{if } x \neq 0 \\ 0 & \text{if } x = 0, \end{cases}$$

then the limit of $f(x)$ as x approaches 0 exists, and $f(0)$ exists, but these values aren't the same: The limit is 1 and the function value is 0.

3.2.2 Formal Definition

Note: This section is, pretty much, "for interest only." That is, it *won't* really be necessary to understand or use the formal definition of a limit in CPSC 413.

"Informally," we say that $\lim_{x \to a} f(x) = L$ if $f(x)$ gets "arbitrarily close" to L as x approaches a.

That is, if you give me any positive real number ϵ that defines a neighbourhood (that is, open interval) $(L - \epsilon, L + \epsilon)$ near the limit L, then I should be able to give back to you another positive real number δ defining a neighbourhood $(a - \delta, a + \delta)$ near the value x is approaching, so that $f(x)$ will lie in "your" neighbourhood whenever x falls into "my" neighbourhood — as long as x is different from a.

A more formal definition, which pretty much says the same thing, is as follows.

Definition 3.1. The *limit of $f(x)$ as x approaches a* exists and is equal to (a finite constant) L,

$$\lim_{x \to a} f(x) = L,$$

if, for every real number $\epsilon > 0$, there exists some real number $\delta > 0$ such that

$$\text{if } \ 0 < |x - a| < \delta \ \text{ then } \ |f(x) - L| < \epsilon. \tag{3.1}$$

Note (again) that this *does not* imply that $f(a) = L$, or even that $f(a)$ is defined, because the definition only mentions a condition that must hold when $|x - a|$ is *strictly* greater than zero.

In CPSC 413 we'll frequently be interested in the limits of function values as x becomes arbitrarily large (as x approaches $+\infty$):

Definition 3.1, Continued:

The limit of $f(x)$ as x approaches $+\infty$ exists and is equal to (a finite constant) L,

$$\lim_{x \to +\infty} f(x) = L,$$

if, for every real number $\epsilon > 0$, there exists some (generally, large) real number $M > 0$ such that

$$\text{if } \ x > M \ \text{ then } \ |f(x) - L| < \epsilon.$$

Finally, the limit of $f(x)$ as x approaches $+\infty$ exists and is equal to $+\infty$,

$$\lim_{x \to +\infty} f(x) = +\infty$$

if, for every real number $U > 0$, there exists some (generally, large) real number $M > 0$ such that

$$\text{if } \ x > M \ \text{ then } \ f(x) > U.$$

If all else fails, then it is occasionally possible to "prove" that $\lim_{x \to a} f(x) = L$ using this definition.

Suppose again, for example, that $f(x) = x^2 + 1$ and let $a = 1$. To prove formally (from the definition) that $\lim_{x \to a} f(x) = L$ in this case, for $L = 2$, suppose that Δ is a small real number, and note that

$$f(1 + \Delta) = (1 + \Delta)^2 + 1 = 2 + 2\Delta + \Delta^2.$$

It is sufficient to consider small values for Δ, so suppose now that $|\Delta| < 1$. Then $|\Delta^2| < |\Delta|$ and it follows from the above that

$$2 - 2|\Delta| < f(1 + \Delta) < 2 + 3|\Delta|$$

in this case. Therefore

$$|f(1 + \Delta) - 2| < 3|\Delta|.$$

Now, given any positive real number ϵ, we can choose δ to the the minimum of 1 and $\epsilon/3$. Then δ is also a positive real number, and it follows from the above derivation that

$$|f(x) - 2| < \epsilon$$

whenever

$$|x - 1| < \delta,$$

establishing (formally) that $\lim_{x \to 1} f(x) = 2$, as desired.

A formal proof that a limit exists, and and that it equals some specified value, generally has the above form: You must use algebraic manipulation to figure out what a specific value for "δ" is, for any given value of "ϵ," so that inequality 3.1 holds.

3.2.3 Limits of Functions Defined on the Integers

The above definition describes the limit of a function that's defined on the real numbers. However, in CPSC 413, we'll frequently want to compute a limit, as the evaluation point *approaches infinity,* of a function that's only defined *on the integers.*

In order to apply the above definition — and the following evaluation rules — we'll just extend the definition of the function we're interested in.

Sometimes it'll be obvious how to do this, because the description of the "function that's only defined on the integers" gives a way to define it at other real values, too. For example, if you want to compute the limit of the identity function $f(n) = n$ it should be clear that you should "extend" this by defining $f(x)$ to be x for every *real* number x as well.

Unfortunately, this isn't always the case. For example, it isn't obvious how to extend the definition of the factorial function $f(n) = n!$ in order to define it on real numbers that aren't integers.

If all else fails, the following trick will work: If we want to compute $\lim_{n \to +\infty} f(n)$, where $f(n)$ is defined on the integers, and it isn't clear how to define a value for f at other real numbers, then we'll define a corresponding function $\hat{f}$ that's defined on the *reals* by the rule

$$\hat{f}(x) = f(\lfloor x \rfloor) \text{ for every real number } x,$$

where, as usual, $\lfloor x \rfloor$ is the largest integer that's less than or equal to x (and is called the *floor* of x). Note that $f(n) = \hat{f}(n)$ for every *integer* n according to this definition.

Now we'll just define $\lim_{n \to +\infty} f(n)$ to be *the same as* $\lim_{x \to +\infty} \hat{f}(x)$, so that the evaluation rules can be applied to the second of these limits, in order to find the value for the first one.

You might be concerned because there are *several* different ways to extend a function that's defined on the integers, to get a function that's defined on the reals. If *any* of the extensions has a limit, then this really will be the limit (as n approaches infinity) of the integer function you started with — and all of the other possible extensions will either have the same limit or won't have any limit at all. Furthermore, the extension suggested above (involving the use of $\lfloor x \rfloor$ as an argument) *will* have a limit if any of the extensions do.

Another way to deal with this problem will be given as part of Theorem 3.6, below.

Finally, please note that the above tricks are only recommended when you want to compute a limit *at infinity* — don't apply them if you want to compute limits as the evaluation point approaches a finite value!

3.2.4 Rules for Calculating Limits

The following rules aren't quite as general as the use of the definition of limit, but they're easier to apply and will be sufficient for CPSC 413 (at least, most of the time):

Theorem 3.2 (Limit of a Constant). *If $f(x) = c$ (that is, f is a constant function), then*

$$\lim_{x \to a} f(x) = c$$

for any real or complex number a, or for $a = +\infty$.

Theorem 3.3 (Sum, Difference, and Product Rules). *Suppose that*

$$\lim_{x \to a} f(x) = L \quad \textit{and} \quad \lim_{x \to a} g(x) = M$$

for functions f and g, finite values L and M, and for any real or complex number a (or for $a = +\infty$). Then

$$\lim_{x \to a} (f(x) + g(x)) = L + M,$$

$$\lim_{x \to a} (f(x) - g(x)) = L - M,$$

and

$$\lim_{x \to a} (f(x) \cdot g(x)) = L \cdot M.$$

Theorem 3.4 (Quotient Rule). *If a, $f(x)$, $g(x)$, L and M are as above, and $M \neq 0$, then*

$$\lim_{x \to a} \frac{f(x)}{g(x)} = \frac{L}{M}.$$

You can extend these rules to *some* of the cases where one or both of L and M equal plus or minus infinity. Unfortunately, there are some cases where you *can't:* The sum

$$(+\infty) + (-\infty)$$

and the quotients

$$\frac{0}{0} \quad \text{and} \quad \frac{+\infty}{+\infty}$$

are all ill-defined — it isn't clear what values they should assume. The last of these (the ratio of $+\infty$ to itself) will arise frequently in CPSC 413, so we need some additional ways to deal with it.

Theorem 3.5 (Cancellation of a Common Factor). *Let f, g, and h be functions, let a be a finite constant, and suppose there is some positive real number δ such that*

$$\text{if } 0 < |x - a| < \delta \text{ then } h(x) \neq 0;$$

then

$$\lim_{x \to a} \frac{f(x) \cdot h(x)}{g(x) \cdot h(x)} = \lim_{x \to a} \frac{f(x)}{g(x)}.$$

Similarly, if there is some (generally large) real number M such that

$$\text{if } x > M \text{ then } h(x) \neq 0,$$

then

$$\lim_{x \to +\infty} \frac{f(x) \cdot h(x)}{g(x) \cdot h(x)} = \lim_{x \to +\infty} \frac{f(x)}{g(x)}.$$

That is, you can "cancel out" the common factor of $h(x)$ in the numerator and denominator without changing the limit in each case. Sometimes this is all you need to do in order to go ahead and compute the limit.

For example, consider the limit

$$\lim_{x \to 0} \frac{3x}{2x}.$$

Let $h(x) = x$; then $h(x)$ is nonzero when x is *close to* 0, in the sense given above, even though $h(0) = 0$, and the limit has the form described above for $f(x) = 3$ and $g(x) = 2$. So we can use Theorem 3.5 to conclude that

$$\lim_{x \to 0} \frac{3x}{2x} = \lim_{x \to 0} \frac{3}{2} = \frac{3}{2}.$$

Here's another fact that is occasionally useful. This doesn't *only* apply to limits of ratios, but it can definitely be useful in this case.

Theorem 3.6 (Sandwiching the Limit). *Let f, g, and h be functions that are defined on the real numbers such that, for a finite constant a and positive real number δ,*

$$\text{if } 0 < |x - a| < \delta \text{ then } f(x) \leq g(x) \leq h(x).$$

Suppose, furthermore, that

$$\lim_{x \to a} f(x) = \lim_{x \to a} h(x);$$

then

$$\lim_{x \to a} g(x) = \lim_{x \to a} f(x)$$

as well.

Similarly, if f, g, and h are functions that are defined on the real numbers and N is a (generally large) constant such that

$$\text{if } x > N \text{ then } f(x) \leq g(x) \leq h(x)$$

and

$$\lim_{x \to +\infty} f(x) = \lim_{x \to +\infty} h(x)$$

then

$$\lim_{x \to +\infty} g(x) = \lim_{x \to +\infty} f(x)$$

as well.

Finally, if f and h are functions that are defined on the reals, g is a function that is defined on the integers*, and N is a (generally large) constant such that (for integers n and real numbers x)*

$$\text{if } n > N \text{ then } f(n) \leq g(n) \leq h(n)$$

and

$$\lim_{x \to +\infty} f(x) = \lim_{x \to +\infty} h(x)$$

then

$$\lim_{n \to +\infty} g(n) = \lim_{x \to +\infty} f(x)$$

as well.

This result can be applied — to compute the middle limit for g — as long as f and h really do provide lower and upper bounds on g within the range you're interested in (that is, near the limit point), and as long as the limits of f and h really do agree.

Here's an example: Suppose you want to compute the limit

$$\lim_{n \to +\infty} \frac{2^n}{n!}$$

It isn't clear how you can do this directly. On the other hand, since

$$n! = \prod_{i=1}^{n} i,$$

it isn't too hard to see that if $n \geq 1$ then

$$\frac{2}{9} \cdot 3^n = 2 \cdot 3^{n-2} = 1 \cdot 2 \cdot \prod_{i=3}^{n} 3 \leq 1 \cdot 2 \cdot \prod_{i=3}^{n} i = n! \leq \prod_{i=1}^{n} n = n^n,$$

so that

$$\frac{2^n}{n^n} \leq \frac{2^n}{n!} \leq \frac{9}{2} \cdot \frac{2^n}{3^n} = \frac{9}{2} \cdot \left(\frac{2}{3}\right)^n. \tag{3.2}$$

Set

$$h(x) = \frac{9}{2} \cdot \left(\frac{2}{3}\right)^x;$$

then clearly h is defined on the real numbers, and

$$\lim_{x \to +\infty} h(x) = \lim_{x \to +\infty} \frac{2}{3} \cdot \left(\frac{2}{3}\right)^x = 0, \qquad \text{since } 0 \leq \frac{2}{3} < 1.$$

We *could* try to set $f(x)$ to be $\frac{2^x}{x^x}$ (based on the inequality given above) and apply the method being described.

However, it'll make things easier if we simply observe that $\frac{2^x}{x^x}$ is nonnegative for all $x \geq 0$, and use the inequality

$$0 \leq \frac{2^n}{n!} \leq \frac{9}{2} \cdot \left(\frac{2}{3}\right)^n$$

for every natural number n that is implied by this observation and by inequality (3.2), above.

Therefore we'll set $f(x) = 0$, $g(n) = \frac{2^n}{n!}$, and $h(x)$ to be as defined above, and we'll set N to be 1. Then

$$f(n) \leq g(n) \leq h(n) \qquad \text{for all } n \geq N,$$

and clearly (since $f(x) = 0$)

$$\lim_{x \to +\infty} f(x) = \lim_{x \to +\infty} h(x) = 0,$$

so we can conclude that

$$\lim_{n \to 0} g(n) = 0$$

as well. Note that this is the limit that we originally wished to compute.

We'll see one more *extremely* useful rule to compute the limits of ratios, after defining derivatives.

3.2.5 Continuous Functions

Definition 3.7. A function $f(x)$ is *continuous at* a if the limit as x approaches a exists and, furthermore,

$$\lim_{x \to a} f(x) = f(a).$$

A function is *continuous on an interval* if it's continuous at every point inside that interval, and it's *continuous* if it's continuous at every real number.

By definition, the limits of continuous functions are extremely easy to compute: the limit of a continuous function is just the value of the function at the limit point.

Polynomials and exponential functions are continuous, and log functions ($f(x) = \log_c x$ for some constant $c > 0$) are continuous on the interval of positive real numbers. You may use these facts without having to prove them in CPSC 413.

Of course, this still leaves us with the problem of computing limits as x approaches $+\infty$.

3.3 Derivatives

3.3.1 Definitions

Suppose $f(x)$ is a function and a is a real number such that f is continuous at a. Informally (or graphically), you can think of $f'(a)$ as the slope of the line that is tangent to the graph $y = f(x)$ at the point $(a, f(a))$. If f is *not* continuous at a then $f'(a)$ doesn't exist.

The derivative of f at a can be defined more formally as a limit:

Definition 3.8. If a function f is continuous at a point a then the *derivative* of f at a, $f'(a)$, is the limit

$$\lim_{\delta \to 0} \frac{f(a+\delta) - f(a)}{\delta}.$$

The function f is said to be *differentiable at* a if f is continuous at a and the above limit exists (otherwise, $f'(a)$ is undefined), and f is said to be *differentiable* if $f'(a)$ exists for every real number a.

If f is differentiable then we'll frequently consider f' to be a function as well — that is, we'll refer to the function $f'(x)$ (or just f'), with the understanding that the value of this function at any real number a is the limit $f'(a)$ given above.

3.3.2 Some Rules for Computing Derivatives

Theorem 3.9 (Derivative of a Constant). *If $f(x) = c$ is a constant function then $f'(x) = 0$.*

Theorem 3.10 (Derivative of a Power). *If $f(x) = x^\alpha$, for any fixed real number α, then f is differentiable and $f'(x) = \alpha \cdot x^{\alpha-1}$. In particular, the derivative of 1 (the function x^0) is 0.*

Theorem 3.11 (Sum, Product, and Quotient Rules). *If functions f and g are both differentiable at a, then*

$$\textit{if} \quad h = f + g, \quad \textit{then} \quad h'(a) = f'(a) + g'(a);$$

$$\textit{if} \quad h = f \cdot g, \quad \textit{then} \quad h'(a) = f(a) \cdot g'(a) + f'(a) \cdot g(a);$$

and

$$\textit{if} \quad h = \frac{f}{g} \quad \textit{and} \quad g(a) \neq 0, \quad \textit{then} \quad h'(a) = \frac{f'(a) \cdot g(a) - f(a) \cdot g'(a)}{(g(a))^2}.$$

Theorem 3.9 and the above sum and product rules can be used to establish that differentiation is a linear operator: If c is a constant and f, g, and h are functions such that $f(x) = c \cdot g(x) + h(x)$, then $f'(x) = c \cdot g'(x) + h'(x)$ as well.

Theorem 3.12 (Chain Rule). *If g is differentiable at a, f is differentiable at $g(a)$, and if $h = f \circ g$ (so that $h(x) = f(g(x))$), then*

$$h'(a) = f'(b) \cdot g'(a), \quad \text{for } b = g(a).$$

3.4 Back to Limits: l'Hôpital's Rule

Now that differentiation has been defined, we can state another extremely useful rule for computing limits of quotients.

Theorem 3.13 (l'Hôpital's Rule). *If a is a real number or $a = +\infty$, and either*

$$\lim_{x \to a} f(x) = \lim_{x \to a} g(x) = 0$$

or

$$\lim_{x \to a} f(x) = \lim_{x \to a} g(x) = +\infty,$$

then

$$\lim_{x \to a} \frac{f(x)}{g(x)} = \lim_{x \to a} \frac{f'(x)}{g'(x)}$$

(provided, of course, that the derivatives shown in the above expression exist). That is, you can differentiate the functions in the numerator and in the denominator, without changing the limit, in these cases.

The first example of the use of l'Hôpital's rule you saw might have been the following: Suppose $f(x) = \sin x$ and $g(x) = x$; then $\lim_{x \to 0} f(x) = \lim_{x \to 0} g(x) = 0$. So, by l'Hôpital's rule,

$$\lim_{x \to 0} \frac{\sin x}{x} = \lim_{x \to 0} \frac{\cos x}{1} = \cos 0 = 1,$$

since $f'(x) = \cos x$ if $f(x) = \sin x$ and $g'(x) = 1$ if $g(x) = x$, and since the cosine function is continuous.

Note, again, that you can *only* apply this rule if either $f(x)$ and $g(x)$ both approach 0 or if they both approach $+\infty$ at the limit you want to compute. On the other hand, you won't *need* this rule in any other case.

3.5 Exponential and Logarithmic Functions

In spite of the fact that the last example involved one, trigonometric functions (sin, cos, tan, and so on) won't be used very much on CPSC 413.

However, the exponential and logarithmic functions will be used frequently, and you should know and be able to apply the basic properties of these functions that are given below.

Here are definitions and derivatives for two important functions (and a constant).

Definition 3.14. *Euler's constant*, commonly written as e, is the limit

$$\lim_{n \to +\infty} \left(1 + \frac{1}{n}\right)^n$$

and is approximately equal to $2.71828\ldots$

Theorem 3.15. *If $f(x) = e^x$ then $f'(x) = e^x$ as well.*

Definition 3.16. If $x > 0$ then the *natural logarithm* of x, $\ln x$, is the (unique!) real number y such that $e^y = x$. Thus if $g(x) = \ln x$ and $f(x) = e^x$ as above, then

$$f(g(x)) = g(f(x)) = x$$

for all $x > 0$.

Theorem 3.17. *If $g(x) = \ln x$ as above, and $x > 0$, then $g'(x) = \frac{1}{x}$.*

We're often interested in exponential and logarithm functions with different bases. You can differentiate these as well, by using the following formulas and the above definitions.

Theorem 3.18. *If $c > 0$ then $c^x = e^{x \ln c}$,*

Theorem 3.19. *If $a, b, c > 0$ then*

$$\log_b c = \frac{\log_a c}{\log_a b}.$$

In particular (when a is Euler's constant),

$$\log_b c = \frac{\ln c}{\ln b}.$$

You should now be able to use the fact that if $f(x) = e^x$ then $f'(x) = e^x$ as well, in order to figure out how to differentiate c^x for $c > 0$ (by using the above formulas and the chain rule). It should be even easier to figure out how to differentiate $\log_b x$ for some constant $b > 0$, given the above information about logarithms (after all, if b is a positive constant, then $\ln b$ and $\frac{1}{\ln b}$ are just constants, as well).

Some additional useful properties of exponential functions and logarithms are given on pages 33–34 of Cormen, Leiserson, and Rivest [3].

3.6 Exercises

1. Compute the each of the following limits or explain why it does not exist.

   (a) $\lim_{x \to 2} \dfrac{x^3 - 8}{x - 2}$

   (b) $\lim_{x \to +\infty} \dfrac{x^2 + 2x + 1}{3x^2 + 5}$

   (c) $\lim_{x \to 0} \dfrac{x}{\cos x - 1}$

   (d) $\lim_{x \to 0} \dfrac{\sin(4x)}{\sin(3x)}$

2. Compute the derivative (with respect to x) of each of the following functions.

   (a) $f(x) = 3x^2 + 2x + 1$
   (b) $f(x) = x \ln x$
   (c) $f(x) = \frac{x}{\ln x}$
   (d) $f(x) = e^{x^2 \ln x}$

3. Derive the quotient rule (as given below), starting with the definition of a derivative, and assuming that the functions f and g are both differentiable at a and that $g(a) \neq 0$.

   Quotient Rule: If $h(x) = \frac{f(x)}{g(x)}$ then

$$h'(a) = \frac{f'(a)g(a) - f(a)g'(a)}{(g(a))^2}$$

4. Prove that

$$\lim_{x \to +\infty} \frac{(\ln x)^n}{x} = 0$$

   for every natural number $n \geq 1$.

Hints for selected exercises are given in the next section; solutions for Exercises 1(c) and 3 are given in Section 10.2.

3.7 Hints for Selected Exercises

Exercise #1(a): Observe that you can cancel out a common factor of $x - 2$ from both the numerator and the denominator of the expression in the limit — or you can use l'Hôpital's rule, instead. (Try using both methods, and confirm that you get the same answer each time.)

Exercise #1(b): You can either treat x^2 as a common factor to be eliminated from both the numerator and denominator, or you can apply l'Hôpital's rule twice.

Exercise #1(d): Apply l'Hôpital's rule. You'll need to treat both the numerator and denominator as compositions of functions and use the chain rule to compute the needed derivatives.

Exercise #2(b): Apply the product rule for derivatives.

Exercise #2(c): Apply the quotient rule for derivatives.

Exercise #2(d): $f(x) = g(h(x))$ for $g(y) = e^y$ and $h(z) = z^2 \ln z$. Apply the product rule to differentiate h, and then apply the chain rule to differentiate f.

Exercise #4: Try to prove this using mathematical induction, by induction on n (with $n = 1$ as your starting point). You'll probably need to apply l'Hôpital's rule in both the basis and inductive step of your proof.

Chapter 4

Antiderivatives and Integrals (Review of Math 249 or 251, Continued)

4.1 Overview

This chapter continues the review of Math 249 or 251, and presents definitions and rules for computing antiderivatives and integrals (also known as indefinite and definite integrals, respectively). Once again, the formal definitions given here are generally less important for CPSC 413 than the evaluation rules that follow them.

At least one of the evaluation rules — "Integration by Parts" — might *not* have been covered in Math 249 or 251. Therefore, while this is an easy rule to learn, it *won't* be required for CPSC 413 assignments or tests; at most, it will be used in the occasional example that's presented in lectures or labs.

4.2 Antiderivatives

Definition 4.1. An *antiderivative* of a function $f(x)$ is another function $g(x)$ whose derivative $g'(x)$ is equal to $f(x)$.

An antiderivative of $f(x)$ is sometimes called an "indefinite integral" of $f(x)$. Any such function is denoted

$$\int f(x)\,dx$$

and is (at most) unique up to a constant term.

4.2.1 Antidifferentiation Formulas

Antiderivative formulas can be obtained, essentially, by "reversing" differentiation formulas. Here are some that you should know.

Theorem 4.2 (Antiderivative of a Power of x). *For any real number n,*

$$\int x^n\,dx = \begin{cases} \frac{1}{n+1}x^{n+1} + C & \text{if } x \neq -1, \\ \ln n + C & \text{if } x = -1. \end{cases}$$

Here (and below), C denotes an arbitrary constant.

Theorem 4.3 (Antiderivative of the Exponential Function).

$$\int e^x \, dx = e^x + C.$$

Theorem 4.4 (Linearity of Antiderivatives). *If c is a constant then*

$$\int (cf(x) + g(x)) \, dx = c \left(\int f(x) \, dx \right) + \int g(x) \, dx.$$

4.3 (Definite) Integrals

Definition 4.5. The *(definite) integral* of a function $f(x)$ between $x = a$ and $x = b$ is denoted

$$\int_a^b f(x) \, dx$$

and may be interpreted as the area of the region bounded by the graph $y = f(x)$, the x-axis, and the vertical lines $x = a$ and $x = b$ (for $a < b$). Any area within this region that is above the x-axis is given a positive value, and any area below the x-axis is given a negative value.

This area is easy to compute if $f(x)$ is a linear function (and you remember how to compute the area of a triangle); it isn't so clear how to compute this, for more general functions.

4.3.1 Fundamental Theorem of Calculus

The following theorem provides a way for us to integrate continuous functions.

Theorem 4.6 (Fundamental Theorem of Calculus). *If $f(x)$ is continuous on the closed interval between a and b, and $F(x)$ is any antiderivative of $f(x)$, so that $F'(x) = f(x)$, then*

$$\int_a^b f(x) \, dx = F(b) - F(a).$$

Thus, we can compute definite integrals if we know how to find (and evaluate) antiderivatives.

4.3.2 Riemann Sums

One way to *estimate* a definite integral, rather than compute it exactly, is to approximate the area under the curve (in the definition of "definite integral") by the sum of the areas of rectangles.

In particular, let

$$x_i = a + i \left(\frac{b-a}{n} \right)$$

for every natural number i such that $0 \leq i \leq n$. For $1 \leq i \leq n$, suppose we approximate the area under the above curve, between $x = x_{i-1}$ and $x = x_i$, by that of the rectangle whose width is $x_i - x_{i-1} = (b-a)/n$ and whose height is $f(x_i)$. The sum of the areas of all these rectangles, for $i = 1, 2, \ldots, n$, is

$$\left(\frac{b-a}{n} \right) \sum_{i=1}^{n} f(x_i).$$

The following theorem states that this sum is an approximation for the definite integral $\int_a^b f(x) \, dx$ if n is sufficiently large.

Theorem 4.7. *Let a and b be real numbers such that $a \leq b$ and let $f(x)$ be a function that is continuous on the closed interval between a and b. Then*

$$\int_a^b f(x)\,dx = \lim_{n\to+\infty}\left(\frac{b-a}{n}\right)\sum_{i=1}^{n} f(x_i).$$

Numerical integration packages use sums like this one — although they generally use ones that converge to the limit faster than this one — in order to estimate the values of definite integrals; these packages are studied in CPSC 491. We'll see a different use for this in CPSC 413 when we consider "summations."

4.3.3 Other Methods for Integration

Finally, here are two more advanced methods to integrate functions.

Integration by Substitution

Recall the chain rule: If $H(x) = F \circ G(x) = F(G(x))$, then $H'(x) = F'(y) \cdot G'(x)$, for $y = G(x)$. We can "reverse" this differentiation formula to obtain an integration formula.

Theorem 4.8 (Integration by Substitution). *If*

$$h(x) = f(G(x)) \cdot g(x)$$

for functions $f(x)$, $g(x)$, and $G(x)$ such that $G'(x) = g(x)$, and if $F(x)$ is a function such that $F'(x) = f(x)$, then

$$\int h(x)\,dx = F(G(x)) + C.$$

In order to apply this we'll generally perform the following steps, given the function $h(x)$.

1. Guess (or somehow find) a function $G(x)$ and set $g(x) = G'(x)$.
2. Define the function f using the rule $f(y) = h(x)/g(x)$, for $y = G(x)$.
3. Somehow find an antiderivative F for the above function f.
4. Return $H(x) = F(G(x))$ as $\int h(x)\,dx$.

For example, suppose that $h(x) = (\sin x) \cdot (\cos x)$, and suppose we "guess" that $G(x) = \sin x$.

Then $g(x) = G'(x) = \cos x$, $y = G(x) = \sin x$, and we have $f(y) = h(x)/g(x) = \sin x = y$. Thus, $f(x) = x$.

Now, an antiderivative of $f(x)$ is easy to find: we can choose $F(x) = \frac{1}{2}x^2 + C$ for any constant C.

We now return $H(x) = F(G(x)) = \frac{1}{2}(\sin x)^2 + C$ as an antiderivative of $h(x)$. Differentiation confirms that $H'(x) = h(x)$, as desired.

For another example, suppose $h(x) = \sin(x+1)$, and guess that $G(x) = x + 1$, so that $g(x) = G'(x) = 1$. Then $y = G(x) = x + 1$, $f(y) = h(x)/g(x) = \sin(x+1) = \sin y$ (so that $f(x) = \sin x$), and we can guess (or look up) an antiderivative of $f(x)$: $F(x) = -(\cos x) + C$.

We now return the antiderivative $H(x) = F(G(x)) = -(\cos(x+1)) + C$; differentiation confirms that $H'(x) = h(x)$, as desired.

Integration by Parts

Now recall the product rule: If $h(x) = f(x) \cdot g(x)$, then $h'(x) = f'(x) \cdot g(x) + f(x) \cdot g'(x)$. Subtracting the last term from both sides (and reversing the left and right hand sides) gives the equation

$$f(x) \cdot g'(x) = h'(x) - f'(x) \cdot g(x),$$

and integrating both sides (and using linearity) gives the following rule.

Theorem 4.9 (Integration by Parts).

$$\int (f(x) \cdot g'(x))\, dx = f(x) \cdot g(x) - \int (f'(x) \cdot g(x))\, dx.$$

This is sometimes useful, because we're given a function of the form $f(x) \cdot g'(x)$ to be integrated, and it's easier to find an antiderivative of $f'(x) \cdot g(x)$, instead.

Suppose, for example, that we wish to integrate the function $h(x) = x \cdot e^x$. Then we can write this as $h(x) = f(x) \cdot g'(x)$, where $f(x) = x$ and $g'(x) = e^x$. Clearly $f'(x) = 1$ and one antiderivative of $g'(x)$ is $g(x) = e^x$, so we have that

$$\int xe^x\, dx = xe^x - \int 1 \cdot e^x\, dx = xe^x - e^x.$$

Differentiating the function on the right confirms that this answer is correct.

4.4 Exercises

Compute each of the following integrals, assuming a and b are positive real numbers such that $b > a > 1$.

1. $\int_a^b \frac{x^3-1}{x-1}\, dx$
2. $\int_a^b (2 - \sqrt{x})\, dx$
3. $\int_a^b e^x\, dx$
4. $\int_a^b x + \frac{1}{x}\, dx$
5. $\int_a^b \frac{1}{x-1}\, dx$
6. $\int_a^b \frac{(\ln x)^3}{x}\, dx$
7. $\int_a^b x^2 e^x\, dx$
8. $\int_a^b \frac{1}{(x-1)(x+1)}\, dx$ (This one is challenging!)

Hints for these problems are given in Section 4.5, below, and a solution for Question 6 appears in Section 10.3.

4.5 Hints for Selected Exercises

This time, *two* sets of hints will be supplied. I suggest that you first attempt the above exercises without looking at any hints at all. Then (if you need them) look at the first set of hints and try again. Finally, use the second set of hints if the first aren't enough.

4.5.1 First Set of Hints

Exercise #1: The integral can be calculated by performing a simple algebraic substitution: Note that $a > 1$ and that $x - 1$ is a factor of the numerator of the expression to be integrated.

Exercise #5: You might calculate this integral by using an application of "integration by substitution" resembling one of the examples given in Section 4.3.3.

Exercise #6: Consider "integration by substitution" for this integral as well. One of the functions involved will be the natural logarithm, $\ln x$.

Exercise #7: Consider integration by parts.

Exercise #8: You'll probably need to manipulate the expression to be integrated in order to compute this integral. Rational functions can be decomposed in a particular way; in particular, there exist *constants* c_1 and c_2 such that

$$\frac{1}{(x-1)(x+1)} = \frac{c_1}{x+1} + \frac{c_2}{x-1}.$$

Since the two expressions are equal, their integrals are equal as well. However, you'll probably find it easier to integrate the version on the right hand side than the one on the left.

4.5.2 Second Set of Hints

Exercise #1: Note that $\frac{x^3-1}{x-1} = x^2 + x + 1$ if $x \neq 1$; that is, these two functions have the same value at any point except $x = 1$.

Exercise #5: To use integration by substitution, "guess" that $G(x) = x - 1$, so that $g(x) = 1$. You should then find that $f(y) = \frac{1}{y}$.

Exercise #7: $\int x^2 e^x \, dx = \int f(x) \cdot g'(x) \, dx$, where $f(x) = x^2$ and $g'(x) = e^x$. Recall that (after reading Chapter 4) you know what $\int xe^x \, dx$ is already!

Exercise #8: See the original hint for this problem, and try $c_1 = \frac{1}{2}$ and $c_2 = -\frac{1}{2}$ (or, perhaps, vice-versa).

Chapter 5

Sample Tests

While there wasn't a "math review" for CPSC 413 at all in 1996, most of the questions on the first class test for that year were essentially on "math review" material, so this test is reproduced below in Section 5.1.

On the other hand, there *was* a math review in 1997. The first class test for CPSC 413 that year was actually a "mock test" — it didn't count — and it was completely based on the material presented in the math review that year. It's reproduced, below, in Section 5.2.

5.1 Class Test for 1996

Instructions:

Attempt all questions. Write answers on the question sheet(s).

No aids allowed.

Duration: 50 minutes.

1. Consider two algorithms, "Algorithm A" and "Algorithm B," for the same problem. Algorithm A uses $4n^2$ steps to solve a problem of size n while Algorithm B uses $100n$ steps to solve a problem of the same size.

   (a) (2 marks) For which inputs sizes (values of n) would Algorithm A be at least as fast as Algorithm B?

   (b) (3 marks) Suppose Algorithm A can be used to solve a problem of size s in time t on one computer. What is the size of a problem that could be solved using the same algorithm, on a new computer that is 100 times as fast?

2. (5 marks) Consider the sequence of values $h_0, h_1, h_2, \ldots$, defined such that $h_0 = 2$, $h_1 = 5$, and $h_{n+2} = 5h_{n+1} - 6h_n$ for $n \geq 0$. Prove that $h_n = 2^n + 3^n$ for all $n \geq 0$.

3. (10 marks) Recall that every node in a binary tree is either an *internal node* (if it has either one or two children) or a *leaf* (if it does not have any children).

   Prove that if a binary tree has exactly l leaves and i internal nodes, then $l \leq i + 1$.

5.2 Class Test for 1997

Instructions:

Attempt all questions. Write answers in the spaces provided below.

No aids allowed.

Total marks available: 25

Duration: 50 minutes.

Note: This quiz will not be marked or count toward your CPSC 413 grade.

1. (4 marks) Compute $\lim_{x \to +\infty} \frac{x \ln x}{x^{1.5}}$.

2. Compute the derivative $f'(x)$ for each of the following functions f.

   (a) (3 marks) $f(x) = \ln(\ln x)$

   (b) (3 marks) $f(x) = \frac{e^x}{x \ln x}$

3. (5 marks) Compute $\int x^2 \ln x \, dx$, and use differentiation to check that your answer is correct.
   Hint: $x^2 \ln x = f'(x) \cdot g(x)$, for $f'(x) = x^2$ and $g(x) = \ln x$. Consider "integration by parts."

4. (10 marks) Suppose

$$b(i,j) = \begin{cases} 0 & \text{if } j < 0 \text{ or } j > i, \\ 1 & \text{if } j = 0 \text{ or } j = i, \\ b(i-1, j-1) + b(i-1, j) & \text{if } 1 \leq j \leq i-1. \end{cases}$$

   Prove that $b(i,j) = \frac{i!}{j!(i-j)!}$ for all $i, j \geq 0$ such that $0 \leq j \leq i$.

   Hint: Consider induction on i; $0! = 1$ and $k! = \prod_{i=1}^{k} i = 1 \cdot 2 \cdot 3 \cdots k$ if $k \geq 1$.

Part III

Analysis of Algorithms

Chapter 6

Asymptotic Notation

6.1 Overview

This chapter includes a formal definition of the "big-Oh" notation that has been used in previous courses to state asymptotic upper bounds for the resources used by algorithms, and introduces additional notation for additional kinds of asymptotic bounds. It also introduces techniques that can be used to compare the rates of growths of the kinds of functions that might arise as resource bounds.

It is expected that CPSC 413 students will know the definitions that are presented here, and that they will be able to apply the techniques for comparing functions that this chapter introduces. Students should also be able to use the definitions to prove various properties of asymptotic notation. A set of exercises and sample tests is given at the end of this chapter (and can be used to assess each of the above skills).

The most generally useful of these techniques require the computation of limits, and differentiation of functions, so it might be helpful to review the material in Chapter 3 before attempting the exercises in this chapter.

6.2 Worst Case Running Time

As suggested above, we're going to be interested in the resources (generally time, but also space) used by algorithms when they are applied to solve problems with inputs that are allowed to be "arbitrarily large."

In order to discuss this, we'll need a notion of the *size* of an input. This will be discussed in more detail in later chapters but, for now, it's sufficient to think of this as being approximately the length of a string of symbols that "represents" the input — which is what the algorithm actually processes. As such, this "input size" will always be a nonnegative integer.

Functions that bound the running times of algorithms will generally be defined on (some of) the nonnegative integers and will generally gave nonnegative real numbers as values.

It'll be convenient to use "partial functions" — which aren't necessarily defined on *all* of the nonnegative integers — to bound running times of algorithms. For example, it'll be convenient to use "$n \log n$" as a function that bounds the running time of an algorithm on input (size) n, even though this function is undefined when $n = 0$.

Therefore, in these notes a function will generally be called a "total function" if it's (supposed to be) defined on *all* the nonnegative integers.

Definition 6.1. The *worst case running time* of an algorithm is a partial function from the set of nonnegative integers to the set of nonnegative real numbers. In particular, the value of this function at any integer $n \geq 0$ is the maximum[1] of the running times used by the given algorithm on (all of the) possible inputs for the algorithm with size n.

The value of this function for *small* inputs is often extremely important. However, in this course, we'll be concentrating on the behaviour of these functions for large inputs instead. As argued in Chapter 1, this can be extremely important for at least some problems and applications.

6.3 Asymptotically Positive Functions

Definition 6.2. A (partial) function f from the set of nonnegative integers to the set of real numbers is *asymptotically positive* if there exists an integer $N \geq 0$ such that

$$\text{for every integer } n \geq N,\ f(n) \text{ is defined, and } f(n) > 0.$$

Note that the last part of this says that $f(n)$ is *strictly* greater than 0, and not just "greater than or equal to zero."

6.3.1 Examples

For example, the function $f(n) = n-1$ is asymptotically positive, since $f(n)$ is defined and $f(n) > 0$ whenever $n \geq 2$. That is, you can set N to be 2 (or, any integer that's greater than or equal to 2) and observe that the function f satisfies the definition with this choice.

On the other hand, the function $f(n) = \sin n$ is *not* asymptotically positive, since there are infinitely many positive integers n such that $\sin n$ is zero or negative (regardless of whether n represents a number of degrees or radians).

Here's another positive example, which includes a more substantial proof.

Claim. Let $f(n)$ be a *polynomial* function of n, in particular,

$$f(n) = c_d n^d + c_{d-1} n^{d-1} + c_{d-2} n^{d-2} + \cdots + c_0,$$

where $c_d, c_{d-1}, c_{d-2}, \ldots, c_0$ are real numbers and c_d is nonzero (so that f has degree d and leading coefficient c_d). Then f is an asymptotically positive function if and only if $c_d > 0$.

Proof of Claim. Since c_d is nonzero, and is a real number, either $c_d > 0$ or $c_d < 0$. In order to prove the claim, it's necessary and sufficient to show that $f(n)$ is always asymptotically positive when $c_d > 0$ and that $f(n)$ is *never* asymptotically positive when $c_d < 0$.

Suppose first that $c_d > 0$. Then, since $c_i \geq -|c_i|$ for $0 \leq i \leq d-1$, if $n \geq 1$ then

$$\begin{aligned} f(n) &= c_d n^d + c_{d-1} n^{d-1} + c_{d-2} n^{d-2} + \cdots + c_0 \\ &\geq c_d n^d - |c_{d-1}| n^{d-1} - |c_{d-2}| n^{d-2} - \cdots - |c_0| \\ &\geq c_d n^d - |c_{d-1}| n^{d-1} - |c_{d-2}| n^{d-1} - \cdots - |c_0| n^{d-1} \\ &= n^{d-1} \left(c_d n - (|c_{d-1}| + |c_{d-2}| + \cdots + |c_0|) \right). \end{aligned}$$

[1] Here's a minor technical point: This definition is sufficient as long as there's always only a *finite* number of inputs of any given size n. You could extend this definition to cases where there are *infinitely* many inputs of a given size by considering a "supremum" instead of a "maximum." If you don't remember what a "supremum" is, don't worry about it — we won't be worrying about this in CPSC 413.

Now, if

$$n > \frac{|c_{d-1}| + |c_{d-2}| + \cdots + |c_0|}{c_d}$$

then (since $c_d > 0$)

$$c_d n - (|c_{d-1}| + |c_{d-2}| + \cdots + |c_0|) > 0.$$

Clearly, $n^{d-1} > 0$ whenever $n \geq 1$. Therefore, if N is *any* integer such that

$$N > \max\left(1, \frac{|c_{d-1}| + |c_{d-2}| + \cdots + |c_0|}{c_d}\right)$$

then (since the product of two positive numbers is always positive, and since f is clearly a total function)

$$f(n) \text{ is defined and } f(n) > 0 \text{ for every integer } n \geq N,$$

implying that f is asymptotically positive.

Suppose, instead, that $c_d < 0$. In this case, it can be shown by a similar argument that $f(n)$ is defined and $f(n) < 0$ for every integer n such that

$$n > \max\left(1, \frac{|c_{d-1}| + |c_{d-2}| + \cdots + |c_0|}{|c_d|}\right).$$

This clearly implies that there are *infinitely many* nonnegative integers n such that $f(n) < 0$ — and $f(n)$ can't possibly be "asymptotically positive" in this case. □

This example has been given in hopes that it will help students to understand the definition of an asymptotically positive function — not because students will be expected to develop similar proofs themselves. On the other hand, students should remember the claimed *result* about polynomials and should be prepared to use it without proof, as needed.

6.3.2 Some Useful Facts

Here are some additional facts about asymptotically positive functions that students should be prepared to use without proof.

Fact 6.3. The functions $f(n) = \log_c n$ and $g(n) = c^n$ are asymptotically positive, for any positive constant c.

Fact 6.4. If f and g are asymptotically positive functions then $f + g$, $f \cdot g$, and f/g are all asymptotically positive functions as well.

Fact 6.5. If f is asymptotically positive and there exists an integer N such that $g(n)$ is defined for every integer $n \geq N$ (as is true, for example, if g is also asymptotically positive), then f^g is asymptotically positive as well.

On the other hand, it *isn't* always true that the *difference* $f - g$ of two asymptotically positive functions f and g are asymptotically positive!

Students who wish to test their understanding of the definition, and who want to exercise their skills in developing proofs, should try to prove each of the above facts.

6.3.3 Extensions of the Definition

It should be noted that you can define "asymptotically positive functions" whose domains are subsets of the set of *real numbers* as well. In this case the definition would require that there is an integer $N \geq 0$ such that

$$\text{for every } \textit{real number } x \geq N,\ f(x) \text{ is defined and } f(x) > 0.$$

All of the definitions that are given below are stated for asymptotically positive functions that are defined on the integers. However, you can modify them, to obtain the corresponding definitions for functions defined on the reals, by making one (and only one) change — and making it all the time: just replace the specification that n is an *integer* by a specification that n is a *real number* — just as happened above, in the definition for "asymptotically positive."

6.3.4 Caution About a Reference

Cormen, Leiserson, and Rivest [3] consider "asymptotically nonnegative functions" instead of "asymptotically positive functions," and define things slightly differently. Their definitions introduce a few more unnecessary complications, so this isn't recommended as a reference for this particular material, even though this book has been used as the textbook for CPSC 413 in the recent past.

6.4 Big-Oh Notation

6.4.1 Definition and Examples

You've used "big-Oh" notation already in previous courses, including CPSC 331. Informally, f is in big-Oh of g ("$f \in O(g)$") if f grows asymptotically at least as slowly as g does; here's a more formal definition that you might not have seen before.

Definition 6.6. If f and g are asymptotically positive functions then $f \in O(g)$ if there exist constants $c > 0$ and $N \geq 0$ such that

$$\text{for every integer } n \geq N,\ f(n) \text{ and } g(n) \text{ are both defined, and } f(n) \leq c \cdot g(n).$$

Informally, this means that f can't grow asymptotically more quickly than g does. Note that it *does not* imply that f and g grow asymptotically at the same rate: As defined here, it is possible that $f \in O(g)$ even though f grows "strictly more slowly" than g (and, this may be different from the way you've seen the notation used before).

This effectively defines "$O(g)$" to be a *set* of asymptotically positive functions (and says when a given function f belongs to this set).

Examples

As the following examples indicate, the above definition can sometimes be used to prove that $f \in O(g)$ for asymptotically positive functions f and g. It can also be used to establish that $f \notin O(g)$ in some cases.

Example 6.7. If $f(n) = 2n$ and $g(n) = n$ then it suffices to set $c = 2$ and $N = 0$, and then observe that $f(n)$ and $g(n)$ are both defined and $f(n) \leq cg(n)$ whenever $n \geq N$ (indeed, $f(n) = cg(n)$ for this choice of c), to prove that $f \in O(g)$.

You could also set c to be any constant that is *greater than* 2, and you could set N to be any constant that is greater than 0, in order to prove this: There's no need to choose constants that are minimal — and choosing ones that are a bit larger than necessary sometimes makes it easier to prove what you need to.

All of the remaining examples involve total functions, just as this first one did. We won't always bother to mention the fact that $f(n)$ (or $g(n)$) is always defined for sufficiently large n in these cases — but you should understand that this is a necessary part of the above definition, and that it isn't always trivial to prove this.

As the next example indicates, there are sometimes "tradeoffs" between the choices of these two constants as well.

Example 6.8. If $f(n) = 2n + 10$ and $g(n) = n$ then you could establish that $f \in O(g)$ by setting c to be 12 and N to be 1 — for $2n + 10 \leq 2n + 10n = 12n$ whenever $n \geq 1$. On the other hand, you could set c to be 3 and N to be 10 — for $2n + 10 \leq 2n + n = 3n$ whenever $n \geq 10$. You could also set c to be 12 and N to be 10 — but you *couldn't* set c to be 3 and N to be 1 (write down the corresponding condition on n and observe that it's false in this last case). In this case, the larger you set c to be, the smaller (and closer to 1) you can set N to be.

It's also true, for each of the above examples, that $g \in O(f)$.

Example 6.9. To see that this isn't always the case, let $f(n) = n$ and $g(n) = n^2$. Note that you can establish that $f \in O(g)$ by choosing $c = 1$ and $N = 1$.

However, $g \notin O(f)$,

To prove this, suppose instead that $g \in O(f)$. Then there exist constants $c > 0$ and N such that $f(n)$ and $g(n)$ are both defined and $g(n) \leq cf(n)$, for every integer $n \geq N$.

In particular, this condition must be satisfied for some integer n that is greater than or equal to both N and $c + 1$. However, in this case (since c and n are both positive, and $c < n$)

$$cn < n^2 = g(n) \leq cf(n) = cn,$$

which is clearly impossible — for cn can't be *strictly* less than itself!

Since the only thing we assumed in order to establish this contradiction was that $g \in O(f)$, it follows that $g \notin O(f)$, as claimed above.

Thus, the fact that $f \in O(g)$ for some pair of asymptotically positive functions f and g *does not* imply that $g \in O(f)$ in every case.

Example 6.10. Next let $f(n) = 3 + \sin n$ and $g(n) = 1$. Then (since $2 \leq f(n) \leq 4$ for every integer n), you can choose constants $c = 4$ and $N = 0$ and then confirm that the condition given in the definition is satisfied with this choice of constants, in order to conclude that $f \in O(g)$.

You should be able to prove that $g \in O(f)$ in this case, too.

Example 6.11. Finally, suppose that $f(n) = n^2$ and

$$g(n) = \begin{cases} n & \text{if } n \text{ is odd,} \\ n^3 & \text{if } n \text{ is even.} \end{cases}$$

Then you can prove that $f \notin O(g)$ by assuming that $f \in O(g)$ and obtaining a contradiction, as in the third example above (and by considering the values of these functions at large odd inputs). You should also be able to prove that $g \notin O(f)$, by assuming that $g \in O(f)$ and obtaining a contradiction (by considering even inputs instead of odd).

Thus there are asymptotically positive functions that are "incomparable" as far as this "big-Oh" notation is concerned.

These examples will be considered again in the remaining sections of this chapter.

6.4.2 Limit Test

All of the above examples are simple ones, partly because it can be quite difficult to prove that $f \in O(g)$ by using the definition directly. Here's a "limit test" that is often easier to apply:

Theorem 6.12 (Limit Test for $O(f)$). *If f and g are asymptotically positive functions and*

$$\lim_{n \to +\infty} \frac{f(n)}{g(n)}$$

exists and is finite *(that is, not equal to $+\infty$), then $f \in O(g)$.*

Proof. Suppose, as in the statement of the theorem, that the given limit exists and is a constant, so that there is some constant L such that

$$\lim_{n \to +\infty} \frac{f(n)}{g(n)} = L.$$

Since f and g are both asymptotically positive, so that $f(n)$ and $g(n)$ are both defined and $f(n)/g(n) \geq 0$ for all sufficiently large integers n, it must be the case that $L \geq 0$.

It follows by Definition 3.1 (and using 1 as the value for the constant ϵ mentioned in the definition) that there is some real number M_1 such that

$$\text{if } n > M_1 \text{ then } \left| \frac{f(n)}{g(n)} - L \right| < 1. \tag{6.1}$$

It's also true that f and g are asymptotically positive, so that there exists a constant M_2 such that $f(n)$ and $g(n)$ are both defined and positive for every integer $n \geq M_2$.

Now let $c = L + 1$ and let $N = 1 + \max(M_1, M_2)$. Equation 6.1, above, applies that if n is any integer that is greater than or equal to N then (since $n > M_1$)

$$\frac{f(n)}{g(n)} - L < 1,$$

so that

$$\frac{f(n)}{g(n)} < L + 1 = c.$$

On the other hand, since $n \geq M_2$, $g(n) > 0$, so that the above inequality implies that

$$f(n) < cg(n)$$

for all such n as well — implying that $f \in O(g)$, as claimed. □

CPSC 413 students won't be expected to be able to reproduce the above proof, but they should understand it. They should also be able to *use* the above theorem in order to prove that $f \in O(g)$ by computing

$$\lim_{n \to +\infty} \frac{f(n)}{g(n)}$$

and observing that it is a constant (when this is the case!).

In order to do this, it's necessary to compute the limit as n approaches infinity of a ratio $f(n)/g(n)$ of functions f and g, where f and g both approach infinity. The techniques for computing limits given in Chapter 3 will be useful as you try to do this.

Note that the above theorem *does not* say that it's necessary for the above limit to exist whenever $f \in O(g)$, and we'll see, shortly, that it's possible that $f \in O(g)$ even though the above limit test "fails."

On the other hand, you can conclude that $f \notin O(g)$ if the limit test "fails" in one particular way:

Theorem 6.13 (Negative Limit Test for $O(f)$). *If f and g are asymptotically positive functions and*

$$\lim_{n \to +\infty} \frac{f(n)}{g(n)} = +\infty$$

then $f \notin O(g)$.

Proof. Suppose, instead that

$$\lim_{n \to +\infty} \frac{f(n)}{g(n)} = +\infty$$

and $f \in O(g)$ for some pair of asymptotically positive functions f and g.

Since f and g are asymptotically positive, there exists a constant M_1 such that $f(n)$ and $g(n)$ are both defined and positive for every integer $n \geq M_1$.

Since $f \in O(g)$ there exists constants $c > 0$ and N such that $f(n)$ and $g(n)$ are both defined and $f(n) \leq cg(n)$, for every integer $n \geq N$.

On the other hand, since

$$\lim_{n \to +\infty} \frac{f(n)}{g(n)} = +\infty,$$

there exists a constant M_2 such that

$$\frac{f(n)}{g(n)} > c$$

for every integer $n \geq M_2$ (see the continuation of Definition 3.1, taking "U" to be c).

The above three conditions imply that if n is any integer that is greater than $\max(M_1, N, M_2)$ then

$$c < \frac{f(n)}{g(n)} \leq c$$

(since $g(n) > 0$ in this case, so that it's possible to divide by $g(n)$ without changing the direction of an inequality).

This is clearly impossible, since c can't be strictly less than itself.

It follows that if

$$\lim_{n \to +\infty} \frac{f(n)}{g(n)} = +\infty$$

then $f \notin O(g)$, as claimed. □

Once again, CPSC 413 students won't be expected to be able to *prove* the above theorem, but they should be able to use it to prove that $f \notin O(g)$ by taking a limit and inspecting its value.

Examples

Consider the examples given above.

If $f(n) = 2n$ and $g(n) = n$ as in Example 6.7 on page 66, then *cancellation of a common factor* can be used to compute the limit you need (see Theorem 3.5 on page 45):

$$\lim_{n\to+\infty} \frac{f(n)}{g(n)} = \lim_{n\to+\infty} \frac{2n}{n} = \lim_{n\to+\infty} \frac{2}{1} = 2.$$

Here, the common factor being cancelled from the numerator and denominator is clearly n. Since the limit has been shown to be a constant, we can conclude that $f \in O(g)$.

Cancellation can also be used to compute the desired limit in the case that $f(n) = 2n + 10$ and $g(n) = n$ (as in Example 6.8):

$$\begin{aligned}
\lim_{n\to+\infty} \frac{f(n)}{g(n)} &= \lim_{n\to+\infty} \frac{2n+10}{n} \\
&= \lim_{n\to+\infty} \frac{(2+10/n)n}{1\cdot n} \\
&= \lim_{n\to+\infty} \frac{2+10/n}{1} \\
&= \lim_{n\to+\infty} 2+10/n \\
&= 2.
\end{aligned}$$

Once again, the limit is a constant, so we can conclude that $f \in O(g)$.

It might not have been obvious that cancellation could be used in the last example — you don't normally think of n as being a "factor" of a constant. On the other hand, since f and g are both *differentiable* functions, and both approach infinity as n does, *l'Hôpital's Rule* can also be used to compute the desired limit (see Theorem 3.13): That is, since

$$\lim_{n\to+\infty} f(n) = \lim_{n\to+\infty} g(n) = +\infty,$$

it's true by l'Hôpital's Rule that

$$\begin{aligned}
\lim_{n\to+\infty} \frac{f(n)}{g(n)} &= \lim_{n\to+\infty} \frac{f'(n)}{g'(n)} \\
&= \lim_{n\to+\infty} \frac{2}{1} \\
&= 2,
\end{aligned}$$

and, once again, this implies that $f \in O(g)$.

In the third example (Example 6.9: $f(n) = n$ and $g(n) = n^2$), it's also true that either cancellation or l'Hôpital's Rule can be used: Using cancellation (of the common factor n) we'd find that

$$\begin{aligned}
\lim_{n\to+\infty} \frac{f(n)}{g(n)} &= \lim_{n\to+\infty} \frac{n}{n^2} \\
&= \lim_{n\to+\infty} \frac{1}{n} \\
&= 0.
\end{aligned}$$

Using l'Hôpital's Rule instead, we'd first confirm that

$$\lim_{n\to+\infty} f(n) = \lim_{n\to+\infty} g(n) = +\infty$$

and then note that, since $f'(n) = 1$ and $g'(n) = 2n$,

$$\begin{aligned}\lim_{n\to+\infty} \frac{f(n)}{g(n)} &= \lim_{n\to+\infty} \frac{f'(n)}{g'(n)} \\ &= \lim_{n\to+\infty} \frac{1}{2n} \\ &= 0.\end{aligned}$$

In either case, since the limit has been shown to exist and to be equal to 0, we can conclude that $f \in O(g)$.

Once again, since

$$\lim_{n\to+\infty} g(n) = \lim_{n\to+\infty} f(n) = +\infty,$$

l'Hôpital's rule can be used to compute the limit we need to check to decide whether $g \in O(f)$:

$$\begin{aligned}\lim_{n\to+\infty} \frac{g(n)}{f(n)} &= \lim_{n\to+\infty} \frac{g'(n)}{f'(n)} \\ &= \lim_{n\to+\infty} \frac{2n}{1} \\ &= +\infty,\end{aligned}$$

so that the above "Negative Limit Test for $O(f)$" (Theorem 6.13) can be used to conclude that $g \notin O(f)$.

The next example (Example 6.10, in which $f(n) = 3 + \sin n$ and $g(n) = 1$) illustrates that it's sometimes necessary to use the definition (or something else that's different from the limit test): It's been shown above, using the definition directly, that $f \in O(g)$. However, this *can't* be proved using the limit test, because the necessary limit

$$\lim_{n\to+\infty} \frac{f(n)}{g(n)} = \lim_{n\to+\infty} 3 + \sin n$$

doesn't even exist! It "doesn't exist" because there are infinitely many of the values

$$3 + \sin 0, 3 + \sin 1, 3 + \sin 2, 3 + \sin 4, 3 + \sin 5, \ldots$$

that are close to 2 (because they correspond to integers n such that $\sin n$ is close to -1), but there are also infinitely many of the above values that are close to 4 (corresponding to integers n such that $\sin n$ is close to $+1$) — the ratio $f(n)/g(n)$ doesn't converge to anything at all.

The desired limit

$$\lim_{n\to+\infty} \frac{f(n)}{g(n)}$$

also doesn't exist in the final example, Example 6.11, in which $f(n) = n^2$ and in which $g(n) = n$ when n is odd and $g(n) = n^3$ when n is even. It was argued above that $f \notin O(g)$ and $g \notin O(f)$ for this example.

Perhaps the most important thing to remember about the last two examples is that they show that you *can't* conclude either that f is, or is *not*, in $O(g)$ when the limit tests fail — more precisely, you can't decide whether $f \in O(g)$ or not, if the limit tests both fail because the desired limit doesn't exist.

6.5 Big-Omega Notation

The next notation is probably *not* familiar to you; it's similar to "big Oh" notation, but it can be used to express asymptotic *lower* bounds (rather than upper bounds) for growth functions.

Informally, f is in big-Omega of g ("$f \in \Omega(g)$") if f grows asymptotically at least as *quickly* as g.

Definition 6.14. If f and g are asymptotically positive functions then $f \in \Omega(g)$ if there exist constants $c > 0$ and $N \geq 0$ such that

$$\text{for every integer } n \geq N,\ f(n) \text{ and } g(n) \text{ are both defined, and } f(n) \geq c \cdot g(n).$$

Note that the only difference between this and Definition 6.6 (for "$f \in O(g)$") is the direction of the inequality at the end.

Examples

You can sometimes prove that $f \in \Omega(g)$ by using the definition directly, just like you can when trying to prove that $f \in O(g)$.

Suppose, as in Example 6.8, that $f(n) = 2n + 10$ and $g(n) = n$. Then if you choose $c = 2$ and $N = 1$, then the condition given in the definition ("$f(n) \geq cg(n)$ whenever $n \geq N$") is satisfied, for this choice of c and N.

Note, by the way, that you definitely *couldn't* use the same pair of constants to try to prove that $f \in O(g)$ by using Definition 6.6, even though $f \in O(g)$ as well.

You should be able to use the above definition to establish that $f \in \Omega(g)$ for all but two of the pairs of functions f and g that have been used as examples in the previous section.

In particular, $f \notin \Omega(g)$ for the third example, Example 6.9, $f(n) = n$ and $g(n) = n^2$, and for Example 6.11, when $f(n) = n^2$ and $g(n) = n$ if n is odd and $g(n) = n^3$ if n is even. In these cases, you can establish these negative results by assuming that $f \in \Omega(g)$ and then deriving a contradiction (like the fact that a constant is strictly less than itself, as in the examples above).

On the other hand, $g(n) = n^2 \geq n = f(n)$ whenever $n \geq 1$, so you can choose $N = 1$ and $c = 1$ and establish that $g \in \Omega(f)$, for Example 6.9, by using the above definition.

Recall that it's already been established that $f \in O(g)$ but $g \notin O(f)$, for this pair of functions; what does this suggest to you about the relationship between the conditions "$f \in O(g)$" $f \in \Omega(g)$" that have been given so far?

6.5.1 Limit Test

Once again, there is a limit test that isn't always applicable, but that is generally easier to use than the formal definition, when it applies.

Theorem 6.15 (Limit Test for $\Omega(f)$). *If f and g are asymptotically positive functions and*

$$\lim_{n \to +\infty} \frac{f(n)}{g(n)}$$

exists and is either $+\infty$ or is a positive *constant (that is, strictly greater than 0), then $f \in \Omega(g)$.*

There's also a "negative" version that's useful when the limit is defined, but doesn't have the value you want:

Theorem 6.16 (Negative Limit Test for $\Omega(f)$). *If f and g are asymptotically positive functions and*

$$\lim_{n\to+\infty} \frac{f(n)}{g(n)} = 0$$

then $f \notin \Omega(g)$.

The proofs of these theorems resemble the proofs of Theorems 6.12 and 6.13, and use the definition of limit (Definition 3.1). Consider trying to prove these theorems, if you want to exercise your theorem proving skills or test your understanding of the definitions.

Examples

Once again, consider the five examples that have been used so far in this chapter.

If $f(n) = 2n$ and $g(n) = n$ (Example 6.7), then cancellation of a common factor can be used to compute the desired limit, as shown above, to establish that

$$\lim_{n\to+\infty} \frac{f(n)}{g(n)} = 2.$$

Since this is a *positive* constant, this implies that $f \in \Omega(g)$ in this case.

It's been shown above that if $f(n) = 2n + 10$ and $g(n) = n$ (as in Example 6.8) then

$$\lim_{n\to+\infty} \frac{f(n)}{g(n)} = 2$$

in this case as well, so $f \in \Omega(n)$ in this case, too.

It's been shown in Example 6.9 ($f(n) = n$ and $g(n) = n^2$) that

$$\lim_{n\to+\infty} \frac{f(n)}{g(n)} = 0.$$

In this case, the negative limit test can be used to conclude that $f \notin \Omega(g)$.

On the other hand, it's been observed already that

$$\lim_{n\to+\infty} \frac{g(n)}{f(n)} = +\infty$$

for this pair of functions — so that the limit test for $\Omega(f)$ can be used to establish that $g \in \Omega(f)$ in this case.

It's been noted already that if $f(n) = 3 + \sin n$ and $g(n) = 1$ as in Example 6.10 then the limit

$$\lim_{n\to+\infty} \frac{f(n)}{g(n)}$$

doesn't exist, so that both the limit test and negative limit test for $\Omega(f)$ "fail." On the other hand, since $2 \leq f(n) \leq 4$ for every integer n, $f(n) \geq 2g(n)$ for every nonnegative integer n, so you can use the definition for $\Omega(f)$, with constants $c = 2$ and $N = 0$, to prove directly that $f \in \Omega(g)$ in this case.

In the final example, Example 6.11, in which $f(n) = n^2$ and $g(n)$ is either n or n^3, depending on the parity of n, the desired limit doesn't exist and $f \notin \Omega(g)$.

Thus, the limit tests are "inconclusive" if the desired limit doesn't exist — that is, it's possible that f really is in $\Omega(g)$ in this case, even though the limit of the ratio $f(n)/g(n)$ doesn't exist, and therefore can't be used to prove it — but it's also possible that $f \notin \Omega(g)$, as well.

6.6 Big-Theta Notation

Informally, f is in big-Theta of g ("$f \in \Theta(g)$") if f grows asymptotically *at the same rate as* g.

Definition 6.17. If f and g are asymptotically positive functions then $f \in \Theta(g)$ if and only if $f \in O(g)$ and $f \in \Omega(g)$. This is the case if and only if there exist positive constants $c_L, c_U > 0$ and $N \geq 0$ such that

$$\text{for every integer } n \geq N,\ f(n) \text{ and } g(n) \text{ are both defined, and } c_L g(n) \leq f(n) \leq c_U g(n).$$

Note that the final version of the definition (the equivalent condition) is essentially what you get by combining the two definitions of "$f \in O(g)$" and "$f \in \Omega(g)$" (that is, combining all their requirements together).

Since $f \in \Theta(g)$ if and only if $f \in O(g)$ *and* $f \in \Omega(g)$, it should be clear that you can prove that $f \in \Theta(g)$ by proving that $f \in O(g)$ and then proving that $f \in \Omega(g)$ as well — and you can prove that $f \notin \Theta(g)$ either by proving that $f \notin O(g)$ or by proving that $f \notin \Theta(g)$.

6.6.1 Limit Test

Again, limit tests exist for $\Theta(f)$ and, again, the comments made about the limit tests that are given above apply here, too.

Theorem 6.18 (Limit Test for $\Theta(f)$). *If f and g are asymptotically positive functions and*

$$\lim_{n \to +\infty} \frac{f(n)}{g(n)}$$

exists and is a finite constant that is strictly greater than zero*, then $f \in \Theta(g)$.*

Note that the above condition on the limit is, essentially, a requirement that it passes both of the "limit tests" that have been given already for $O(f)$ and $\Omega(f)$.

Theorem 6.19 (Negative Limit Test for $\Theta(f)$). *If f and g are asymptotically positive functions and*

$$\textit{either} \quad \lim_{n \to +\infty} \frac{f(n)}{g(n)} = 0 \quad \textit{or} \quad \lim_{n \to +\infty} \frac{f(n)}{g(n)} = +\infty$$

then $f \notin \Theta(g)$.

The above condition on the limit is, essentially, that it satisfies one of the two conditions given in the "negative limit tests" that have been given so far.

In Examples 6.7 and 6.8, the limit test can be used to prove that $f \in \Theta(g)$. The negative limit test can be used to prove that $f \notin \Theta(g)$ in Example 6.9. The last two examples show that nothing can be concluded if the above limit doesn't exist — note that $f \in \Theta(g)$ in Example 6.10 while $f \notin \Theta(g)$ in Example 6.11.

6.7 Little-Oh Notation

Informally, f is in little-oh of g ("$f \in o(g)$") if f grows asymptotically *strictly more slowly* than g.

Definition 6.20. If f and g are asymptotically positive functions then $f \in o(g)$ if for every positive constant $c > 0$, there exists a constant $N \geq 0$, such that $f(n)$ and $g(n)$ are both defined and $f(n) \leq cg(n)$ for every integer $n \geq N$.

Note the difference in the way the constants c and N are used here: This is a condition that must hold *for every* choice of the positive multiplier c (no matter how small) — and N is allowed to depend on (that is, be a function of) the multiplier c that is being considered. In general, N gets larger as c gets smaller.

6.7.1 Limit Test

Theorem 6.21 (Limit Test for $o(f)$). *If f and g are asymptotically positive functions then $f \in o(g)$ if and only if*

$$\lim_{n \to +\infty} \frac{f(n)}{g(n)} = 0.$$

There is a major difference between this limit test and all the earlier ones: It's *necessary* for the limit to exist, and for it to be equal to zero, in this case, if $f \in o(g)$.

Since both the definition and the limit test involve asymptotically positive functions — meaning that you can rely on the fact that $f(n)/g(n)$ is defined, and positive, for sufficiently large n — this is actually a reasonably easy theorem to prove: In this case, the condition given above (for it to be true that $f \in o(g)$) is virtually identical to the condition that

$$\lim_{n \to +\infty} \frac{f(n)}{g(n)} = 0,$$

as given in Definition 3.1.

If f and g are as defined in Example 6.9 then either Definition 6.20 or the limit test given in Theorem 6.21 can be used to prove that $f \in o(g)$. If f and g are as in Examples 6.7 or 6.8 then $f \notin o(g)$ because the desired limit exists but has the wrong value. Finally, if f and g are as in Examples 6.10 or 6.11 then $f \notin o(g)$ because the desired limit doesn't exist at all.

6.7.2 Little-Omega Notation

Informally, f is in little-omega of g ("$f \in \omega(g)$") if f grows asymptotically *strictly more quickly* than g.

Definition 6.22. If f and g are asymptotically positive functions then $f \in \omega(g)$ if for every positive constant $c > 0$, there exists a constant $N \geq 0$, such that $f(n)$ and $g(n)$ are both defined and $f(n) \geq cg(n)$ for every integer $n \geq N$.

Note that the only difference between this and the formal definition of "$f \in o(g)$" is the direction of the inequality at the end.

6.7.3 Limit Test

Theorem 6.23 (Limit Test for $\omega(f)$). *If f and g are asymptotically positive functions then $f \in \omega(g)$ if and only if*

$$\lim_{n \to +\infty} \frac{f(n)}{g(n)} = +\infty.$$

Again, existence of the limit, and the fact that it is $+\infty$, is a necessary (as well as sufficient) condition here.

If f and g are as defined in Example 6.9 then $g \in \omega(f)$. The limit test can used to prove that $f \notin \omega(g)$ for each of Examples 6.7 – 6.11, because the desired limit either has the wrong value or doesn't exist at all in each case.

6.8 Some "Rules of Thumb"

Here are some relationships between functions that you should remember (and be able to use quickly):

Fact 6.24. A polylogarithmic function grows asymptotically more slowly than any fixed (positive) power of n. That is, if c is any constant and $\epsilon > 0$ is also any constant then

$$(\log n)^c \in o(n^\epsilon),$$

no matter how large c is, and no matter how small ϵ is (as long as ϵ is positive). This is true, no matter what (constant) base of the logarithm is used.

Fact 6.25. A polynomial of degree d_1 grows asymptotically more slowly than a polynomial of degree d_2 whenever $d_1 < d_2$. In particular, for all constants d_1 and d_2,

$$\text{if } d_1 < d_2 \text{ then } n^{d_1} \in o(n^{d_2}).$$

Fact 6.26. A polynomial grows asymptotically more slowly than an exponential function whose base is strictly greater than one. In particular, for every constant d and for every positive constant $\epsilon > 0$,

$$n^d \in o((1+\epsilon)^n),$$

no matter how large d is and no matter how small ϵ is (as long as ϵ is strictly greater than zero).

6.9 Application to the Analysis of Algorithms

In some cases you can draw conclusions about the relative behaviour of algorithms, for sufficiently large inputs, based on asymptotic relationships between the algorithms' growth functions.

For example, if two algorithms A_1 and A_2 solve the same problem, the worst case running time for A_1 is in $O(f_1)$, and the worst case running time for A_2 is in $\Omega(f_2)$, for asymptotically positive functions f_1 and f_2, and if $f_1 \in o(f_2)$, then algorithm A_1 will be faster than A_2, when run on at least some large inputs.

If A_1's worst case running time is in $O(f_1)$, A_2's worst case running time is in $\Omega(f_2)$, and $f_1 \in O(f_2)$, then A_1 might *not* ever be faster than A_2 — but there will exist some constant c such that the time used by A_1 is at most c times that used by A_2, on infinitely many large inputs.

You can't conclude *either* of these things, if you've only established that A_2's running time is in "$O(f_2)$" instead of in "$\Omega(f_2)$" (why?).

Obviously, this isn't all you'd want to be able to prove when comparing the performance of two algorithms — but it's a start. If you've established the first of the above two results, then you could perform a more careful analysis in order to discover *how large* the inputs must be in order

for A_1 to be faster than A_2. You might also try to discover whether the "worst case" is atypical, or whether it's close to the "average case" as well.

In the second case (you know that $f_1 \in O(f_2)$ and not that $f_1 \in o(f_2)$), you could perform a more careful analysis, in order to determine what the above constant c really is. The result isn't very interesting if c is large, and you can't really consider A_1 to be superior based on the above results unless $c < 1$.

These applications of asymptotic notation to algorithm analysis motivate the attention being paid to asymptotic notation in CPSC 413. This will be continued in the next two chapters, in which we'll see ways to compute worst case running times of algorithms from their source code or pseudocode. We'll also see a bit about how you can perform the "more careful analysis" that's mentioned above.

6.10 Exercises

1. Rank the following functions of n by order of asymptotic growth. That is, reorder them as $g_1, \ldots, g_{12}$ so that $g_i \in O(g_{i+1})$ for $1 \leq i \leq 11$. Then, say whether $g_i \in \Theta(g_{i+1})$, for $1 \leq i \leq 11$, as well. It is not necessary to give proofs for any of your claims.

(a) $\ln \ln n$	(e) $\left(\frac{3}{2}\right)^n$	(i) $\left(\frac{2}{3}\right)^n$
(b) $n2^n$	(f) e^n	(j) n^2
(c) $n!$	(g) $\ln n$	(k) $\sqrt{\log_2 n}$
(d) 1000000	(h) $2^{(1.01^n)}$	(l) $\log_2 n$

2. Prove that $n(\ln n)^2 \in O(n\sqrt{n})$.

3. Prove that $2n^3 + 3n^2 \log n \in \Theta(n^3)$.

4. Prove that $n^2 \ln n \in \Omega(n(\ln n)^2)$.

5. Using only the definition of "$O(f)$" (for an asymptotically positive function f), prove that $n^4 + 3n^2 + 8n + 100 \in O(n^4)$.

6. Prove or disprove each of the following claims.

   (a) For all asymptotically positive functions f and g, either $f \in O(g)$, $g \in O(f)$, or both.

   (b) For all asymptotically positive functions f and g, either $f \in O(g)$, $f \in \Omega(g)$, or both.

   (c) For all asymptotically positive functions f and g, if $f \in O(g)$ and $f \notin \Theta(g)$, then $f \in o(g)$.

   (d) For all asymptotically positive functions f and g, $f \in O(g)$ if and only if $g \in \Omega(f)$.

   (e) For all asymptotically positive functions f, g, and h, if $f \in O(g)$ and $g \in O(h)$, then $f \in O(h)$.

   (f) For all asymptotically positive functions f, g, and h, if $f \in O(h)$ and $g \in O(h)$, then $f + g \in O(h)$.

Hints for selected exercises are given in the next section; solutions for Exercises #2, 4, 5, 6(a), 6(c), 6(d), and 6(f) can be found in Section 11.1.1.

6.11 Hints for Selected Exercises

Exercise #1: You should be able to use the "rules of thumb" given in Section 6.8 to do quite a bit of the work needed to classify these functions — these should be enough to split the functions into three or four sets, such that every function in one set grows asymptotically faster than every function in another.

After that, you should try to use limit tests and (if necessary) the definitions of $O(f)$, et cetera, to complete the question. Be prepared to use cancellation and l'Hôpital's rule in combination (and possibly, more than once each) to evaluate the limits you obtain.

Note that pictures can be misleading: You should *only* use them to try to gain intuition, and you should use the rules of thumb, limit tests, and definitions from this chapter, instead of pictures, in order to prove things about the asymptotic rates of growth of functions.

Exercises #2–4: Choose an appropriate limit test and then apply it. Be prepared to use cancellation or l'Hôpital's rule to compute the limit you need.

Exercise #6: Some of these claims are true, and some are false.

Since they all describe properties of *all* pairs (or triples) of asymptotically positive functions, it's sufficient to give *one* pair (or triple) of such functions that don't have the stated property, in order to prove that a claim is false. Consider the five examples that were used in this chapter when looking for this kind of "counterexample."

On the other hand, you'll need to use the definitions that were given in this chapter in order to prove that some of these claims are true.

6.12 Sample Tests

The following tests were used in fall 1996 and 1997, respectively. Solutions for these tests can be found in Section 11.1.2.

6.12.1 Class Test for 1996

Instructions:

Attempt all questions. Write answers on the question sheets.

No aids allowed.

Duration: 50 minutes.

1. (5 marks — 1 mark for each part) For any pair of functions $f(n)$ and $g(n)$, exactly one of the following six claims is correct.

   (i) $f(n) \in \Theta(g(n))$.
   (ii) $f(n) \in o(g(n))$.
   (iii) $f(n) \in O(g(n))$, but $f(n) \notin \Theta(g(n))$ and $f(n) \notin o(g(n))$.
   (iv) $f(n) \in \omega(g(n))$.
   (v) $f(n) \in \Omega(g(n))$, but $f(n) \notin \Theta(g(n))$ and $f(n) \notin \omega(g(n))$.
   (vi) None of the above.

   For each of the following pairs of functions $f(n)$ and $g(n)$ say which one of these claims is correct. **You do not need to prove your answer.**

   (a) $f(n) = n^3$, $g(n) = 100n^2 \ln n$ **Answer:** _____

   (b) $f(n) = 2^n$, $g(n) = 3^n$ **Answer:** _____

   (c) $f(n) = n + (\ln n)^2$, $g(n) = n - \sqrt{n}$ **Answer:** _____

   (d) $f(n) = \begin{cases} n & \text{if } n \text{ is even,} \\ n^2 & \text{if } n \text{ is odd,} \end{cases}$ $g(n) = \frac{n^2}{2}$ **Answer:** _____

   (e) $f(n) = n^{1000}$, $g(n) = \frac{1.5^n}{n^2}$ **Answer:** _____

2. (5 marks) Prove that $n^2 \in \Omega(n \ln n)$.

3. (10 marks — 5 marks for each part) Say whether each of the following claims is true or false, and prove your answer.

   (a) For every function f such that $f(n)$ is defined and $f(n) > 0$ for all $n \in \mathbb{N}$, $f \in \Theta(f)$.

   (b) For all functions f and g such that $f(n)$ and $g(n)$ are defined for all $n \in \mathbb{N}$, and $f(n)$ and $g(n)$ are strictly greater than 0 whenever $n \geq n_0$ for some integer n_0 (so that f and g are asymptotically positive), if $f \in \Omega(g)$ then $f \in \omega(g)$.

6.12.2 Class Test for 1997

Instructions:

Attempt all questions. Write answers on the question sheets.

No aids allowed.

Duration: 50 minutes.

1. (6 marks — 1 mark for each part) For any pair of functions $f(n)$ and $g(n)$, exactly one of the following six claims is correct. (However, each claim might be true about zero, one, or several of the following pairs of functions!)

   (i) $f(n) \in \Theta(g(n))$.

   (ii) $f(n) \in o(g(n))$.

   (iii) $f(n) \in O(g(n))$, but $f(n) \notin \Theta(g(n))$ and $f(n) \notin o(g(n))$.

   (iv) $f(n) \in \omega(g(n))$.

   (v) $f(n) \in \Omega(g(n))$, but $f(n) \notin \Theta(g(n))$ and $f(n) \notin \omega(g(n))$.

   (vi) None of the above.

   For each of the following pairs of functions $f(n)$ and $g(n)$ say which one of these claims is correct. **You do not need to prove your answer.**

   (a) $f(n) = \frac{1}{10}n^2$, $g(n) = 20n$ **Answer:** _____

   (b) $f(n) = n^2 + 2n + 1$, $g(n) = n^2 - 2n + 1$ **Answer:** _____

   (c) $f(n) = \begin{cases} n & \text{if } n \text{ is odd,} \\ n^3 & \text{if } n \text{ is even,} \end{cases}$ $g(n) = n^2$ **Answer:** _____

   (d) $f(n) = n^4$, $g(n) = 2^n$ **Answer:** _____

   (e) $f(n) = n^2$, $g(n) = \begin{cases} n^2 + 2n + 1 & \text{if } n \text{ is odd,} \\ n \ln n & \text{if } n \text{ is even,} \end{cases}$ **Answer:** _____

   (f) $f(n) = \frac{1}{1000}(\ln n)^{1000}$, $g(n) = n^{1/1000}$ **Answer:** _____

2. (5 marks) Prove that $2n + 3 \ln n \in \Theta(n)$.

3. (6 marks) Say whether the following claim is true or false, and then prove your answer.

   Claim: For all asymptotically positive functions f and g, if $f \in o(g)$ then $g \in \omega(f)$.

4. Suppose now that you have two algorithms, A and B, that solve the same problem. The worst case running time of algorithm A (for input size n) is $T_A(n)$, and the worst case running time of algorithm B is $T_B(n)$. $T_A \in o(T_B)$.

   (a) (2 marks) Give the definition for the "worst case running time" of an algorithm.

   (b) (2 marks) Give the definition for "$f \in o(g)$," for asymptotically positive functions f and g.

   (c) (1 mark) Does the above statement about algorithms A and B imply that algorithm A will *always* solve the problem more quickly than algorithm B? That is, this does imply that algorithm A will solve the problem more quickly than algorithm B, *on every possible input*? Your answer should be either "Yes" or "No."

   (d) (3 marks) Say, in your own words, what the the above statement *does* imply about the relative performance of the two algorithms. Be as precise as you can (but try to do something different than just restating your answer for the first parts of this question).
   Note: You shouldn't need more than the rest of this page to give a complete answer for this question!

Chapter 7

Summations

7.1 Overview

This chapter (and the chapter that follows) includes techniques for producing bounds on the worst case running times of programs from their source code or pseudocode. This chapter concentrates on techniques for analyzing summations, which arise when analyzing programs with loops.

It is expected that CPSC 413 students will know the definitions that are presented here, that they will be able to write down summations that bound the worst case running times of simple (nonrecursive) programs with loops from source code or pseudocode (when given sufficient information about the cost of individual instructions), and that they will be able to apply the given techniques to find functions in "closed form" that are either equal to or that bound summations.

Some problems can only be solved if a specific technique is applied, and some techniques give better estimates than others — so students will also be expected to choose from the given techniques, based on the problem to be solved.

A set of exercises and sample tests is given at the end of this chapter, and can be used to assess the above skills.

Some of the techniques introduced in this chapter require mathematical induction, and one of them involves integration, so it might be helpful to review the material in Chapters 2 and 4 before attempting the exercises in this chapter.

7.2 A Motivating Example and a Cost Measure

Consider the following sorting program.

```
i := 1
while (i < n) do
        j := i
        while (j ≥ 1 and A[j] > A[j + 1]) do
                temp := A[j]
                A[j] := A[j + 1]
                A[j + 1] := temp
                j := j − 1
        end while
        i := i + 1
end while
```

This takes an array A of length n (with indices $1, 2, \ldots, n$) of integers as input and returns A, with its contents rearranged in nondecreasing order, as output. For now, we'll consider the length n of the input of the array to be the "size" of the input for this program, so the time function we'll try to derive will be expressed as a function of n.

7.2.1 Unit Cost Criterion

The above statement is, essentially, an "assumption" that the integers stored in the array each occupy unit space — or, perhaps, a "convention" that we'll charge unit space for any integer.

For now, we will also charge a single *step* (in time) for the execution of a single instruction written in a high level programming language (or in pseudocode).

Taken together, this set of assumptions is called the *Unit Cost Criterion*.

It should be clear that you *don't* get a very accurate measure of the time or space used by a program, if you perform an analysis that uses this convention, and the integers (or real number, etc.) used by your program can grow to be arbitrarily large, so that you require more than a fixed constant number of memory locations to store each.

On the other hand, this *can* be used to obtain asymptotic estimates of growth functions (with a multiplicative constant not determined), if the integers, reals, and so on, *do* all have constant size.

Use of the criterion also simplifies the job of analyzing algorithms.

Note that we've already briefly discussed two problems, and algorithms for them, in a setting where the unit cost criterion *was not* used: In Chapter 1 we considered two problems, *Integer Squaring* and *Integer Primality*, that had a single integer as input.

When we looked at those, we considered the length of a decimal representation of the input integer to be the "size" of the algorithms' input, and we counted the number of operations on digits (or, perhaps, the number of operations on integers of some other fixed size) in our analysis.

If we'd been using "the unit cost criterion" there, then we'd have considered the "input size" for all inputs for these problems to be "one," which would not have produced a very useful analysis.

Later on, when computational complexity is introduced, we'll be dropping the unit cost criterion again. However, we'll use it for now.

It will be true, by the way, that the *techniques* for algorithm analysis introduced in this chapter and the next one will still be applicable, without much change, after the unit cost criterion is abandoned. The analyses themselves will just be a bit more complicated when more realistic cost measures for single operations are used.

7.2.2 Analysis of the Example

The program given on the first page is a doubly nested **while** loop, with a few additional instructions.

In order to figure out how many steps can be used by the algorithm on an input of size n (that is, for an input array A of length n), we'll consider pieces of the program, deriving running time functions for each. In particular, we'll start with the inner loop body and then work outward until a time function for the entire program has been found.

The inner loop body consists of exactly four statements. According to the "unit cost criterion" defined here, we will charge 4 steps for an execution of this loop body.

Now consider the inner loop, itself. Suppose that on some execution of the loop, the loop body is executed exactly k times, and that $T_0(i)$ steps are performed for the i^{th} execution of the loop

body, for $1 \leq i \leq k$. Then — if we charge one for each execution of the loop's *test* (which isn't part of the loop body) as well, then the total number of steps executed by the loop, in this case, would be

$$1 + \sum_{i=1}^{k} (1 + T_0(i)) = k + 1 + \sum_{i=1}^{k} T_0(i).$$

Of course, we've argued already that "$T_0(i)$" is just 4, so the cost for the above execution of the loop would really be

$$k + 1 + \sum_{i=1}^{k} 4 = k + 1 + 4k = 5k + 1.$$

If you examine the loop itself, you will discover that the number of executions of the loop body (on some execution of the loop) depends on

- the value of the variable j, at the beginning of the execution of the loop, and
- the relative magnitudes of the contents of the array in positions $1, 2, \ldots, j+1$ at the beginning of the execution of the loop.

Since j is decremented every time the loop body is executed and the loop terminates unless the latest value of this variable is positive, the loop body can't be executed more than j times (where I'm referring, here, to the *initial* value for the variable with name "j").

The loop body might be executed fewer times than this: Indeed, it might not be executed at all (when $A[j+1]$ is smaller than $A[j]$). However, if $A[j+1]$ is larger than each of $A[1], A[2], \ldots, A[j]$ then it will be necessary to move $A[j+1]$ all the way to the front of the array, so that this loop body *can* sometimes be executed the maximal number, j, of times.

Thus the worst case running time of this inner loop is $5j + 1$.

The *outer* loop body contains the inner loop as well as two more statements — a first statement, which defines the value of j to be used when the inner loop is executed (it is set to i), and a final statement, which increments i. The number of steps used to execute this inner loop body in the worst case is therefore a function of i, namely

$$5i + 3.$$

Now, when the outer loop is executed, it is always executed with the initial value of i set to 1, and it only terminates when the value of i reaches n. Thus the outer loop body is always executed exactly $n - 1$ times, when the (initial) value of i is $1, 2, \ldots, n-1$, and (again, including steps for the execution of the loop's *test*), the number of steps used to execute the outer loop in the worst case is clearly *at most*

$$\begin{aligned}
1 + \sum_{i=1}^{n-1} (1 + (5i + 3)) &= 1 + \sum_{i=1}^{n-1} (5i + 4) \\
&= 1 + 5 \cdot \left(\sum_{i=1}^{n-1} i \right) + 4 \cdot \left(\sum_{i=1}^{n-1} 1 \right) \\
&= 1 + 5 \cdot \frac{n(n-1)}{2} + 4(n-1) \\
&= \frac{5}{2} n^2 + \frac{3}{2} n - 3.
\end{aligned}$$

Concluding this part of the analysis, we see that the entire program consists of only one initial statement and this outer loop, so that the number of steps used by the program in the worst case on an input of size n is clearly at most

$$\frac{5}{2}n^2 + \frac{3}{2}n - 2.$$

Note: It's stated that the worst running case is *at most* (rather than, exactly *equal to*) this value, because there's nothing in this analysis that proves that it's *possible* for the inner loop to use its worst case running time, every time it's executed. You'd need to prove that this can happen, if you wanted to state that the worst case running time of the outer loop was *exactly* the function of n given above.

In fact — *in this particular example* — the above function really *is* the worst case running time for the program, because *exactly* $\frac{5}{2}n^2 + \frac{3}{2}n - 2$ steps will be used when this program is executed, if the input array includes n distinct integers and is sorted in *decreasing* order instead of *increasing* order (so that the largest element is originally at the *front* of the array instead of the back, and so on). In this case, when the inner loop is used to try to move $A[j+1]$ into its correct location, it will *always* be true that $A[j+1]$ is smaller than each of $A[1], A[2], \ldots, A[j]$, so that it will be necessary to move $A[j+1]$ all the way to the front of the array.

When you're using the kind of analysis method given here (finding worst case running times for pieces of a program and adding these functions together), you can always safely conclude that the resulting function is an *upper bound* for the running time for the whole program, as we did here. You can't conclude immediately that it's also a *lower bound* for the worst case running time, so you need to describe an input, of the required size, for which the program really *does* use the running time you obtained in your analysis, if you want to conclude that your time function is exact.

It *is not* always the case that this kind of analysis technique gives results that are "tight;" sometimes, instead, one can actually manage to prove that parts of a program only exhibit their "worst case" behaviour very *infrequently*, so that the worst case running time for the entire program (which contains these subprograms) is actually much *smaller* than the above kind of analysis would suggest.

Proving this kind of thing is not easy, but it has been used successfully to show that some kinds of dynamic (or "self adjusting") data structures are, provably, much more efficient than this kind of analysis would lead you to think. This is definitely *beyond the scope of this course*, but you'll find some material about this in Chapter 18 ("Amortized Analysis") of Cormen, Leiserson, and Rivest [3], in case you're interested in this.

The topic, "Amortized Analysis," is sometimes included in CPSC 517.

Another Note: The above analysis is *far* more detailed than similar ones that you'll see in the rest of CPSC 413; we won't generally worry so much about the multiplicative constants, and will generally be content, instead, with an asymptotic result — in this case, the result that the worst running time is in $\Theta(n^2)$ would be sufficient.

This kind of asymptotic result ("$\Theta(n^2)$") is generally easier to obtain than the derivation of a more exact worst case running time function. Indeed, we won't always *be able to* obtain more than an asymptotic result, if we want to express the time function in a usable form (namely, in "closed form," as described below).

Final Note: As the above example should indicate, *summations* are obtained when you develop worst case running times for programs that include **while** (or, **repeat** or **for**) loops.

However, these functions aren't particularly useful — not readable, and hard to compare to other running times — if they're left in this form.

7.3 Expressions in Closed Form

Definition 7.1. An expression is in *closed form* if it contains only a *fixed* number of

- constants and variables,
- arithmetic expressions ($+$, $-$, $\times$, $\div$),
- exponentials and logarithms, and
- compositions of the above functions.

For example, polynomials and rational functions (fractions with polynomials as numerator and denominator) are in closed form, as are exponential functions, logarithms, and compositions like e^{2n^2+1}.

On the other hand, summations *aren't* generally in closed form, because they include more than a constant number of additions.

We'd like to express our running time functions as expressions in closed form, since this makes it easier to interpret them and to use them to compare algorithms.

Therefore, for the above example, the expression

$$\frac{5}{2}n^2 + \frac{3}{2}n - 2$$

is preferable to the summations used to derive it.

An asymptotic bound on the running time (especially, a tight one) — like the bound "$\Theta(n^2)$" for the above program — is useful too.

The rest of this chapter contains techniques that can be used to find expressions in closed form that are either equal to, or close bounds for, summations.

Techniques for finding expressions that are equal to summations will be presented first, and techniques for finding asymptotic bounds (which are frequently more general, or easier to apply, or both) will be presented after that.

7.4 Recognition of Special Cases

You should be able to recognize each of the following summations and you should know their closed form expressions immediately, once you've recognized them.

7.4.1 Arithmetic Series

Definition 7.2. An *arithmetic series* is any series of the form

$$\sum_{i=1}^{n}(a + bi)$$

for constants a and b (that is, for constants or functions a and b that don't depend on the index of summation, i).

Theorem 7.3 (Closed Form for an Arithmetic Series). *The arithmetic series*

$$\sum_{i=1}^{n}(a + bi)$$

has closed form

$$na + b\frac{n(n+1)}{2}.$$

That is,

$$\sum_{i=1}^{n}(a+bi) = na + b\frac{n(n+1)}{2}$$

for every integer $n \geq 0$.

Proof. The above identity will be proved by induction on n.

Basis: If $n = 0$ then

$$\sum_{i=1}^{n}(a+bi) = \sum_{i=1}^{0}(a+bi) = 0,$$

because there are no terms in the sum to be added together. On the other hand,

$$na + b\frac{n(n+1)}{2} = 0 \cdot a + b \cdot 0 = 0$$

as well, when $n = 0$, so that the identity is correct in this case.

Inductive Step: We now need to prove that if

$$\sum_{i=1}^{n}(a+bi) = na + b\frac{n(n+1)}{2}$$

then

$$\sum_{i=1}^{n+1}(a+bi) = (n+1)a + b\frac{(n+1)(n+2)}{2},$$

for every integer $n \geq 0$.

Suppose, therefore, that n is an integer such that $n \geq 0$ and

$$\sum_{i=1}^{n}(a+bi) = na + b\frac{n(n+1)}{2}.$$

Then (considering the final term of the sum separately)

$$\begin{aligned}
\sum_{i=1}^{n+1}(a+bi) &= \sum_{i=1}^{n}(a+bi) + (a+b(n+1)) \\
&= (n+1)a + b\frac{n(n+1)}{2} + a + b\frac{2(n+1)}{2} \\
&= (n+2)a + b\frac{(n+2)(n+1)}{2} \\
&= (n+2)a + b\frac{(n+1)(n+2)}{2},
\end{aligned}$$

as desired. Since the integer $n \geq 0$ was arbitrarily chosen it follows that if

$$\sum_{i=1}^{n}(a+bi) = na + \frac{n(n+1)}{2}$$

then

$$\sum_{i=1}^{n+1}(a+bi) = (n+1)a + \frac{(n+1)(n+2)}{2}$$

for every integer $n \geq 0$, as is required to complete the inductive step.

It follows by induction on n that the theorem is correct. □

CPSC 413 students certainly *should* be able to prove the above theorem, since it only requires a straightforward use of mathematical induction. In practice, though, this won't be necessary — it'll be sufficient to *recognize* that a given summation is an arithmetic series and then just to state and use the above closed form.

Recall, by the way, that summation is "linear." That is, for any constants c, d (or functions that don't depend on i) and functions f and g,

$$\sum_{i=1}^{n}(cf(i)+dg(i)) = c\left(\sum_{i=1}^{n} f(i)\right) + d\left(\sum_{i=1}^{n} g(i)\right),$$

so, in particular,

$$\sum_{i=1}^{n}(a+bi) = a\left(\sum_{i=1}^{n} 1\right) + b\left(\sum_{i=1}^{n} i\right),$$

and you should be able to *derive* the closed form for an arbitrary arithmetic series, as long as you can remember that

$$\sum_{i=1}^{n} 1 = n \qquad \text{and} \qquad \sum_{i=1}^{n} i = \frac{n(n+1)}{2}.$$

Finally, you should note that an arithmetic series was recognized and its closed form was used in the example given in Section 7.2.

7.4.2 Geometric Series

Definition 7.4. A *geometric series* is any series of the form

$$\sum_{i=1}^{n} ar^{i-1}$$

where a and r are constants (or functions that don't depend on i) and $r \neq 1$.

Theorem 7.5 (Closed Form for a Geometric Series). *The geometric series*

$$\sum_{i=1}^{n} ar^{i-1}$$

has the closed form

$$a\frac{r^n - 1}{r - 1}.$$

That is,

$$\sum_{i=1}^{n} ar^{i-1} = a\frac{r^n - 1}{r - 1}$$

for every integer $n \geq 0$.

Note that (using linearity) you should be able to derive this for any choice of a, as long as you can remember the closed form when $a = 1$,

$$\sum_{i=1}^{n} r^{i-1} = \frac{r^n - 1}{r - 1}.$$

Once again CPSC 413 students certainly *should* be able to prove the above theorem — it only requires mathematical induction — but this won't be necessary in practice. Instead, it's sufficient to recognize that a given series is a geometric series, and then state and apply the closed form for it without proving it.

7.4.3 Telescoping Sums

Definition 7.6. A *telescoping sum* is any summation of the form

$$\sum_{i=1}^{n}(f(i+1) - f(i)),$$

where f is a function of i.

Theorem 7.7 (Closed Form for a Telescoping Sum). *If the function* f *is given by an expression in closed form, then the telescoping sum*

$$\sum_{i=1}^{n}(f(i-1) - f(i))$$

has closed form

$$f(n+1) - f(1).$$

That is,

$$\sum_{i=1}^{n}(f(i+1) - f(i)) = f(n+1) - f(1)$$

for every integer $n \geq 0$.

Yet again, CPSC 413 students should be able to theorem is correct but will only be expected to apply it, without proof, in practice.

Sometimes telescoping sums will be "disguised" — you may (only) be able to express them in the above form by performing some algebraic manipulations on what you've been given.

Suppose, for example, that you want to find

$$\sum_{i=1}^{n} \frac{1}{i^2+i}.$$

At first glance it might not be clear what to do. However, it is possible to rewrite this sum in a form that allows us to use the above closed form to solve it:

$$\sum_{i=1}^{n} \frac{1}{i^2+i} = \sum_{i=1}^{n} (f(i+1) - f(i)), \qquad \text{for } f(i) = -\frac{1}{i}.$$

The above theorem can then be used to conclude immediately that

$$\sum_{i=1}^{n} \frac{1}{i^2+i} = f(n+1) - f(1) = -\frac{1}{n+1} + 1 = \frac{n}{n+1}.$$

7.5 Completing and Confirming a Pattern

Sometimes you suspect — or are told — that a summation has a closed form in some particular format, but there are unknown parameters to fill in as well.

In other words, you may be told, or suspect, that

$$\sum_{i=1}^{n} f(i) = g(c_1, c_2, \ldots, c_l, n),$$

for some particular function g and some unknown constants $c_1, c_2, \ldots, c_l$. Frequently the above expression is linear in $c_1, c_2, \ldots, c_l$ (and this is the easiest case to deal with).

In such a case, you can often make progress as follows.

1. Consider the values of the summation for small values of n ($n = 1, 2, \ldots, l$ will often be sufficient if there are l unknown constants) — that is, compute each of these values.

2. Use these values, along with the above general expression "$g(c_1, c_2, \ldots, c_l, n)$, to form a system of equations in the unknown values $c_1, c_2, \ldots, c_n$. If the above expression is linear in $c_1, c_2, \ldots, c_n$, then the resulting system will be a system of *linear* equations, so that you can use Gaussian Elimination to perform the next step:

3. Solve the system you just obtained for the unknowns, $c_1, c_2, \ldots, c_l$.

4. Plug the values you obtained for $c_1, c_2, \ldots, c_n$ into the original expression to produce a closed form for the given summation that's a function of n.

5. Use mathematical induction on n to confirm that the closed form you've derived really is correct.

Suppose, for example, that you forgot the formula for an arithmetic series, but you remembered that

$$\sum_{i=1}^{n} i$$

was a quadratic polynomial in n, that is,

$$\sum_{i=1}^{n} i = c_0 + c_1 n + c_2 n^2,$$

for three unknown constants, c_0, c_1, and c_2.

In order to apply the above method, you'd examine the sum for the cases $n = 1$, $n = 2$, and $n = 3$, and try to use this to solve for the unknown constants.

Since

$$\sum_{i=1}^{1} i = 1,$$

$$\sum_{i=1}^{2} i = 1 + 2 = 3,$$

and

$$\sum_{i=1}^{3} i = 1 + 2 + 3 = 6,$$

the system of equations you'd get is

$$\begin{aligned} c_0 + 1 \cdot c_1 + 1 \cdot c_2 &= 1 \\ c_0 + 2 \cdot c_1 + 4 \cdot c_2 &= 3 \\ c_0 + 3 \cdot c_1 + 9 \cdot c_2 &= 6. \end{aligned}$$

In matrix form, this is the system

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 4 \\ 1 & 3 & 9 \end{bmatrix} \cdot \begin{bmatrix} c_0 \\ c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 3 \\ 6 \end{bmatrix}.$$

This system can be solved using Gaussian Elimination, to discover that $c_0 = 0$ and $c_1 = c_2 = \frac{1}{2}$, suggesting that

$$\sum_{i=1}^{n} i = c_0 + c_1 n + c_2 n^2 = \frac{1}{2}(n^2 + n).$$

In order to complete this example, you should use mathematical induction on n to confirm that this formula really is correct, for every integer $n \geq 1$.

Here is a variation on this technique: Sometimes you might guess a closed form for a summation after checking its value for small values of n, without being told what form it takes ahead of time.

If your guess is correct, then you may be able to prove this by using induction on n.

For example, rather than writing

$$S(n) = \sum_{i=1}^{n} \frac{1}{i^2 + i}$$

as a telescoping sum (as above), you might evaluate this at small values, noting that

$$S(1) = \frac{1}{2}, \quad S(2) = \frac{2}{3}, \quad S(3) = \frac{3}{4}, \quad S(4) = \frac{4}{5}, \ldots$$

and "guess" at around this point that $S(n) = \frac{n}{n+1}$ for *every* integer $n \geq 1$.

Of course you wouldn't have proved it at this point (and you might even guess incorrectly), but, in this case (since the guess is correct) you could then prove that your answer is correct by using mathematical induction on n.

7.6 Applying Linear Operators

Note: This technique isn't usable very often and may be left out of lectures due to lack of time. If that is the case then no questions requiring it will be included in any tests this term.

Suppose you've established that

$$\sum_{i=1}^{n} f_i(x) = g_n(x) \tag{7.1}$$

for two families of functions of x, $f_1(x), f_2(x), \ldots$ and $g_1(x), g_2(x), \ldots$. Recall that a *linear operator* L (on functions) maps functions to functions in such a way that

$$L(c_1 h_1(x) + c_1 h_2(x)) = c_1 L(h_1(x)) + c_2 L(h_2(x))$$

for any functions $h_1(x)$ and $h_2(x)$ and for any constants c_1 and c_2.

Then you can apply the operator L to both sides of equation (7.1), above. Since the operator L is a linear operator and the expression on the left hand side of the equation is a sum of a *finite* number of terms (for any fixed n), you can correctly conclude that

$$\sum_{i=1}^{n} L(f_i(x)) = L\left(\sum_{i=1}^{n} f_i(x)\right) = L(g_n(x)).$$

There are two linear operators on functions that you might use employ here to solve new summations from old ones: differentiation and definite integration.

7.6.1 Differentiation

The operator

$$L(f(x)) = f'(x)$$

is a linear operator, and you can use this with any equation

$$\sum_{i=1}^{n} f_i(x) = g_n(x)$$

to conclude that

$$\sum_{i=1}^{n} f_i'(x) = g_n'(x)$$

as well.

Suppose, for example, that we start with the fact that

$$\sum_{i=1}^{n} x^{i-1} = \frac{x^n - 1}{x - 1};$$

this is the closed form for a geometric series (such that the "ratio" r mentioned in the definition for a geometric series happens to be the variable x).

This matches the above form, when $f_i(x) = x^{i-1}$ for $1 \leq i \leq n$ and

$$g_n(x) = \frac{x^n - 1}{x - 1}.$$

In this case, $f_i'(x) = (i-1)x^{i-2}$ for $1 \leq i \leq n$ and

$$g'(n)(x) = \frac{(x-1)(nx^{n-1}) - (x^n - 1)(1)}{(x-1)^2} = \frac{(n-1)x^n - nx^{n-1} + 1}{(x-1)^2}.$$

Therefore, we can apply this method to conclude that

$$\sum_{i=1}^{n}(i-1)x^{i-2} = \frac{(n-1)x^n - nx^{n-1} + 1}{(x-1)^2}.$$

If we multiply both sides of the above identity by x, we have that

$$\sum_{i=1}^{n}(i-1)x^{i-1} = \frac{(n-1)x^{n+1} - nx^n + x}{(x-1)^2}.$$

If we denote $i - 1$ by j then we have that

$$\sum_{j=0}^{n-1} jx^j = \frac{(n-1)x^{n+1} - nx^n + x}{(x-1)^2}.$$

Now, replace $n - 1$ by m:

$$\sum_{j=0}^{m} jx^j = \frac{mx^{m+2} - (m+1)x^{m+1} + x}{(x-1)^2}.$$

Since $0x^0 = 0$, so that the first term of the sum doesn't contribute anything to it, it's also true that

$$\sum_{j=1}^{m} jx^j = \frac{mx^{m+2} - (m+1)x^{m+1} + x}{(x-1)^2}.$$

Finally, let's replace j by i and m by n, just so that we're always using the same variables for the index of summation and the number of terms; then we have that

$$\sum_{i=1}^{n} ix^i = \frac{nx^{n+2} - (n+1)x^{n+1} + x}{(x-1)^2}.$$

If you'd like to check this derivation (just to be sure that no arithmetical errors have been made), then you can use induction on n to prove that this last identity is correct.

7.6.2 Definite Integration

The operator

$$L(f(x)) = \int_0^x f(t)\,dt$$

is also a linear operator, and you can use this with any equation

$$\sum_{i=1}^{n} f_i(x) = g_n(x)$$

to conclude that

$$\sum_{i=1}^{n} \left(\int_0^x f_i(t)\,dt \right) = \int_0^x g_n(t)\,dt$$

as well.

While the last statement is correct, it can sometimes be difficult to use this to find closed forms for summations: You need to be able to find closed form expressions for the *integral* given here on the right hand side, in order to make this work.

The remaining methods can be used to find asymptotic bounds for summations, rather than expressions that are equal to them. It'll often be necessary to use several of them in combination.

7.7 Bounding Terms

Suppose now that $n > m$, $l \le f(i) \le U$ for $m \le i \le n$, and we wish to estimate the sum

$$\sum_{i=m}^{n} f(i).$$

It's easy to see that

$$(n - m + 1) \cdot l \le \sum_{i=m}^{n} f(i) \le (n - m + 1) \cdot U,$$

because there are $n - m + 1$ terms in the sum and each has a value that's between l and U.

This holds, for example, if

$$l = \min_{m \le i \le n} f(i) \quad \text{and} \quad U = \max_{m \le i \le n} f(i).$$

Suppose, furthermore that f is a *nondecreasing* function, so that

$$\text{if } x \le y \text{ then } f(x) \le f(y);$$

then

$$\min_{m \le i \le n} f(i) = f(m) \quad \text{and} \quad \max_{m \le i \le n} f(i) = f(n),$$

so that the above inequality can be written in a simpler form (or, at least, one that involves no unknowns):

Theorem 7.8. *If $m < n$ and f is nondecreasing then*

$$(n - m + 1) \cdot f(m) \leq \sum_{i=m}^{n} f(i) \leq (n - m + 1) \cdot f(n).$$

For example, consider the sum

$$\sum_{i=1}^{n} \log_2 n.$$

Since the function $f(n) = \log_2 n$ is nondecreasing, the above theorem can be used. Since $\log_2 1 = 0$, it can be used to conclude that

$$0 \leq \sum_{i=1}^{n} \log_2 i \leq n \log_2 n.$$

7.8 Splitting the Sum

Suppose, once again, that we wish to estimate the sum

$$\sum_{i=1}^{n} f(i).$$

As the above example should suggest, the previous technique isn't generally enough to give (provably) tight asymptotic bounds for summations by itself. It often works better along with the observation that if m is any integer between 1 and n then

$$\sum_{i=1}^{n} f(i) = \sum_{i=1}^{m} f(i) + \sum_{i=m+1}^{n} f(i),$$

so that if

$$l_1 \leq \sum_{i=1}^{m} f(i) \leq U_1$$

and

$$l_2 \leq \sum_{i=m+1}^{n} f(i) \leq U_2$$

then

$$l_1 + l_2 \leq \sum_{i=1}^{n} f(i) \leq U_1 + U_2.$$

For example, consider the sum

$$\sum_{i=1}^{n} \log_2 i$$

once again, and let $m = \lfloor \frac{n}{2} \rfloor$. Consider again the *lower bounds* you'd obtain for pieces of this summation. Since $f(x) = \log_2 x$ is a nondecreasing function (at least, over the positive reals),

$$\sum_{i=1}^{m} \log_2 i \geq m \cdot \log_2 1 = 0,$$

which is no more helpful than before; however, the approach of "bounding terms" can also be used now to conclude that

$$\begin{aligned}
\sum_{i=m+1}^{n} \log i &\geq (n - (m+1) + 1) \log_2(m+1) \\
&= (n-m) \cdot \log_2(m+1) \\
&= (n - \lfloor \tfrac{n}{2} \rfloor) \cdot \log_2 (1 + \lfloor \tfrac{n}{2} \rfloor) \\
&\geq (\tfrac{n}{2}) \cdot \log_2 (\tfrac{n}{2}) \\
&= \tfrac{n}{2} \cdot (\log_2 n - \log 2) \\
&\geq \tfrac{1}{4} n \log_2 n
\end{aligned}$$

whenever $n \geq 4$. The above derivation used the fact that

$$n - \left\lfloor \frac{n}{2} \right\rfloor = \left\lceil \frac{n}{2} \right\rceil \geq \frac{n}{2}$$

as well as the fact that

$$\left\lfloor \frac{n}{2} \right\rfloor + 1 \geq \left\lceil \frac{n}{2} \right\rceil \geq \frac{n}{2}.$$

It follows that

$$\sum_{i=1}^{n} \log_2 i \geq \tfrac{1}{4} n \log_2 n \qquad \text{if } n \geq 4$$

as well. Combined with the *upper bound* obtained above, by bounding terms for the original sum, this implies that

$$\sum_{i=1}^{n} \log_2 i \in \Theta(n \log_2 n).$$

It often helps to choose m to be $\lfloor \frac{n}{2} \rfloor$, as above, if your function grows relatively "slowly" — if it's only polylogarithmic or even polynomial in its input, as above.

Choosing m to be $n - 1$ (so that you're effectively just splitting off the last term in the sum and considering it separately) works instead if the function grows more quickly — say, at least exponentially in its input.

7.9 Approximation of Sums by Integrals

Suppose, again, that f is a nondecreasing function that's defined on the real numbers between $i-1$ and i so that, in particular, $f(i-1) \leq f(x) \leq f(i)$ for any real number x such that $i - 1 \leq x \leq i$. This fact can be used to argue that

$$f(i-1) \leq \int_{i-1}^{i} f(t)\, dt \leq f(i);$$

changing perspective (and considering the case for $i+1$ as well),

$$\int_{i-1}^{i} f(t)\,dt \le f(i) \le \int_{i}^{i+1} f(t)\,dt,$$

if f is defined and nondecreasing on the set of real numbers between $i-1$ and $i+1$. Now, since

$$\int_{0}^{n} f(t)\,dt = \sum_{i=1}^{n} \int_{i-1}^{i} f(t)\,dt$$

and

$$\int_{1}^{n+1} f(t)\,dt = \sum_{i=1}^{n} \int_{i}^{i+1} f(t)\,dt,$$

this implies that if f is nondecreasing then

$$\int_{0}^{n} f(t)\,dt \le \sum_{i=1}^{n} f(i) \le \int_{1}^{n+1} f(t)\,dt,$$

if f is defined and nondecreasing on the set of real numbers between 0 and $n+1$.

The following more general result can be proved in the same way.

Theorem 7.9. *If m and n are integers such that $m \le n$, and f is a nondecreasing function defined on the set of real numbers between $m-1$ and $n+1$, then*

$$\int_{m-1}^{n} f(t)\,dt \le \sum_{i=m}^{n} f(i) \le \int_{m}^{n+1} f(t)\,dt.$$

Thus, if f is nondecreasing, then you can estimate the sum

$$\sum_{i=m}^{n} f(i)$$

by applying the above theorem and computing the definite integrals that it mentions.

This approach has the disadvantage of needing integration; it often has the *advantage* that it gives much *better* approximations than the last few methods that have been discussed.

For example, this method can be used to derive a much tighter approximation for the summation

$$\sum_{i=1}^{n} \log_2 i$$

than the one given above.

First note that the theorem can't be applied directly to estimate the sum, because the function $g(n) = \log_2 i$ *is not* defined on the set of real numbers between 0 and $n+1$ (it isn't defined at 0).

We'll avoid this problem by splitting off the first term: since $\log_2 1 = 0$,

$$\sum_{i=1}^{n} \log_2 i = \log_2 1 + \sum_{i=2}^{n} \log_2 i = \sum_{i=2}^{n} \log_2 i.$$

Now, since

$$f(n) = \log_2 n = \frac{\ln n}{\ln 2} = (\log_2 e) \ln n$$

is defined and is nondecreasing on the set of real numbers between 1 and $n+1$, we can conclude from the above theorem that

$$\int_1^n (\log_2 e) \ln t \, dt \leq \sum_{i=2}^{n} \log_2 i \leq \int_2^{n+1} (log_2 e) \ln t \, dt.$$

Now, since $\log_2 e$ is a constant, and $G(t) = t \ln t - t$ is an antiderivative for $g(t) = \ln t$,

$$\begin{aligned} \int_1^n (\log_2 e) \ln t \, dt &= (\log_2 e) \int_1^n \ln t \, dt \\ &= (\log_2 e)((n \ln n - n) - (1 \ln 1 - 1)) \\ &= (\log_2 e) n \ln n - (\log_2 e) n + \log_2 e, \end{aligned}$$

$$\begin{aligned} \int_2^{n+1} (\log_2 e) \ln t \, dt &= (\log_2 e) \int_2^{n+1} \ln t \, dt \\ &= (\log_2 e)((n+1) \ln(n+1) - (2 \ln 2 - 2)) \\ &\leq (\log_2 e)(n+1) \ln(n+1) + (\log_2 e) \\ &\leq (\log_2 e)(n+1) \ln(ne) \end{aligned}$$

whenever $n \geq 3$, and

$$\begin{aligned} (\log_2 e)(n+1) \ln(ne) &= (\log_2 e)(n+1)((\ln n) + 1) \\ &= (\log_2 e) n \ln n + (\log_2 e) n + (\log_2 e) \ln n + \log_2 e, \end{aligned}$$

implying that

$$(\log_2 e) n \ln n - (\log_2 e) n + \log_2 e \leq \sum_{i=1}^{n} \log_2 i \leq (\log_2 e) n \ln n + (\log_2 e) n + (\log_2 e) \ln n + \log_2 e$$

whenever $n \geq 3$. Subtracting $(\log_2 e) n \ln n$ everywhere, we see that

$$-(\log_2 e) n + \log_2 e \leq \sum_{i=1}^{n} \log_2 i - (\log_2 e) n \ln n \leq (\log_2 e) n + (\log_2 e) \log_2 n + \log_2 e,$$

whenever $n \geq 3$, which implies (since $(\log_2 e)(\ln n) = \log_2 n$) that

$$\left| \sum_{i=1}^{n} \log_2 i - n \log_2 n \right| \in O(n).$$

Since $(\log_2 e) n \ln n \in \omega(n)$, this means that we really have improved on the estimate for the sum that we'd obtained before, by approximating it so that the error in the approximation is sublinear in the value of the sum itself.

7.10 Completing and Confirming a Pattern, Continued

It's also sometimes possible to guess, and confirm, an *asymptotic bound* for a summation by using mathematical induction. In particular, to try to confirm that

$$\sum_{i=1}^{n} f(i) \in \Theta(g(n))$$

for functions f and g, one can generally proceed by developing two proofs by mathematical induction — one being that

$$\sum_{i=1}^{n} f(i) \leq cg(n)$$

for every integer $n \geq 1$, for some unknown positive constant c, which implies that the sum is in $O(g)$, and the other being that

$$\sum_{i=1}^{n} f(i) \geq \hat{c}g(n)$$

for every integer $n \geq 1$, and for some other unknown positive constant $\hat{c}$ which implies that the sum is in $\Omega(g)$.

In the process of trying to develop these proofs you generally discover conditions (frequently, inequalities) that the unknown constants c and $\hat{c}$ must satisfy. Once you've developed these proofs, you can frequently *choose* specific constants c and $\hat{c}$ that satisfy all these conditions. You can then observe that you can "plug these constants" into the proofs you've developed and that the proofs remain valid. Once you've done all that, you can conclude that

$$\sum_{i=1}^{n} f(i) \in \Theta(g(n)),$$

as desired.

Suppose, once again, that you want to use the fact that

$$\sum_{i=1}^{n} i \in \Theta(n^2)$$

but that you don't want (or need) to find an exact representation of the summation in closed form. According to the argument given above, you'd first try to prove by induction on n that

$$\sum_{i=1}^{n} i \leq cn^2$$

for every integer $n \geq 1$ and for some unknown positive constant c.

You would discover when establishing the basis (the case for $n = 1$) that this could only be true if $c \geq 1$.

You would probably also discover that the inductive step could be completed correctly, provided that

$$cn^2 + n + 1 \leq c(n+1)^2 = cn^2 + 2cn + c;$$

this is also satisfied if $c \geq 1$, as you could confirm by comparing the coefficients of 1, n, and n^2 in the leftmost and rightmost expressions shown here.

At the end of this you could therefore choose $c = 1$ (or, if you wanted to, $c = 2$, etc.), and then go back and confirm that the proof remains correct for your specific choice of c.

To derive a lower bound you'd repeat the process, trying to prove by induction on n that

$$\sum_{i=1}^{n} i \geq \hat{c}n^2$$

for every integer $n \geq 1$ and for some unknown constant $\hat{c}$.

In this case, you would probably discover that you need to assume that $\hat{c} \leq 1$ in order to complete the basis, but also that you want to assume that $\hat{c} \leq \frac{1}{2}$ in order to (easily) complete the inductive step.

Choosing $\hat{c} = \frac{1}{2}$ (or, if you want to, $\hat{c} = \frac{1}{3}$, etc.), you could go back over the proof and confirm that it remains correct for your choice of $\hat{c}$.

Having proved that

$$\frac{1}{2}n^2 \leq \sum_{i=1}^{n} i \leq n^2$$

for all $n \geq 1$, you can now conclude that

$$\sum_{i=1}^{n} i \in \Theta(n^2).$$

7.10.1 Avoid This Mistake!

This method that's just been described involves the development of one or two proofs of inequalities by mathematical induction that involve unknown constants. For example, if $T(n)$ is a function of n that is given as a summation and you wish to prove that $T(n) \in O(g(n))$ for some other function $g(n)$ (usually, in closed form), then you claim and prove by induction that

$$T(n) \leq cg(n)$$

for all (sufficiently large) n, for some "unknown" positive c.

In the process of developing this proof, you generally discover constraints on the constant c that must be satisfied if your proof is to be correct.

At the end of that — if all goes well — you should be able to choose a *specific* (and "known") value v for the constant c. If there was time for it, you could go back and replace the unknown constant c by the value you've selected, in your original proof, and you could then confirm that (you'd proved that) $T(n) \leq vg(n)$ for all sufficiently large n, for the specific value v that you chose at the end of this process.

Similarly, if you want to prove that $T(n) \in \Omega(g(n))$ then you can claim and prove by induction that

$$T(n) \geq \hat{c}g(n)$$

for all (sufficiently large) n, for some "unknown" positive constant $\hat{c}$. The process that you follow after completing this proof is the same as the process you follow after completing the proof that involved the constant c, when you were trying to establish that $T(n) \in O(g(n))$.

When you develop the proofs involving "unknown constants" that are described above you **must** treat the unknown values as **constants** — things that (each) have **one fixed** value that is unchanged, throughout the proof. Otherwise, you will not be able to *choose* one value for the unknown, at the end of the process, and you won't be establish the conclusion (namely, the asymptotic bound) that you wanted to.

Unfortunately, one reasonably common **error** that students make in this process is to treat the unknown whose something whose value can "shift" or "change" throughout the proof.

Here is an example of a **false claim** and a **fallacious proof** of the claim. The **only** technical error that is made in this proof is the error that has just been described.

False Claim: If

$$T(n) = \sum_{i=1}^{n} 1$$

then $T(n) \in O(1)$.

Fallacious Proof: It will be proved by induction on n that $T(n) \leq c$ for some unknown constant c, for every integer $n \geq 0$.

Basis: If $n = 0$ then $T(n) = 0$, so $T(n) \leq c$ in this case **provided that** $c \geq 0$.

Inductive Step: Suppose that $n \geq 0$ and that $T(n) \leq c$.

Then

$$\begin{aligned}
T(n+1) &= \sum_{i=1}^{n+1} 1 && \text{by the definition of } T \\
&= \left(\sum_{i=1}^{n} 1\right) + 1 && \text{splitting the sum} \\
&= T(n) + 1 && \text{since } T(n) = \sum_{i=1}^{n} 1 \\
&\leq c + 1 && \text{by the inductive hypothesis} \\
&\leq \hat{c} && \text{for } \hat{c} = c + 1.
\end{aligned}$$

Therefore $T(n+1)$ is less than or equal to a constant if $T(n)$ is. It follows by induction on n that $T(n) \in O(1)$. □

Now note that $T(n) = n$, so that $T(n)$ is not in $O(1)$ at all.

The **problem** with this proof, again, is that one constant replaced another in the middle of the inductive step: Since you wanted to show that $T(n) \leq c$ for every integer n and for a **fixed** constant c, you needed to assume that

$$T(n) \leq c$$

in the inductive step, and use this to prove that

$$T(n+1) \leq c$$

for the **same** constant c — and not just that

$$T(n+1) \leq \hat{c},$$

for a different constant $\hat{c}$ — because the fact that $T(n+1) \leq \hat{c}$ *does not* imply that $T(n+1) \leq c$, as well.

Let's think about what's really been "proved," above: If you take the proof, and think about what it implies about the values $T(0)$, $T(1)$, $T(2)$, and so on, then you should discover that it establishes, only, that

1. $T(0) \leq c_0$, for some value $c_0 \geq 0$;

2. $T(1) \leq c_1$, for some value $c_1 \geq 1$;

3. $T(2) \leq c_2$, for some value $c_2 \geq 2$;

4. $T(3) \leq c_3$, for some value $c_3 \geq 3$;

and, in general, that "$T(n) \leq c_n$, for some value $c_n \geq n$."

There's no single constant c that satisfies all the conditions stated for $c_0, c_1, c_2, \ldots, c_n, \ldots$ so you certainly *haven't* proved that "$T(n) \in O(1)$," by giving the argument that's listed above — because (once again) you *haven't* established that there exists a positive **constant** c such that $T(n) \leq c$ for every nonnegative integer n.

This mistake is made on tests and exams in CPSC 413 on a regular basis. Since it suggests a fundamental misunderstanding of the method (or of mathematical induction, or both), a heavy penalty is imposed whenever this error is found.

7.11 Analysis of Nonrecursive Algorithms

If you can find (tight) asymptotic bounds for summations that are in closed form, then you can find asymptotic bounds for worst case running times for loops, as argued at the beginning of these notes. As argued there, these bounds won't always be asymptotically tight — you'll need to confirm that your hypothetical "worst case" really can happen by describing an input of the given size that achieves it, if your code is at all complicated — but at least you can do more than you may have been able to do before.

Conditional statements, like **if-then** statements, **if-then-else** statements, or **case** statements, are at least as "easy" to handle, since the worst case running time for any such statement is just the maximum of the worst case running times for each of the possible cases (ways to flow through the code) that can arise.

Thus, in principle, you now know how to derive an asymptotic bound (in closed form, but not necessarily "tight") for any program that has no subroutines: Start from the innermost loop and work outward, analyze all cases in a case statement, and so on.

You can do the same for *nonrecursive* programs as well; start with subroutines that don't call any others and apply the technique described above. At this point you'll know how much to "charge" for a *call* to any of these subroutines made elsewhere in your code. This will allow you to derive worst case running times for other subroutines in your program. Continuing to do this, you'll eventually get to, and produce a worst case running time for, your main program.

Recursive programs can't be handled in quite this way; they'll be the motivation for the next "algorithm analysis" topic.

7.12 Exercises

1. Find expressions in closed form that are equal to each of the following summations.

   (a) $\sum_{i=1}^{n}(4i+5)$

   (b) $\sum_{i=1}^{n} 5^i$

   (c) $\sum_{i=1}^{n} \sin(i+1) - \sin i$

   (d) $\sum_{i=1}^{n} \frac{i}{(2i+1)^2(2i-1)^2}$

2. Find a closed form for

   $$\sum_{i=1}^{n} i^2$$

   and prove that it is correct, starting only with the information that it is a polynomial function of n with degree three.

3. Find asymptotically tight bounds in closed form for each of the following summations. That is, for each function $\sum_{i=1}^{n} f(i)$, try to find an expression $g(n)$ in closed form such that

   $$\sum_{i=1}^{n} f(i) \in \Theta(g(n)).$$

   As time permits, you should try to apply several (or even all) of the techniques that were introduced in class in order to solve these problems, in order to gain practice using these techniques, and you should compare the answers that you get using them. Note, though, that some of the techniques might not be applicable in all cases, and others might be difficult to apply.

   (a) $\sum_{i=1}^{n} i^3$

   (b) $\sum_{i=1}^{n} i^4 \ln i$

   (c) $\sum_{i=1}^{n} \frac{1}{i}$

   (d) $\sum_{i=1}^{n} \frac{1}{i^2}$

   (e) $\sum_{i=1}^{n} (i!) \ln i$

4. Prove that

$$\sum_{i=1}^{n} i^2 \in \Theta(n^3)$$

without solving the summation exactly. That is, try to complete the pattern suggested above in order to prove an *asymptotic bound*, rather than doing Question 2 all over again.

5. It was shown in this chapter that you could approximate a summation of a *nondecreasing* function by integrals. Find (and prove correct) a similar approximation of a *nonincreasing* function instead.

   Then, try to apply this (where appropriate) to improve your answers for Question 3, above.

Hints for selected exercises are given in the next section; solutions for Exercises #1(d), 3(b) and 4 can be found in Section 11.2.1.

7.13 Hints for Selected Exercises

Exercise #1(d): Start by trying to find some constant c such that

$$\frac{i}{(2i+1)^2(2i-1)^2} = \frac{c}{(2i+1)^2} - \frac{c}{(2i-1)^2};$$

it should be easy to find a closed form for the summation after that.

Exercise #3(c): Observe that the sum of the *last* half of the terms of this sum is in $\Theta(1)$, and try to use this to prove (using the strong form of mathematical induction) that the sum is in $\Theta(\log_2 n)$.

If you'd like try a different approach then you can observe that

$$\sum_{i=1}^{n} \frac{1}{i} = \sum_{j=0}^{n-1} \frac{1}{n-j},$$

because the second sum just lists the same terms as the first, in reverse order. Note that the function

$$f(x) = \frac{1}{n-x}$$

is defined and nondecreasing on the set of real numbers x between 0 and $n-1$. You should be able to use approximation by integrals to estimate the sum, given this information.

Exercise #3(d): Note that if $i \geq 1$ then

$$\frac{1}{i^2+i} \leq \frac{1}{i} \leq \frac{2}{i^2+i},$$

and that

$$\frac{1}{i^2+i} = \frac{1}{i} - \frac{1}{i+1}.$$

Now, use the second observation to prove that

$$\sum_{i=1}^{n} \frac{1}{i^2+i} \in \Theta(1),$$

and then use the first observation to conclude that

$$\sum_{i=1}^{n} \frac{1}{i^2} \in \Theta(1)$$

as well.

Alternatively, you can reverse the order of the terms in the given summation (as in the second approach suggested above for Exercise #3(c)) to make the sum look like a sum of a nondecreasing function, and then use approximation by an integral to finish.

Exercise #5: This can be answered by reading, and making small changes to, the derivation of the approximation of the sum of a nondecreasing function by an integral that's given in this chapter.

7.14 Sample Tests

The following tests were used in fall 1996 and 1997, respectively. Solutions for these tests can be found in Section 11.2.2.

7.14.1 Class Test for 1996

Instructions:

Attempt all questions. Write answers on the question sheets.

No aids allowed.

Duration: 50 minutes.

1. (10 marks) Find a function $f(x)$ (in closed form) such that $\sum_{i=1}^{n} i^4 - f(n) \in O(n^4)$, and prove that your answer is correct.

   Note: You may use the fact that the function $g(x) = x^4$ is an increasing function, without proving it.

2. (10 marks) Prove that $\sum_{i=1}^{n} i \log_2 i \in \Theta(n^2 \log_2 n)$.

   Hint: $\log_2(n/2) = (\log_2 n) - 1 \geq \frac{1}{2} \log_2 n$ for sufficiently large n (in particular, for $n \geq 4$).

7.14.2 Class Test for 1997

Instructions:

Attempt all questions. Write answers on the question sheets.

No aids allowed.

Duration: 50 minutes

1. (6 marks in total; marks for each part shown below)

   Each of the following summations is in a special form (for example, possibly, an arithmetic series) that you should be able to recognize, and whose closed form you should already know.

   Say way special form each summation has, and write down the *exact* solution of the above summation in closed form (*not* just the general solution for summations of this type). It isn't necessary to simplify your answer.

   (a) $\sum_{i=0}^{n-1} 3 \cdot 6^i$

   (1 mark:) This is a _____

   (2 marks:) Solution in closed form: _____

   (b) $\sum_{i=1}^{n} (5^{i+1} \ln(i+1) - 5^i \ln i)$

   (1 mark:) This is a _____

   (2 marks:) Solution in closed form: _____

2. (1 mark) *Briefly* explain why you might need to find asymptotic bounds for summations in closed form, when estimating the worst case running times of programs from source code.

3. Suppose $f(x)$ is a nondecreasing function of x.

   (a) (3 marks) State the approximation of $\sum_{i=1}^{n} f(i)$ by integrals defined in class. That is, state the integrals that are known to be upper and lower bounds for this sum, and say which is which!

   (b) (5 marks) Now, use approximation by integrals to prove that

   $$\sum_{i=1}^{n} i^{2.5} \in \Theta(n^{3.5}).$$

Note: You may assume that the function $f(x) = x^{2.5}$ is a nondecreasing function of x (defined on the nonnegative reals) without having to prove it.

4. (10 marks) Prove that

$$\sum_{i=1}^{n} i(\ln i)^3 \in \Theta(n^2(\ln n)^3).$$

There are several ways that you could prove this using techniques introduced in class; each will be acceptable (provided that the methods are used correctly).

If you get stuck, you can say *what* you'd need to do in order to finish (even though you couldn't manage to do it) in order to earn part marks.

Note: You may assume that the function $f(x) = x(\ln x)^3$ is a nondecreasing function of x (defined on the positive reals) without proving it.

Chapter 8

Recurrences

8.1 Overview

This chapter is a continuation of the previous one, in the sense that it includes techniques for producing bounds on the worst case running times of programs from their source code or pseudocode. The previous chapter included techniques for analyzing nonrecursive programs; this one introduces techniques for the analysis of recursive programs as well.

The chapter includes an introduction to "recurrences," which often arise when you try to write down the worst case running time of a recursive program, but which aren't in closed form, and it presents several techniques for producing functions in closed form that are either equal to or asymptotic bounds for functions that are given by recurrences. Some general techniques, that aren't always very easy to apply, are presented first. These include a "substitution method" and an "iteration method," as well as several techniques for simplifying recurrences. More specific techniques, that aren't always applicable at all but that are frequently easier to use when they *are* applicable, are presented after that.

As usual, CPSC 413 students will be expected to know the definitions that are presented here. They will also expected to be able to write down expressions — which may involve both summations and recurrences — for the worst case running times of programs, when given the programs' source code or pseudocode. They will also be expected to recognize *which* of the techniques (being introduced here) are applicable to a given recurrence, and to be able to *apply* these techniques in order to write down functions in closed form that are equal to or that bound these expressions.

A set of exercises and sample tests is given at the end of this chapter, and can be used to assess the above skills.

Some of the techniques involved in this chapter require mathematical induction, so it might be helpful to review the material in Chapter 2 before attempting the exercises in this chapter, if you haven't already done so.

8.2 Motivating Examples

8.2.1 Example: Fibonacci Numbers

Consider the following recursive function; this takes an integer n as input and returns the n^{th} Fibonacci number, F_n, as output (returning 0 if n is negative).

```
integer function fib (n: integer)

if (n ≤ 0) then
   return 0
elsif (n = 1) then
   return 1
else
   return fib(n − 1) + fib(n − 2)
end if

end function
```

For the moment, let's define $T(n)$ to be the number of steps executed by this function on *the input* n (we won't worry about the time used by this a program, defined as a function of *input size,* just yet).

Then (using the "unit cost" criterion, as defined in the previous chapter), it seems that

$$T(n) = 2 \qquad \text{if } n \leq 0,$$

and

$$T(1) = 3,$$

provided that we charge one for each test that must be executed, as well as one more step when a value is returned.

If $n \geq 2$, then two tests are executed and fail; the function is then recursively applied — twice — and then the results are added together and returned. If we charge one step for the final addition and return of the result, then can conclude that

$$T(n) = T(n-1) + T(n-2) + 3 \qquad \text{if } n \geq 2.$$

8.2.2 Another Example: Binary Search

Here is another recursive function. Think of the input parameters as being

- a sorted array A of integers of length n (with indices $1, 2, \ldots, n$)
- indices i and j, *usually* such that $1 \leq i \leq j \leq n$, but possibly also with $j < i$
- an integer *key* that you're looking for, inside a portion of the array A (namely, the piece in between positions i and j)

and think of the output as being 0 if *key* is not in any of the positions $i, i+1, \ldots, j$, and as being an integer h such that $i \leq h \leq j$ and $A[h] = key$, otherwise.

```
integer function BSearch(A, i, j, key)

if (i > j) then
   return 0
else
   mid := ⌊(i+j)/2⌋
   if (A[mid] = key) then
      return mid
   elsif (A[mid] > key) then
      return BSearch(A, i, mid − 1, key)
   else
      return BSearch(A, mid + 1, j, key)
   end if
end if

end function
```

For this case, let's try to define a function $T(m)$ in terms of an integer m, where $m = j - i + 1$. That is, m is the length of the subarray that's being examined by the above function (put another way, m is the number of possible locations within the array in which *key* might be located if it's found).

If $m \leq 0$ then, necessarily, $i > j$, so that the function terminates after executing the first test — which succeeds — and after returning the constant 0. Therefore,

$$T(m) = 2 \qquad \text{if } m \leq 0.$$

Otherwise, *mid* is defined to be the "middle" position within the subarray being searched. Since $m = j - i + 1 > 0$,

$$(mid - 1) - i + 1 = \left\lfloor \frac{i+j}{2} \right\rfloor - i = \left\lfloor \frac{j-i}{2} \right\rfloor = \left\lfloor \frac{m-1}{2} \right\rfloor$$

and

$$j - (mid + 1) + 1 = j - \left\lfloor \frac{i+j}{2} \right\rfloor = \left\lceil \frac{j-i}{2} \right\rceil = \left\lceil \frac{m-1}{2} \right\rceil.$$

Assuming (correctly!) that $T(m)$ is a nondecreasing function of m, the "worst case" occurs when $A[mid] < key$, so that all the tests executed at the top level of the code fail and the right subarray, with length $\left\lceil \frac{m-1}{2} \right\rceil$, must be searched. This implies that

$$T(m) \leq 5 + T\left(\left\lceil \frac{m-1}{2} \right\rceil\right) \qquad \text{if } m \geq 1.$$

We can't conclude yet that $T(m)$ is always *equal* to the expression shown on the right hand side, above, because we haven't argued that it's always possible to design an input including an array of size m, with a right subarray of size $\lceil \frac{m-1}{2} \rceil$, such that the worst case number $T(\lceil \frac{m-1}{2} \rceil)$ of operations *will* be used to complete the operation of the algorithm on this right subarray (it's at least plausible that the algorithm will always be faster than that, at this point during the computation).

However, it *is* the case that this maximum number of operations can be used at this point. In particular, it's possible to prove by induction on m that this maximum number of operations really will be used, as long as *key* is strictly greater than every element stored in the array. Therefore,

$$T(m) = 5 + T(\lceil \tfrac{m-1}{2} \rceil) \qquad \text{if } m \geq 1.$$

8.3 Recurrences

Definition 8.1. A *recurrence* is an equation or inequality that describes a function in terms of its values on smaller inputs.

An example of a recurrence is the description of the running time function "$T(n)$" for the function *fib*, on input n, that we just derived:

$$T(n) = \begin{cases} 2 & \text{if } n \leq 0, \\ 3 & \text{if } n = 1, \\ T(n-1) + T(n-2) + 3 & \text{if } n \geq 2. \end{cases} \tag{8.1}$$

Another example is the recurrence derived for the number of steps executed by *BSearch*, defined as a function of m:

$$T(m) = \begin{cases} 2 & \text{if } m \leq 0, \\ T(\lceil \frac{m-1}{2} \rceil) + 5 & \text{if } m \geq 1. \end{cases} \tag{8.2}$$

Here are two more examples of recurrences; they both describe the function $T(n)$ that was defined for *fib*, though neither specifies it exactly:

$$T(n) \leq \begin{cases} 3 & \text{if } n \leq 0, \\ 4 & \text{if } n = 1, \\ T(n-1) + T(n-2) + 3 & \text{if } n \geq 2; \end{cases} \tag{8.3}$$

$$T(n) \geq \begin{cases} 0 & \text{if } n \leq 0, \\ 1 & \text{if } n = 1, \\ T(n-1) + T(n-2) & \text{if } n \geq 2. \end{cases} \tag{8.4}$$

The following *isn't* a recurrence for $T(n)$, even though T satisfies this definition. It isn't a recurrence because the final part of this expresses $T(n)$ in terms of *larger* as well as smaller values, and not in terms of smaller values alone:

$$T(n) = \begin{cases} 2 & \text{if } n \leq 0, \\ 3 & \text{if } n = 1, \\ T(n+1) - T(n-1) & \text{if } n \geq 2. \end{cases}$$

It *also* fails to define $T(n)$ uniquely; you'd add the condition that the last equation is also satisfied when $n = 1$ (making it possibly look like you'd "defined" $T(2)$ *twice*) in order to achieve this. However, we won't be worrying about the properties of "non-recurrences" like this one.

Unfortunately, it isn't always easy to express the worst case running time for a recursive algorithm as a recurrence. However, if the recursive algorithm is reasonably well behaved — if the *size* of the input is decreased whenever the algorithm calls itself recursively, or some *other* function of the input is decreased whenever this happens (and the algorithm can be proved to terminate whenever the value of this other function is small) — then one *can* state the worst case running

time for the algorithm as a recurrence, as we did here for the two examples *fib* and *BSearch*. We'll only look at these "well behaved" kinds of recursive algorithms in CPSC 413.

As before, we'd like to find time functions that are given as expressions *in closed form*, and recurrences aren't in this form. We'll therefore consider methods to find exact solutions as well as asymptotic bounds for recurrences that are in closed form.

8.4 Methods for Solving Recurrences: The Substitution Method

8.4.1 Outline of the Method

The "substitution method" is generally used to confirm that conjectured *asymptotic bounds* for recurrences are correct. It's the same as the method that was called "completing and confirming a pattern" in the previous chapter, except that it will be applied here to recurrences instead of summations.

In order to prove that

$$T(n) \in O(g(n))$$

where $T(n)$ is defined by a recurrence, and $g(n)$ is a function in closed form (which has also been given to you, or which you've guessed), you should try to prove that

$$T(n) \leq cg(n) \text{ for every integer } n \geq 0, \text{ for some positive constant } c.$$

In order to prove that

$$T(n) \in \Omega(g(n)),$$

you should try to prove that

$$T(n) \geq \hat{c}g(n) \text{ for every integer } n \geq 0, \text{ for some (other!) positive constant } \hat{c}.$$

As was the case for the previous method, you'll try to use mathematical induction (on n) to generate each proof.

In the course of doing this, you'll identify a number of constraints on the constants c and $\hat{c}$ that must be satisfied. At the end of all this (if things work), you should be able to choose constants c and $\hat{c}$ that satisfy all of them — and you should also be able to confirm that your proofs are complete and correct, if you go back and "plug in" these values for the constants.

Finally, in order to prove that

$$T(n) \in \Theta(g(n)),$$

you should try to apply this method twice, once to prove that

$$T(n) \in O(g(n)),$$

and then again to prove that

$$T(n) \in \Omega(g(n)).$$

When this method was introduced for summations it was noted that a "common mistake" should be avoided — you should remember to treat c and $\hat{c}$ as *constants* whose values you don't happen to know yet, and not as *variables*. It's just as easy to make this mistake when you're using the method to deal with recurrences as it is for summations, and you should review the material in Section 7.10.1 if this doesn't seem familiar.

8.4.2 Application to an Example

Suppose you are given the recurrence

$$T(n) = \begin{cases} 1 & \text{if } n = 0, \\ 1 & \text{if } n = 1, \\ T(n-1) + T(n-2) & \text{if } n \geq 2, \end{cases}$$

and you are asked to prove that $T(n) \in \Theta(g(n))$, for

$$g(n) = \left(\tfrac{1+\sqrt{5}}{2}\right)^n.$$

You might be given the additional "hint" that

$$\left(\tfrac{1+\sqrt{5}}{2}\right)^2 = \tfrac{3+\sqrt{5}}{2} = \tfrac{1+\sqrt{5}}{2} + 1.$$

In order to prove this you'd start by trying to establish that

$$T(n) \in O(g(n))$$

for $g(n)$ as above, and in order to do *this* you'd try to use induction on n to prove that

$$T(n) \leq c\left(\tfrac{1+\sqrt{5}}{2}\right)^n \qquad \text{for every integer } n \geq 0,$$

for some positive constant c, using induction on n.

Since the recurrence expresses $T(n)$ in terms of $T(n-1)$ *and* $T(n-2)$, we'll use a form of induction that includes two cases in the basis.

Basis: If $n = 0$ then $T(n) = T(0) = 1$, as defined above, and

$$cg(n) = c\left(\tfrac{1+\sqrt{5}}{2}\right)^n = c\left(\tfrac{1+\sqrt{5}}{2}\right)^0 = c,$$

so that $T(0) \leq cg(0)$ **provided that** $c \geq 1$.

If $n = 1$ then $T(n) = T(1) = 1$, again as defined above, and

$$cg(n) = c\left(\tfrac{1+\sqrt{5}}{2}\right)^n = c\left(\tfrac{1+\sqrt{5}}{2}\right)^1 = c\left(\tfrac{1+\sqrt{5}}{2}\right),$$

so that $T(1) \leq cg(1)$ **provided that** $c \geq \frac{2}{1+\sqrt{5}}$.

Now,

$$\tfrac{2}{1+\sqrt{5}} < 1,$$

so it is necessary and sufficient to choose c to be greater than or equal to 1 in order to satisfy both of the conditions on c that were identified in the basis.

Inductive Step: Suppose now that n is greater than or equal to 0 and that

$$T(n) \leq cg(n) \qquad \text{and} \qquad T(n+1) \leq cg(n+1).$$

It is necessary and sufficient to prove, using only these assumptions, that

$$T(n+2) \leq cg(n+2)$$

as well.

Now since $n \geq 0$, $n + 2 \geq 2$, so that

$$\begin{aligned}
T(n+2) &= T(n+1) + T(n) && \text{by the recursive definition of } T, \\
&\leq c\left(\tfrac{1+\sqrt{5}}{2}\right)^{n+1} + c\left(\tfrac{1+\sqrt{5}}{2}\right)^{n} && \text{by the inductive hypothesis} \\
&= c\left(\tfrac{1+\sqrt{5}}{2}\right)^{n} \cdot \left(\tfrac{1+\sqrt{5}}{2} + 1\right) \\
&= c\left(\tfrac{1+\sqrt{5}}{2}\right)^{n} \cdot \left(\tfrac{1+\sqrt{5}}{2}\right)^{2} && \text{by the above hint,} \\
&= c\left(\tfrac{1+\sqrt{5}}{2}\right)^{n+2} && \text{as required.}
\end{aligned}$$

Thus the "inductive step" can be completed without adding any additional conditions on the constant c.

It follows that we can choose $c = 1$ in order to satisfy all the conditions that were identified in the basis and the inductive step, proving that

$$T(n) \leq \left(\tfrac{1+\sqrt{5}}{2}\right)^{n} \qquad \text{for every integer } n \geq 0.$$

This implies that

$$T(n) \in O(g(n))$$

for $g(n)$ as above.

Next, we should try to prove that $T(n) \in \Omega(g(n))$ by proving, using induction on n, that there is a positive constant $\hat{c}$ such that

$$T(n) \geq \hat{c}\left(\tfrac{1+\sqrt{5}}{2}\right)^{n}$$

for every integer $n \geq 0$.

The desired proof is almost identical to the one given above: All you need to do, to get it, is to

- replace c with $\hat{c}$, throughout;
- reverse all inequalities, replacing "$\leq$" with "$\geq$" and vice-versa, *except*, of course, for the one in the inequality

  $$\tfrac{2}{1+\sqrt{5}} < 1;$$

- conclude at the end of the basis that it is necessary and sufficient to choose

  $$\hat{c} \leq \tfrac{2}{1+\sqrt{5}}$$

  in order to satisfy all the constraints on $\hat{c}$ that had been identified at that point.

After doing this, you should have a correct proof that

$$T(n) \geq \hat{c}g(n) \qquad \text{for every integer } n \geq 0$$

for some positive constant $\hat{c} \leq \frac{2}{1+\sqrt{5}}$, and this implies that

$$T(n) \in \Omega(g(n)).$$

Since $T(n) \in O(g(n))$ and $T(n) \in \Omega(g(n))$, $T(n) \in \Theta(g(n))$, as desired.

8.4.3 Application to Another Example

Suppose that you were given a recurrence of the form

$$T(n) = \begin{cases} 0 & \text{if } n \leq 1, \\ 1 & \text{if } n = 2, \\ T(\lfloor \frac{n}{2} \rfloor) + 1 & \text{if } n \geq 3. \end{cases}$$

Suppose, as well, that you suspected (correctly!) that

$$T(n) \in \Theta(\log_2 n).$$

In order to *prove* that this is the case, using the substitution method, you'd first try to establish by induction on n that

$$T(n) \leq c \log_2 n \qquad \text{for every integer } n \geq 1$$

for some positive constant c. Since the recurrence relates the value of the function on input n to the value on input $\lfloor \frac{n}{2} \rfloor$, rather than on input $n-1$, it'll be most convenient to use the "strong" form of mathematical induction (namely, the last version of mathematical induction introduced in Section 2.3). The proof will therefore look something like the following.

Basis: If $n = 1$ then $T(1) = 0$, as defined by the above recurrence.

On the other hand, if $n = 1$ then $c \log_2 n = c \log_2 1 = 0$ as well, regardless of the choice of the constant c.

So if $n = 1$ then (since $0 \leq 0$), $T(n) \leq c \log_2 n$ for *every* constant $c \geq 0$.

Inductive Step: Suppose, now that $n \geq 1$ and that

$$T(m) \leq c \log_2 m$$

for every integer m such that $1 \leq m \leq n$. It is necessary and sufficient to prove, using only this assumption, that

$$T(n+1) \leq c \log_2 (n+1)$$

as well.

Now, since $n \geq 1$, $n + 1 \geq 2$, and so it follows by the given recurrence for $T(n)$ that

$$\begin{aligned} T(n+1) &= T(\lfloor \tfrac{n+1}{2} \rfloor) + 1 \\ &\leq \left(c \log_2 \lfloor \tfrac{n+1}{2} \rfloor \right) + 1 && \text{by the inductive hypothesis, since } 1 \leq \lfloor \tfrac{n+1}{2} \rfloor \leq n \\ &\leq \left(c \log_2 \left(\tfrac{n+1}{2} \right) \right) + 1 && \text{since } f(x) = \log_2 x \text{ is increasing, and } c > 0 \\ &= c \log_2 (n+1) - c \log_2 2 + 1 \\ &= c \log_2 (n+1) - c + 1 \\ &\leq c \log_2 (n+1) && \text{as required, } \textbf{provided that } c \geq 1. \end{aligned}$$

Now, only one condition on c (apart from the fact that it must be positive) was identified during this proof, namely, the condition that $c \geq 1$ on the last line of the above sequence of equations.

Therefore can "choose" c to be any such constant after the fact — we'll pick $c = 1$ — and conclude that

$$T(n) \leq c \log_2 n \qquad \text{for every integer } n \geq 1$$

for this choice of c. We can now conclude *from this* that

$$T(n) \in O(\log_2 n).$$

Two things should be noted about the above proof. Note first that we started with $n = 1$ instead of with $n = 0$. "By default," you should generally try to start with $n = 0$. We'd run into a problem if we did this here, because we'd need to make an argument about the value of "$\log_2 0$" in order to establish the basis if we did things this way, and of course $\log_2 0$ is undefined. Starting from $n = 1$, instead, eliminates this problem.

The second noteworthy thing has already been mentioned: We're using the strong form of mathematical induction in order to deal with a recurrence which defines $T(n)$ in terms of $T(m)$ where m is *not* $n - 1$ (or $n - 2$) in all cases; the strong form of induction is useful when dealing with a recurrence that expresses $T(n)$ in terms of $T(m)$ where m is smaller than n by a constant *factor* (rather than having $n - m$ being bounded by a constant), as is the case here.

Next, to prove that $T(n) \in \Omega(\log_2 n)$, we should try to prove that

$$T(n) \geq \hat{c} \log_2 n \qquad \text{for every integer } n \geq 1.$$

If the recurrence for T had been slightly different — in particular, if it had been

$$T(n) = \begin{cases} 0 & \text{if } n \leq 1, \\ 1 & \text{if } n = 2, \\ T(\lceil \frac{n}{2} \rceil) + 1 & \text{if } n \geq 3, \end{cases}$$

so it included the *ceiling* instead of the *floor* function, then this would have been reasonably straightforward, in the sense that we could have used the proof given above, after modifying it by replacing c with $\hat{c}$ and changing the direction of the inequalities (replacing "$\leq$" with "$\geq$", and vice-versa) throughout.

However, we *can't* do this, using the recursive definition as originally given, because the resulting "proof" would depend on the inequality

$$\left(\hat{c} \log_2 \lfloor \tfrac{n+1}{2} \rfloor\right) + 1 \geq \left(\hat{c} \log_2 \left(\tfrac{n+1}{2}\right)\right) + 1,$$

which is *incorrect* whenever n is an even integer.

Unfortunately, a problem like this will arise reasonably frequently when you're trying to solve recurrences in order to analyze algorithms.

There are several ways to overcome this problem. One way is to develop a proof using the following outline:

1. Prove that T is a *nondecreasing* function — that is, that $T(n) \leq T(n + 1)$ for every integer $n \geq 1$. This will turn out to be straightforward.

2. Conclude from this that if $n \geq 2$ then

$$T(n) \geq T(2^k), \qquad \text{for } k = \lfloor \log_2 n \rfloor.$$

3. Now, try to prove by *induction on* k (and *not* on n) that there exists a positive constant $\hat{c}$ such that

$$T(2^k) \geq \hat{c} \log_2(2^k) = \hat{c}k \qquad \text{for every integer } k \geq 0;$$

that is, prove that the desired inequality holds for $T(n)$ *whenever* n *is a power of two.*

You'll discover now that the "floor" function doesn't cause a problem, because it will always be applied to an *integer* in your proof (that is, it will never have any "effect"), so the proof would be the same as it would be, if the floor function didn't appear in the function definition at all.

In other words, you'll be taking advantage of the fact that if $\hat{T}$ is a function defined on the real numbers by the recurrence

$$\hat{T}(n) = \begin{cases} 0 & \text{if } n \leq 1, \\ 1 & \text{if } 1 < n \leq 2, \\ \hat{T}(\frac{n}{2}) + 1 & \text{if } n > 2, \end{cases}$$

then $T(2^k) = \hat{T}(2^k)$ for every integer $k \geq 0$, *and* it's reasonably easy to prove that $\hat{T}(n) \geq \log_2 x$ for every real number $x > 0$ (using induction on $\lceil x \rceil$).

So, you'll be able to show that

$$T(n) \geq \hat{c} \log_2 n \qquad \text{if } \hat{c} = 1, \text{ whenever } n \text{ is a power of two.}$$

4. Use the last two things you established to prove that if $n \geq 2$ then

$$\begin{aligned} T(n) &\geq T(2^k) && \text{for } k = \lfloor \log_2 n \rfloor \\ &\geq \lfloor \log_2 n \rfloor \\ &\geq \tfrac{1}{2} \log_2 n, && \text{since } \lfloor \log_2 n \rfloor \geq \tfrac{1}{2} \log_2 n \text{ when } n \geq 2. \end{aligned}$$

Now, this implies that

$$T(n) \in \Omega(\log_2 n).$$

The trick used here — establishing that T is a nondecreasing function, proving the desired inequality for $T(n)$ when n is of a special form, and then combining the two facts to prove a weaker (but sufficient) inequality for $T(n)$ in the general case — will be discussed in more detail in Section 8.6.1, below.

As an exercise, you should try to develop a proof that $T(n) \in \Omega(\log_2 n)$ by following the given outline, for $T(n)$ as above. In case you get stuck (or want to compare your proof to mine), a more complete proof that *does* follow this outline appears below.

1: Proof that T is Nondecreasing

We'll first prove that T is a nondecreasing function, using the recursive definition

$$T(n) = \begin{cases} 0 & \text{if } n \leq 1, \\ 1 & \text{if } n = 2, \\ T(\lfloor \frac{n}{2} \rfloor) + 1 & \text{if } n \geq 3. \end{cases}$$

Clearly, $T(n) \leq T(n+1)$ if n is an integer and $n < 0$, since T is a constant function on the negative integers, so it's only necessary to prove that $T(n) \leq T(n+1)$ for every *nonnegative* integer n.

We'll prove this by induction on n, using the strong form of mathematical induction.

Basis: If $n = 0$ then $T(n) = T(0) = 0$ and $T(n+1) = T(1) = 0$, so $T(n) \leq T(n+1)$ in this case.

Inductive Step: Suppose now that $n \geq 0$ and that

$$T(m) \leq T(m+1)$$

for every integer m such that $0 \leq m \leq n$. It is necessary and sufficient to prove, using only these assumptions, that

$$T(n+1) \leq T(n+2)$$

in order to complete the inductive step.

If $n = 0$ then $T(n+1) = T(1) = 0$ and $T(n+2) = T(2) = 1$, so that $T(n+1) \leq T(n+2)$ in this case.

If $n = 1$ then $T(n+1) = T(2) = 1$ and $T(n+2) = T(3) = T(1)+1 = 1$, so that $T(n+1) \leq T(n+2)$ in this case too.

In all other cases, $n \geq 2$, so that $n+1 \geq 3$ and $n+2 \geq 3$. It therefore follows by the above recursive definition of $T(n)$ that

$$T(n+1) = T(\lfloor \tfrac{n+1}{2} \rfloor) + 1$$

and

$$T(n+2) = T(\lfloor \tfrac{n+2}{2} \rfloor) + 1.$$

Now, either n is odd or n is even.

If n is odd then $\lfloor \frac{n+1}{2} \rfloor = \lfloor \frac{n+2}{2} \rfloor$, so it follows from the above that

$$T(n+1) = T(\lfloor \tfrac{n+1}{2} \rfloor) + 1 = T(\lfloor \tfrac{n+2}{2} \rfloor) + 1 = T(n+2),$$

and this clearly implies that $T(n+1) \leq T(n+2)$.

On the other hand, if n is even, then

$$\lfloor \tfrac{n+2}{2} \rfloor = \lfloor \tfrac{n+1}{2} \rfloor + 1,$$

but it's also true (since $n \geq 2$) that

$$0 \leq \lfloor \tfrac{n+1}{2} \rfloor \leq n.$$

It therefore follows by the above inequalities, and the inductive hypothesis, that

$$T(\lfloor \tfrac{n+1}{2} \rfloor) \leq T(\lfloor \tfrac{n+2}{2} \rfloor)$$

in this case. The desired inequality,

$$T(n+1) \leq T(n+2)$$

can be obtained from the previous one by adding one to both sides.

It now follows by induction on n that $T(n) \leq T(n+1)$ for every integer $n \geq 0$, as desired.

2: Proof That $T(n) \geq T(2^k)$ for $k = \lfloor \log_2 n \rfloor$

This should be fairly obvious, since n and $2^{\lfloor \log_2 n \rfloor}$ are both integers and since

$$2^{\lfloor \log_2 n \rfloor} \leq n.$$

It would be acceptable if you simply wrote this conclusion down, after completing the above proof that T is a nondecreasing function.

However, if you really wanted to, you could prove this last inequality using induction on the difference between n and $2^{\lfloor \log_2 n \rfloor}$.

3: Proof that $T(n) \geq \hat{c} \log_2 n$ if n is a Power of Two

This really *could* be established by taking the proof that "$T(n) \leq c \log_2 n$" from the earlier subsection, replacing c by $\hat{c}$, reversing the directions of the inequalities, and discarding the floors and the ceilings, throughout.

Observe, also, that the recurrence is defined in order to ensure that $T(2^k) = k$ for every integer $k \geq 0$; *this* could be established quite easily by induction on k, and then you could simply choose $\hat{c}$ to be 1 after that.

4: Finishing Things Off

The final piece of the proof has already been given completely, in the description of Step 4 on page 118.

8.5 The Iteration Method

8.5.1 Outline of the Method

When using "the iteration method," you try to use the recursive definition of your function to express it as a summation. This generally involves "unwinding" the recursive definition, or "iterating" it, in order to express the value $T(n)$ of the function at n as a sum or more and more other values along with the value of the function, $T(m)$, at a smaller and smaller argument m.

In general, a pattern should become evident. That is, you should eventually be able to guess the form of the expression you obtained after applying the recursive definition to the original expression l times, for any integer $l \geq 0$ (or, perhaps, $l \geq 1$), as a function of l.

After you've managed to guess what the pattern should be, you use induction (on "l") in order to prove that the pattern really does hold.

It is generally possible, at this point, to use the pattern to express $T(n)$ as a summation of values that *aren't* recursively defined, plus (usually) either $T(1)$ or $T(0)$ — whichever is explicitly given — by choosing an appropriate value to substitute for l.

After all this, you're left with the problem of finding a closed form for a *summation*, rather than for a recurrence.

8.5.2 Application to an Example

Consider, again, the recurrence that was used in the second example for the previous method,

$$T(n) = \begin{cases} 0 & \text{if } n \leq 1, \\ 1 & \text{if } n = 2, \\ T(\lfloor \frac{n}{2} \rfloor) + 1 & \text{if } n \geq 3. \end{cases}$$

Before we begin, there is something that should be noted that will be useful here and for later problems:

Lemma 8.2. *If h is an integer that is greater than or equal to two and n and i are nonnegative integers, then*

$$\left\lfloor \frac{\lfloor \frac{n}{h^i} \rfloor}{h} \right\rfloor = \left\lfloor \frac{n}{h^{i+1}} \right\rfloor.$$

Proof. This is easily proved once you observe that every nonnegative integer n has a "base h" representation,

$$n = \sum_{j=0}^{\ell} a_j h^j$$

where $0 \leq a_j < h$ for $0 \leq j \leq \ell$ and where either $a_\ell \neq 0$ or $n = 0$ and $\ell = 0$. If n has this representation, and i is a nonnegative integer, then

$$\left\lfloor \frac{n}{h^i} \right\rfloor = \begin{cases} \sum_{j=0}^{\ell-i} a_{i+j} h^j & \text{if } i \leq \ell, \\ 0 & \text{if } i > \ell; \end{cases}$$

and this can be used to validate the above identity, directly. □

Now, back to the above example. If n is sufficiently large then we can repeatedly apply the recursive definition (that is, apply it "iteratively") and observe that

$$\begin{aligned}
T(n) &= T(\lfloor \tfrac{n}{2} \rfloor) + 1 && \text{by the recursive definition of } T(n) \\
&= (T(\lfloor \tfrac{n}{4} \rfloor) + 1) + 1 && \text{by the recursive definition again, and the lemma} \\
&= T(\lfloor \tfrac{n}{2^2} \rfloor) + \sum_{i=1}^{2} 1 \\
&= (T(\lfloor \tfrac{n}{2^3} \rfloor) + 1) + \sum_{i=1}^{2} 1 && \text{by the recursive definition and the lemma, again} \\
&= T(\lfloor \tfrac{n}{2^3} \rfloor) + \sum_{i=1}^{3} 1 \\
&= (T(\lfloor \tfrac{n}{2^4} \rfloor) + 1) + \sum_{i=1}^{3} && \text{by the recursive definition and the lemma, again} \\
&= T(\lfloor \tfrac{n}{2^4} \rfloor) + \sum_{i=1}^{4} \cdots
\end{aligned}$$

The above derivation is correct as long as $\lfloor \frac{n}{2^3} \rfloor \geq 3$, that is, provided that $n \geq 24 = 3 \cdot 2^3$. (Why?)

Based on this, you might *guess* the following.

Conjecture. If $l \geq 0$ and $n \geq 3 \cdot 2^{l-1}$ then

$$T(n) = T(\lfloor \tfrac{n}{2^l} \rfloor) + \sum_{i=1}^{l} 1. \tag{8.5}$$

It turns out that this conjecture is *correct* — and you can prove it (fairly easily) as follows.

Proof of Conjecture. This will be proved by induction on h.

Basis: If $l = 0$ then

$$T(\lfloor \tfrac{n}{2^l} \rfloor) + \sum_{i=1}^{l} 1 = T(\lfloor \tfrac{n}{1} \rfloor + 0 = T(n),$$

as required.

Inductive Step: Suppose now that $l \geq 0$, and that if $n \geq 3 \cdot 2^{l-1}$ then

$$T(n) = T(\lfloor \tfrac{n}{2^l} \rfloor + \sum_{i=1}^{l} 1.$$

It is necessary and sufficient to prove, using only this assumption, that if $n \geq 3 \cdot 2^l$ then

$$T(n) = T(\lfloor \tfrac{n}{2^{l+1}} \rfloor) + \sum_{i=1}^{l+1} 1.$$

Suppose, then, that $n \geq 3 \cdot 2^l$. Then $n \geq 3 \cdot 2^{l-1}$ as well, so that

$$T(n) = T(\lfloor \tfrac{n}{2^l} \rfloor) + \sum_{i=1}^{l} 1$$

by the inductive hypothesis.

Since $n \geq 3 \cdot 2^l$, $\frac{n}{2^l} \geq 3$, so that $T(\lfloor \frac{n}{2^l} \rfloor) = T(\lfloor \frac{n}{2^{l+1}} \rfloor) + 1$ (using the recurrence for T and the above lemma). Therefore,

$$\begin{aligned} T(n) &= T(\lfloor \tfrac{n}{2^l} \rfloor) + \sum_{i=1}^{l} 1 && \text{as argued above,} \\ &= (T(\lfloor \tfrac{n}{2^{l+1}} \rfloor) + 1) + \sum_{i=1}^{l} 1 \\ &= T(\lfloor \tfrac{n}{2^{l+1}} \rfloor) + \sum_{i=1}^{l+1} 1, && \text{as desired.} \end{aligned}$$

It follows by induction on l that if $n \geq 3 \cdot 2^{l-1}$ then

$$T(n) = T(\lfloor \tfrac{n}{2^l} \rfloor) + \sum_{i=1}^{l} 1$$

for every integer $l \geq 0$. □

Now, if $n \geq 3$, we can set

$$l = \lfloor \log_2 n \rfloor - 1;$$

$l \geq 0$ since $n \geq 3$, and

$$n = 2^{(\log_2 n)} \geq 2^{\lfloor \log_2 n \rfloor} = 2^{l+1} = 4 \cdot 2^{l-1} \geq 3 \cdot 2^{l-1},$$

so that we can immediately conclude — using the result that was established by induction on l — that

$$\begin{aligned} T(n) &= T(\lfloor \tfrac{n}{2^l} \rfloor) + \sum_{i=1}^{l} 1 \\ &= T(\lfloor \tfrac{n}{2^{\lfloor \log_2 n \rfloor - 1}} \rfloor) + \sum_{i=1}^{\lfloor \log_2 n \rfloor - 1} 1. \end{aligned}$$

Now,

$$\frac{n}{2} < 2^{\lfloor \log_2 n \rfloor} \leq n,$$

so

$$\frac{n}{4} < 2^{\lfloor \log_2 n \rfloor - 1} \leq \frac{n}{2},$$

and therefore (since $n > 0$),

$$2 \leq \tfrac{n}{2^{\lfloor \log_2 n \rfloor - 1}} < 4.$$

It follows that

$$2 \leq \lfloor \tfrac{n}{2^{\lfloor \log_2 n \rfloor - 1}} \rfloor \leq 3,$$

so that

$$T(\lfloor \tfrac{n}{2^{\lfloor \log_2 n \rfloor - 1}} \rfloor)$$

is either $T(2)$ or $T(3)$.

However, an examination of the recurrence for $T(n)$ confirms that

$$T(2) = T(3) = 1,$$

so we can conclude from all this that

$$\begin{aligned} T(n) &= T(\lfloor \tfrac{n}{2^{\lfloor \log_2 n \rfloor - 1}} \rfloor) + \sum_{i=1}^{\lfloor \log_2 n \rfloor - 1} 1 \\ &= 1 + \sum_{i=1}^{\lfloor \log_2 n \rfloor - 1} 1. \end{aligned}$$

Now we're left with the problem of finding a closed form for a summation. In the case of this example, it's a trivial one, and it should be clear now that

$$T(n) = 1 + \sum_{i=1}^{\lfloor \log_2 n \rfloor - 1} 1 = 1 + (\lfloor \log_2 n \rfloor - 1) = \lfloor \log_2 n \rfloor.$$

Of course, since the sum was so easy to solve, you could have solved it immediately — writing l instead of

$$\sum_{i=1}^{l} 1$$

wherever the former expression appears in the above derivation. The sum has been left in, in this example, to make it look more like what you'd have when the problem is harder.

A comparison with the derivation given in the previous subsection should confirm that we've now obtained a more precise result than we had before — we'd only showed that

$$T(n) \in \Theta(\log_2 n)$$

using the substitution method. However, we had to perform some additional algebraic manipulations, and see and confirm a pattern (as given by the above "conjecture"), in apply the iteration method.

You don't often get a "pattern" that's quite as simple as the above. It's slightly more common to get a pattern that looks more like

$$T(n) = a_l T(\lfloor \tfrac{n}{b_l} \rfloor) + \sum_{i=1}^{l} f(i),$$

where a_l and b_l are frequently positive integers that depend on l (but not on n) and where $f(i)$ is some function of i. Other "patterns" are possible as well.

8.6 Simplifying the Problem by (Correctly) Modifying the Recurrence

The examples in the last few sections have involved "sanitized" recurrences that resemble but are simpler than the recurrences that were derived when the example algorithms *fib* and *BSearch* were analyzed.

You'll have trouble applying either the substitution method or the iteration method directly to these recurrences, because of things like

- the use of floors and ceilings to ensure that the function being defined is only recursively applied to integers, and
- the appearance of extra additive terms appearing in the right hand side of the definitions.

The "techniques" in this section are actually methods that allow us to simplify recurrences, reducing the problems associated with ceiling and floor functions and additional additive terms. We'll generally apply these techniques to convert a given recurrence into one that is easier to solve, in such a way that the solution for the simpler recurrence can be used to recover a solution or bound for the one we started with.

8.6.1 Restricting the Domain

The second example for the "substitution method," particularly the part in which we tried to prove that

$$T(n) \in \Omega(\log_2 n),$$

also illustrates the use of the simplification method. This method seems useful when floors and ceilings are involved — especially when $T(n)$ is defined in terms of either $T(\lfloor \frac{n}{h} \rfloor)$ or $T(\lceil \frac{n}{h} \rceil)$, where h is some positive integer that is greater than or equal to two.

The "method" can be described as follows.

1. Prove (generally using induction on n) that the function T is nondecreasing, that is, $T(n) \leq T(n+1)$ for every integer $n \geq 0$, using the given recurrence for T.

2. Identify an infinite set of integers n such that $T(n) = \hat{T}(n)$, where $\hat{T}$ is a function defined on the reals, by taking the given recurrence for T and eliminating all applications of the floor or ceiling function — and, furthermore, where $\hat{T}(n)$ is only recursively defined in terms of $\hat{T}(m)$ where $0 \leq m < n$ and m is an *integer*.

   In the special case given above, that $T(n)$ is only defined in terms of the value of the function at the floor or ceiling of $\frac{n}{h}$, you can generally choose this set of integers to include all the integer powers of h.

3. Now, try to apply the substitution or iteration method to $T(n)$ (or $\hat{T}(n)$) for integers n that belong to the set you chose in the previous step. You'll frequently find that this is easier than it would be to apply the substitution method to $T(n)$ directly in the general case, because the floor and ceiling functions don't cause as many problems.

4. Let's suppose that you've managed to prove that

   $$\hat{c}g(n) \leq T(n) \leq cg(n)$$

   for positive constants $\hat{c}$ and c, some useful function g, in the *special case* that n belongs to the infinite set chosen above. Now, take advantage of the fact that T is a nondecreasing function (as proved in step 1) to conclude that

   $$\hat{c}g(n_L) \leq T(n_L) \leq T(n) \leq T(n_U) \leq cg(n_U)$$

   where n_L and n_U are members of the infinite set you chose in step 2, such that

   $$n_L \leq n \leq n_U.$$

   In the case that your infinite set consisted of all the integer powers of h, as above, you'd generally choose

   $$n_L = h^{\lfloor \log_h n \rfloor} \qquad \text{and} \qquad n_U = h^{\lceil \log_h n \rceil}$$

   so that

   $$n_L > \frac{n}{h} \qquad \text{and} \qquad n_U < h \cdot n.$$

   If the function g grows sufficiently *slowly* (say, it's only logarithmic or polynomial, rather than exponential in n), then $g(n_U)$ won't be too much larger than $g(n_L)$, and this may be good enough to allow you to prove that

   $$T(n) \in \Theta(g(n)).$$

As noted above, this method was applied in the derivation at the end of Section 8.4.3.

8.6.2 Change of Variable

You can sometimes make progress by rewriting a recursive definition, so that the function is defined in terms of a different variable (namely, some function of the variable that the first recurrence mentioned).

Consider, again, the recurrence used in the second example for "the substitution method,"

$$T(n) = \begin{cases} 0 & \text{if } n \leq 1, \\ 1 & \text{if } n = 2, \\ T(\lfloor \frac{n}{2} \rfloor) + 1 & \text{if } n \geq 3. \end{cases} \tag{8.6}$$

It was argued (in essence) in that section that it would be sufficient to consider the special case that n is a power of two, that is, that

$$n = 2^k \tag{8.7}$$

for some *integer* k.

Suppose, then, that this is the case, and define the function in terms of k — using equations 8.6 and 8.7 to rewrite it as

$$\begin{aligned} \hat{T}(k) &= T(n) \\ &= T(2^k) \\ &= \begin{cases} 0 & \text{if } 2^k \leq 1, \\ 1 & \text{if } 2^k = 2, \\ T(\lfloor \frac{2^k}{2} \rfloor) + 1 & \text{if } 2^k \geq 3, \end{cases} \\ &= \begin{cases} 0 & \text{if } k \leq 0, \\ 1 & \text{if } k = 1, \\ T(2^{k-1}) + 1 & \text{if } k \geq 2, \end{cases} \\ &= \begin{cases} 0 & \text{if } k \leq 0, \\ 1 & \text{if } k = 1, \\ \hat{T}(k-1) + 1 & \text{if } k \geq 2. \end{cases} \end{aligned}$$

In the second-to-last line, we took advantage of the fact that k was required to be an *integer* to conclude that k must be greater than or equal to two in order for 2^k to be greater than or equal to three.

Rewriting this more succinctly, we have

$$\hat{T}(k) = \begin{cases} 0 & \text{if } k \leq 0, \\ 1 & \text{if } k = 1, \\ \hat{T}(k-1) + 1 & \text{if } k \geq 2. \end{cases}$$

and it should be *very* easy to see (and *prove* by induction on k) that this recurrence has the solution

$$\hat{T}(k) = \max(0, k).$$

Since $k = \log_2 n$ and $T(n) = \hat{T}(k)$,

$$T(n) = \hat{T}(k) = \max(0, k) = \max(0, \log_2 n) = \log_2 n \qquad \text{if } k \geq 0, \text{ so that } n = 2^k \geq 1.$$

Now (if we wish to) we can deal with the general case, by proving as before that $T(n)$ is a nondecreasing function of n, so that

$$T(2^{k-1}) \leq T(n) \leq T(2^k)$$

if $k = \log_2 n$; once again, this would allow us to conclude immediately that

$$\log_2 n - 1 \leq T(n) \leq \log_2 n$$

for every integer $n \geq 1$.

8.6.3 Change of Function

It can also be useful to define a different *function* in terms of the old one by deriving a recurrence for the new function from the recurrence from the old one. If this is done carefully then the new recurrence will be easier to solve, and bounds for the original function can be obtained from bounds for the new one.

Consider the time function that was defined for the algorithm *fib*. As shown in Section 8.2.2, this satisfies the recurrence given as equation (8.1), which is as follows:

$$T(n) = \begin{cases} 2 & \text{if } n \leq 0, \\ 3 & \text{if } n = 1, \\ T(n-1) + T(n-2) + 3 & \text{if } n \geq 2. \end{cases}$$

The additional additive term "+3" in the final case makes this difficult to handle using the techniques that have been introduced so far. (Give it a try, if you don't see why this is the case.)

Suppose, instead, that we consider the function

$$U(n) = T(n) + 3.$$

Then

$$U(n) = T(n) + 3 = 5 \qquad \text{if } n \leq 0,$$

and

$$U(1) = T(1) + 3 = 6.$$

If $n \geq 2$ then $T(n) = T(n-1) + T(n-2) + 3$, so that

$$\begin{aligned} U(n) &= T(n) + 3 \\ &= (T(n-1) + T(n-2) + 3) + 3 \\ &= (T(n-1) + 3) + (T(n-2) + 3) \\ &= U(n-1) + U(n-2). \end{aligned}$$

That is,

$$U(n) = \begin{cases} 5 & \text{if } n \leq 0, \\ 6 & \text{if } n = 1, \\ U(n-1) + U(n-2) & \text{if } n \geq 2. \end{cases}$$

Now, since the Fibonacci numbers satisfy the recurrence

$$F_n = \begin{cases} 0 & \text{if } n = 0, \\ 1 & \text{if } n = 1, \\ F_{n-1} + F_{n-2} & \text{if } n \geq 2, \end{cases}$$

you should find that it is *very* easy to use the substitution method to prove that

$$U(n) \in \Theta(F_{n+1}).$$

Indeed, you should be able to show that

$$5F_{n+1} \leq U(n) \leq 6F_{n+1} \qquad \text{for } n \geq 0.$$

Now, since $U(n) = T(n) + 3$, $T(n) = U(n) - 3$, so

$$5F_{n+1} - 3 \leq T(n) \leq 6F_{n+1} - 3$$

for all $n \geq 0$, as well.

Alternatively, you could use a proof that's similar to the one given in Section 8.4.2 and prove that

$$U(n) \in \Theta\left(\left(\tfrac{1+\sqrt{5}}{2}\right)^n\right),$$

and then conclude that

$$T(n) \in \Theta\left(\left(\tfrac{1+\sqrt{5}}{2}\right)^n\right)$$

as well.

8.6.4 Concluding Remarks about Simplifications

You might be wondering *when* you should try to simplify a recurrence, and which "simplification" you should try to make. Unfortunately, no easy-to-follow rules for this come to mind — except, possibly, that you should consider making a "change of function," by adding or subtracting a constant or some other slow growing function, whenever you are having trouble because of extra additive terms in a recurrence.

So, this would seem to be a case where experience or practice is required.

Here's a practice exercise: Consider the function $T(m)$ that was produced when the binary search algorithm was analyzed. It has the recurrence shown in equation 8.2 on page 112.

Prove, by induction on m, that this is a nondecreasing function of m.

Now, it won't help very much *at all* to consider the case that m is a power of two.

On the other hand, it should turn out to be *very* helpful to consider the case that $m + 1$ is a power of two, instead.

That is, you should consider the special case that

$$m = 2^k - 1$$

for an integer $k \geq 0$.

Use the "change of function" approach, given above, to use the recursive definition of $T(m)$ in order to produce a recursive definition for $\hat{T}(k)$, where k is an integer, $m = 2^k - 1$, and $\hat{T}(k) = T(m)$.

You should find, after you've done this, that the iteration method can be used without too much trouble to find a closed form for the function $\hat{T}$.

Then, use the fact (which you proved at the beginning) that T is an increasing function of m, to recover a tight asymptotic bound, in closed form, for T. That is, use all this to discover a function $g(m)$ which is in closed form, such that you can conclude from the above that

$$T(m) \in \Theta(g(m)).$$

8.7 The Master Theorem

The last few sections have discussed general techniques for solving recurrences that aren't always very easy to apply. This section, and the one that follows it, contain more specialized techniques that can only be applied to solve recurrences that have particular forms, but which are generally easy to apply (when they're applicable at all).

8.7.1 A Common Pattern for a Recurrence

Many interesting algorithms solve an instance of a problem by creating and recursively solving a *fixed* number of smaller instances of the problem that are each *much* smaller — *by a constant factor* — than the original one, and then generating a solution for the original instance using the solutions for the derived ones.

Suppose therefore that a, b, and c are constants (*independent* of n) such that a is greater than or equal to one, b is strictly greater than one, and c is greater than or equal to zero. Let $f(n)$ be some asymptotically positive function of n, and suppose we have an algorithm with the following properties:

1. The algorithm uses more than zero steps, but at most c steps, on every input instance whose size is less than b.

2. It solves instances of size n, for $n \geq b$, by forming a smaller instances that are each of size at most $\lceil \frac{n}{b} \rceil$, solving these smaller instances recursively, and then deriving a solution for the original instance using the solutions for the derived ones.

3. The number of steps it uses for an input of size n, *excluding* the cost of the recursive calls, is at most $f(n)$ in the worst case.

Let $T(n)$ be the number of steps used by this algorithm on an input instance of size n in the worst case, and **suppose $T(n)$ is nondecreasing**. Then $T(n)$ satisfies the following recurrence:

$$T(n) \leq \begin{cases} c & \text{if } n < b, \\ aT\left(\lceil \frac{n}{b} \rceil\right) + f(n) & \text{if } n \geq b. \end{cases}$$

The assumption that $T(n)$ is nondecreasing was used in order to bound the time needed to solve an instance of size *less than or equal to* $\lceil \frac{n}{b} \rceil$ by $T\left(\lceil \frac{n}{b} \rceil\right)$, above.

Similar algorithms, with slightly different behaviours, might have running times bounded by recurrences involving $T(\lfloor \frac{n}{b} \rfloor)$ instead of $T(\lceil \frac{n}{b} \rceil)$.

With additional effort (proving that the "worst case" really can arise, on a regular basis, when subproblems are being formed), you can sometimes prove that an algorithm's running time *also*

satisfies a recurrence like

$$T(n) \geq \begin{cases} \hat{c} & \text{if } n < b, \\ aT\left(\lceil \frac{n}{b} \rceil\right) + \hat{f}(n) & \text{if } n \geq b, \end{cases}$$

as well, where $\hat{c}$ is also a positive constant (possibly different from c) and $\hat{f}(n)$ is also an asymptotically positive function (possibly different from $f(n)$ — but, ideally, in $\Theta(f(n))$). That is, you can sometimes prove that the solution of a recurrence of this form is also a *lower bound* for the number of steps used by an algorithm on inputs of size n in the worst case.

Note, though, that this doesn't follow immediately from the properties of the algorithm that are stated before the first recurrence.

The following result can often be used to find closed forms for functions bounding the worst case running times of these algorithms.

Theorem 8.3 (Master Theorem). *Let a, b, and c be constants such that a is greater than or equal to one, b is strictly greater than one, c is greater than zero, and let $f(n)$ be an asymptotically positive function of n such that $f(n)$ is defined whenever $n \geq b$.*

Let $T(n)$ be a function such that $0 < T(n) \leq c$ whenever $n < b$ and such that

$$T(n) = aT\left(\lceil \frac{n}{b} \rceil\right) + f(n) \tag{8.8}$$

whenever $n \geq b$. Then $T(n)$ can be bounded asymptotically as follows.

1. *If $f(n) \in O(n^{(\log_b a)-\epsilon})$ for some constant $\epsilon > 0$ (which does not depend on n), then $T(n) \in \Theta(n^{\log_b a})$.*
2. *If $f(n) \in \Theta(n^{\log_b a})$ then $T(n) \in \Theta(n^{\log_b a} \log_2 n)$.*
3. *If $f(n) \in \Omega(n^{(\log_b a)+\epsilon})$ for some constant $\epsilon > 0$ (which does not depend on n), and there exists a constant d such that $d < 1$ and*

   $$af\left(\lceil \frac{n}{b} \rceil\right) \leq df(n)$$

   for all sufficiently large n, then $T(n) \in \Theta(f(n))$.

Furthermore, the function $T(n)$ is also bounded asymptotically as above if $0 < T(n) \leq c$ whenever $n < b$ and

$$T(n) = aT\left(\lfloor \frac{n}{b} \rfloor\right) + f(n)$$

whenever $n \geq b$, instead of satisfying equation (8.8).

Two things should be noted about this theorem:

1. It doesn't *quite* apply to recurrences of the form given in the previous subsection, because those included inequalities, and the recurrence given in the statement of the theorem is an equality. However, we'll see that the master theorem is extremely useful in spite of this.
2. As we'll see later, it also doesn't specify an asymptotic bound for *every* recurrence of this form.

8.7.2 Proof of the Master Theorem

There isn't time to include a proof of the above theorem in lectures, and it isn't necessary to understand the proof in order to apply it.

However, a proof *is* included in Section 4.4 of Cormen, Leiserson, and Rivest's book [3]. Slightly different notation is used in their book (and, you should read the material on pages 53–54 of their book, before looking at Section 4.4). However, the "master theorem" given above is implied by the version of the theorem that is stated and proved in the text, so that the proof in Cormen, Leiserson, and Rivest's book proves the above theorem (as well as the version in their text).

You might be wondering why the other techniques for analyzing recurrences were included in CPSC 413, if the master theorem exists. One reason is that there *are* useful recursive programs whose running times are given by recurrences that can't be solved using the master theorem. Another (in case you want to understand a proof of what you're using) is that the *proof* of the master theorem uses these other techniques, as you'll discover if you look at Cormen, Leiserson, and Rivest's book.

8.7.3 The "Regularity Condition"

The final case covered by Theorem 8.3 includes what Cormen, Leiserson, and Rivest call a "regularity condition," namely (for the version given above) that that there is some constant d such that $d < 1$ and

$$af\left(\lceil \tfrac{n}{b} \rceil\right) \leq df(n)$$

for all sufficiently large n.

As Cormen, Leiserson, and Rivest note, this condition is satisfied in virtually all the cases when you'd like to apply this in order to analyze an algorithm and the other conditions required for this case are satisfied.

However, it doesn't hold for all functions. Consider, for example, the function

$$f(n) = \begin{cases} 0 & \text{if } n \leq 1, \\ 2^{(2^{(2^u)})} & \text{where } u = \lceil \log_2(\log_2 n) \rceil, \text{ if } n \geq 2. \end{cases}$$

If f is is defined as above then $f(n) \in \Omega(2^n)$; indeed, $f(n) \geq 2^n$ for every integer $n \geq 2$.

However, if we define a function T by the recurrence

$$T(n) = \begin{cases} 1 & \text{if } n \leq 1, \\ 4T\left(\lceil \tfrac{n}{2} \rceil\right) + f(n) & \text{if } n \geq 2, \end{cases}$$

then this corresponds to a recurrence with the form described in the master theorem, with $a = 4$, $b = 2$, and $f(n)$ as above. Since $\log_b a = 2$, it's certainly the case that $f(n) \in \Omega(n^{\log_b a + \epsilon})$ for a positive constant ϵ. However, the third case of the master theorem is not satisfied, because there are infinitely many integers n such that $\lceil \log_2(\log_2 \frac{n}{2}) \rceil = \lceil \log_2(\log_2 n) \rceil$, so that

$$af\left(\lceil \tfrac{n}{b} \rceil\right) = 4f\left(\lceil \tfrac{n}{2} \rceil\right) = 4f(n) > f(n) \quad \text{infinitely often,}$$

eliminating the possibility that there might exist any constant $d < 1$ such that

$$af\left(\lceil \tfrac{n}{b} \rceil\right) \leq df(n)$$

for all sufficiently large n. That is, the third case does not hold because this "regularity condition" is not satisfied.

Worse yet, it isn't just the case that the conditions described as necessary in the theorem are not satisfied: The conclusion you'd like to achieve in this case isn't satisfied either, because

$$T(n) \in \Omega(f(n)) \qquad \text{but} \qquad T(n) \notin \Theta(f(n))$$

when $T(n)$ is defined as above.

8.7.4 A Simplified Master Theorem for an Important Special Case

Theorem 8.4 (Simplified Master Theorem). *Let a, b, and c be constants such that a is greater than or equal to one, b is strictly greater than one, c is greater than zero, and let $f(n)$ be an asymptotically positive function of n such that $f(n)$ is defined whenever $n \geq b$.*

Suppose, furthermore, that

$$f(n) \in \Theta(n^{\alpha}(\log_2 n)^{\beta})$$

for constants α and β (that are, as usual, independent of n).

Now let $T(n)$ be a function such that $0 < T(n) \leq c$ whenever $n < b$ and such that

$$T(n) = aT\left(\lceil \tfrac{n}{b} \rceil\right) + f(n) \tag{8.9}$$

whenever $n \geq b$. Then $T(n)$ can be asymptotically bounded as follows.

1. *If $\alpha < \log_b a$ then $T(n) \in \Theta(n^{\log_b a})$.*
2. *If $\alpha = \log_b a$ and $\beta = 0$ then $T(n) \in \Theta(n^{\log_b a} \log_2 n)$.*
3. *If $\alpha > \log_b a$ then $T(n) \in \Theta(f(n))$.*

Furthermore, the function $T(n)$ is also bounded asymptotically as above if $0 < T(n) \leq c$ whenever $n < b$ and

$$T(n) = aT\left(\lfloor \tfrac{n}{b} \rfloor\right) + f(n)$$

whenever $n \geq b$, instead of satisfying equation (8.9), above.

This version of the theorem is more restrictive than the original one, since it includes the new condition that

$$f(n) \in \Theta(n^{\alpha}(\log_2 n)^{\beta}).$$

Therefore, there may be cases when you *must* try to use the original version, because this last condition is not satisfied.

However, this second version is easier to apply, when it can be applied at all — note that the constant "ϵ" mentioned in the original version (not to mention, the need to prove asymptotic bounds in order to established each of the three cases), and the "regularity condition" (for the third case) have been eliminated.

8.7.5 Proof of the Simplified Version of the Theorem (Optional)

A proof of this simplified version of the theorem is given in the rest of this section. More precisely, it's shown here that Theorem 8.4 is a consequence of Theorem 8.3. This isn't required reading for CPSC 413.

However, it *might* be of interest if you'd like to discover how one can show that "the regularity condition" is satisfied when you're trying to *apply* the original version of the master theorem; something like this is included in the proof of the third case that's mentioned in the simplified version of the theorem.

Recall now that each version of the theorem gives three different asymptotic bounds corresponding to three possible cases. Since each of the *conclusions* (asymptotic bounds) for the three cases in the simplified theorem is the same as the conclusion given in the corresponding case in the original theorem, it is sufficient to show that the each of the *conditions* that must be satisfied (as stated in the simplified theorem) implies the corresponding condition in the original version of the theorem. Thus, the three cases will each be considered separately, and this is how the first two cases will be handled. As shown below, the final case will require a bit more work.

Suppose, henceforth, that $f(n) \in \Theta(n^\alpha (\log_2 n)^\beta)$ for constants α and β.

The First Case

If the first case for the simplified theorem was applicable then $\alpha < \log_b a$. Let

$$\epsilon = \tfrac{1}{2} (\log_b a - \alpha) ;$$

then $\epsilon > 0$ and $\log_b a = \alpha + 2\epsilon$.

Since $f(n) \in \Theta(n^\alpha (\log_2 n)^\beta)$, and $f(n)$ is asymptotically positive, there exist constants $d > 0$ and $N \geq 0$ such that

$$0 < f(n) \leq d n^\alpha (\log_2 n)^\beta \qquad \text{whenever } n \geq N.$$

It follows that if $n \geq N$ then

$$0 \leq \frac{f(n)}{n^{\log_b a - \epsilon}} \leq \frac{d n^\alpha (\log_2 n)^\beta}{n^{\alpha+\epsilon}} = \frac{d (\log n)^\beta}{n^\epsilon}.$$

Since $(\log_2 n)^\beta \in o(n^\epsilon)$ (for any constant β),

$$\lim_{n \to +\infty} \frac{d (\log_2 n)^\beta}{n^\epsilon} = 0,$$

and this implies that

$$\lim_{n \to +\infty} \frac{f(n)}{n^{\log_b a - \epsilon}} = 0$$

as well. Therefore $f(n) \in O(n^{\log_b a - \epsilon})$, so the first case of the original version of the master theorem is applicable, as required.

The Second Case

If the second case of the simplified theorem is applicable then $\alpha = \log_b a$ and $\beta = 0$, so that

$$f(n) \in \Theta(n^\alpha) = \Theta(n^{\log_b a}),$$

and the second case of the original theorem is applicable too.

The Third Case

Suppose, finally, that the third case of the simplified version of the master theorem is applicable; then $\alpha > \log_b a$.

The Lower Bound is Easy. It is easy to establish by induction on n that $T(n) \geq 0$ for every positive integer n. This, and the original recursive definition of T, imply that

$$T(n) \geq 0 + f(n) = f(n) \quad \text{whenever } n \geq b,$$

so that $T(n) \in \Omega(f(n))$.

Switching Recurrences. Since $f(n) \in \Theta(n^\alpha (\log_2 n)^\beta)$, there exist constants $e > 0$ and $N \geq 0$ such that

$$f(n) \leq e n^\alpha (\log_2 n)^\beta \quad \text{whenever } n \geq N.$$

Since it's true that $n^\alpha (\log_2 n)^\beta > 0$ whenever $n \geq b > 1$, this implies that

$$f(n) \leq \hat{e} n^\alpha (\log_2 n)^\beta \quad \text{whenever } n \geq b,$$

for some different (usually, larger) positive constant $\hat{e}$.

Now let

$$U(n) = \begin{cases} T(n) & \text{if } n < b, \\ aU\left(\lceil \frac{n}{b} \rceil\right) + \hat{f}(n) & \text{if } n \geq b, \end{cases}$$

where $\hat{f}(n) = \hat{e} n^\alpha (\log_2 n)^\beta \geq f(n)$ whenever $n \geq b$.

It is easily proved (by induction on n) that $T(n) \leq U(n)$ for every integer $n \geq 0$. Therefore, if we can manage to prove that $U(n) \in O(f(n))$ then we can conclude that $T(n) \in O(f(n))$ as well. Since we've already established that $T(n) \in \Omega(f(n))$, this will imply that $T(n) \in \Theta(f(n))$, as required.

The remainder of this proof will therefore serve to establish that $U(n) \in O(f(n))$. We'll prove that by showing that the third case of the original master theorem can be applied to the above recurrence for $U(n)$ (rather than the recurrence for $T(n)$).

Confirming the First Condition. Since the third case of the simplified master theorem is applicable, $\alpha > \log_b a$. Therefore, let

$$\epsilon = \tfrac{1}{2}(\alpha - \log_b a);$$

then $\epsilon > 0$ and $\alpha = (\log_b a) + 2\epsilon$. It follows that if $n \geq b$ then

$$\frac{\hat{f(n)}}{n^{(\log_b a)+\epsilon}} \geq \frac{\hat{e} n^\alpha (\log_2 n)^\beta}{n^{\alpha-\epsilon}} = \hat{e} \cdot \frac{n^\epsilon}{(\log_2 n)^{-\beta}}.$$

Since $n^\epsilon \in \omega((\log_2 n)^{-\beta})$,

$$\lim_{n \to +\infty} \frac{n^\epsilon}{(\log_2 n)^{-\beta}} = +\infty,$$

and the above inequality implies that

$$\lim_{n\to+\infty} \frac{\hat{f}(n)}{n^{(\log_b a)+\epsilon}} = +\infty$$

as well. Therefore

$$f(n) \in \Omega(n^{(\log_b a)+\epsilon}),$$

as is needed to satisfy the *first* condition needed to establish that the third case of the master theorem is applicable.

Confirming the Second ("Regularity") Condition (Sketch). Finally, note that

$$\begin{aligned}\frac{a\hat{f}\left(\lceil \frac{n}{b} \rceil\right)}{\hat{f}(n)} &= \frac{a\hat{e}\lceil (\frac{n}{b})\rceil^{\alpha} \left(\log_2 \lceil \frac{n}{b} \rceil\right)^{\beta}}{\hat{e}n^{\alpha}(\log_2 n)^{\beta}} \\ &= \frac{a\lceil (\frac{n}{b})\rceil^{\alpha} \left(\log_2 \lceil \frac{n}{b} \rceil\right)^{\beta}}{n^{\alpha}(\log_2 n)^{\beta}}.\end{aligned}$$

Suppose now that $\beta \geq 0$; then this can be used to establish that

$$\frac{a\left(\frac{n}{b}\right)^{\alpha}\left(\log_2 \frac{n}{b}\right)^{\beta}}{n^{\alpha}(\log_2 n)^{\beta}} \leq \frac{a\hat{f}\left(\lceil \frac{n}{b} \rceil\right)}{\hat{f}(n)} \leq \frac{a\left(\frac{n}{b}+1\right)^{\alpha}\left(\log_2 \left(\frac{n}{b}+1\right)\right)^{\beta}}{n^{\alpha}(\log_2 n)^{\beta}}.$$

Now, using a considerable amount of algebra, it can be established that the limit as n approaches $+\infty$ of both the lower bound and the upper bound are the same, and both are equal to $\left(\frac{1}{b}\right)^{2\epsilon}$.

It follows that

$$\lim_{n\to+\infty} \frac{a\hat{f}\left(\lceil \frac{n}{b} \rceil\right)}{\hat{f}(n)} = \left(\frac{1}{b}\right)^{2\epsilon}$$

as well; this limit is strictly less than one, since $b > 1$ and $\epsilon > 0$.

A similar argument can be used to prove this equation in the case $\beta < 0$ as well; simply exchange the logarithmic factors in the upper and lower bounds that are derived near the beginning of the argument (that follows the assumption that $\beta \geq 0$), and establish that both of these bounds have limit $\left(\frac{1}{b}\right)^{2\epsilon}$ as $n \to +\infty$ as well.

Now, it remains only to choose the constant

$$d = \frac{1+\left(\frac{1}{b}\right)^{2\epsilon}}{2} = \left(\frac{1}{b}\right)^{2\epsilon} + \frac{1}{2}\left(1 - \left(\frac{1}{b}\right)^{2\epsilon}\right);$$

then $\left(\frac{1}{b}\right)^{2\epsilon} < d < 1$, and the above limit can be used to establish that

$$\frac{a\hat{f}\left(\lceil \frac{n}{b} \rceil\right)}{\hat{f}(n)} \leq d$$

for all sufficiently large n. Since $\hat{f}(n)$ is an asymptotically positive function, this implies that

$$a\hat{f}\left(\lceil \frac{n}{b} \rceil\right) \leq d\hat{f}(n)$$

whenever n is sufficiently large, as well. That is, the "regularity condition" is satisfied for the recurrence for $U(n)$.

Conclusion, for the Third Case. Since all the required conditions are satisfied, the third case of the original version of the master theorem applies to the recurrence for $U(n)$. It follows that

$$U(n) \in \Theta(\hat{f}(n)) = \Theta(f(n)),$$

so, in particular,

$$U(n) \in O(f(n)),$$

as required. Therefore, $T(n) \in \Theta(f(n))$ (as claimed by the third case for the "simplified" version of the master theorem), as argued above.

8.7.6 Application of the Master Theorems

In order to apply each of the above theorems, given a recurrence, you must confirm that the recurrence has the form that the theorem covers. In the process you should discover the values of the constants a, b, and c, and the function $f(n)$.

You should then try to prove that

$$f(n) \in \Theta(n^{\alpha}(\log_2 n)^{\beta})$$

for constants α and β. If you manage to do this as well then you can apply Theorem 8.4 to conclude that

1. $T(n) \in \Theta(n^{\log_b a})$ if $\alpha < \log_b a$;
2. $T(n) \in \Theta(n^{\log_b a} \log_2 n)$ if $\alpha = \log_b a$ and $\beta = 0$;
3. $T(n) \in \Theta(f(n)) = \Theta(n^{\alpha}(\log_2 n)^{\beta})$ if $\alpha > \log_b a$.

On the other hand, if you *are not* able to confirm that $f(n) \in \Theta(n^{\alpha}(\log_2 n)^{\beta})$ for positive constants α and β, then you should check whether any of the three cases mentioned in the *original version* of the theorem, Theorem 8.3 applies. That is, you should check whether any of the conditions

1. $f(n) \in O(n^{(\log_b a)-\epsilon})$ for a positive constant ϵ,
2. $f(n) \in \Theta(n^{\log_b a})$,
3. $f(n) \in \Omega(n^{(\log_b a)+\epsilon})$ for a positive constant ϵ, and there exists a positive constant $d < 1$ such that

   $$af\left(\lceil \tfrac{n}{b} \rceil\right) \leq df(n)$$

   for all sufficiently large n,

is satisfied for the recurrence you've been given. These three cases are mutually exclusive (this is true for the cases in the simplified version of the theorem, as well as the cases in the original), so that *at most* one case can hold.

If you are forced to use the original version of the theorem, then one way to decide which case is possible is to apply a limit test, that is, to evaluate the limit

$$\lim_{n \to +\infty} \frac{f(n)}{n^{\log_b a}}.$$

If this limit exists and is zero, then $f(n) \in o(n^{\log_b a})$, so that only the *first* case is possible (but, this case might not apply either). If this limit exists and is $+\infty$ then $f(n) \in \omega(n^{\log_b a})$, so that only the *third* case is possible (but this might not apply either). Finally, if the limit exists and is a positive constant then $f(n) \in \Theta(n^{\log_b a})$, in which case the second case definitely *does* hold.

Of course it is possible that the limit doesn't exist at all; then the first and third cases are impossible (because the limit tests for little-oh and little-omega were *necessary* as well as sufficient), so only the second case is possible. However, it *is* possible that you'll be able to prove that $f(n) \in \Theta(n^{\log_b a})$ even though the above limit doesn't exist — and you'll then be able to conclude that the second case holds, immediately after that.

If one of the three cases covered in the master theorem *does* apply, then the theorem states an asymptotic bound in closed form for your recurrence, so you can simply write it down. Otherwise, you must conclude that the master theorem is not applicable and (if necessary) try to find a closed form by other means.

An Example

Suppose, for example, that you've been given an algorithm whose running time satisfies the recurrence

$$T(n) \leq \begin{cases} 7 & \text{if } n \leq 1, \\ T\left(\lfloor \frac{n}{2} \rfloor\right) + 5 & \text{if } n \geq 2. \end{cases}$$

Now, the above recurrence *isn't* in the form you want, since it includes an inequality. However, if you define the function

$$U(n) = \begin{cases} 7 & \text{if } n \leq 1, \\ U\left(\lfloor \frac{n}{2} \rfloor\right) + 5 & \text{if } n \geq 2, \end{cases}$$

then it's *very* easy to prove by induction on n that $T(n) \leq U(n)$ for every integer $n \geq 1$, and this latter recurrence *is* in the desired form.

In particular, it has the form

$$U(n) = \begin{cases} c & \text{if } n < b, \\ aU\left(\lfloor \frac{n}{b} \rfloor\right) + f(n) & \text{if } n \geq b, \end{cases}$$

where $a = 1$, $b = 2$, $c = 7$, and $f(n) = 5$.

Furthermore, $f(n) = 5 \in \Theta(n^{\alpha}(\log_2 n)^{\beta})$, for $\alpha = \beta = 0$, so that the simplified version of the master theorem (Theorem 8.4) can be applied.

Now, $\log_b a = \log_2 1 = 0$, and therefore $n^{\log_b a} = n^0 = 1$. It should be clear, now, that the middle case mentioned in the simplified master theorem applies:

$$\alpha = \log_b a \quad \text{and} \quad \beta = 0,$$

so we can conclude, by this theorem, that

$$U(n) \in \Theta(n^{\log_b a} \log_2 n) = \Theta(\log_2 n).$$

Since $T(n) \leq U(n)$, we can also conclude that

$$T(n) \in O(\log_2 n).$$

We *can't* conclude that $T(n) \in \Theta(\log_2 n)$, because the original recurrence for $T(n)$ (which included an inequality, rather than an equation) is satisfied by functions that grow more slowly than logarithmically in n — including, for example, the constant function $T(n) = 1$.

Another Example

Suppose, instead, that T had been defined by the recurrence

$$T(n) \geq \begin{cases} 7 & \text{if } n \leq 1, \\ T\left(\lfloor \frac{n}{2} \rfloor\right) + 5\lceil \sqrt{n}\, \rceil & \text{if } n \geq 2. \end{cases}$$

Now, you'd consider a function $L(n)$ where, this time,

$$L(n) = \begin{cases} 7 & \text{if } n \leq 1, \\ L\left(\lfloor \frac{n}{2} \rfloor\right) + 5\lceil \sqrt{n}\, \rceil & \text{if } n \geq 2; \end{cases}$$

in this case, it's easy to see that $T(n) \geq L(n)$ for every integer $n \geq 1$ (and to prove it by induction on n).

Once again, the recurrence for $L(n)$ has the desired form, and it corresponds to the choice of constants $a = 1$, $b = 2$, $c = 7$, and $f(n) = 5\lceil \sqrt{n}\, \rceil$.

Now, $f(n) \in \Theta(n^{\alpha}(\log_2 n)^{\beta})$ for $\alpha = \frac{1}{2}$ and $\beta = 0$ so that, once again, the simplified version of the master theorem can be applied.

As before, $\log_b a = 0$, so that $n^{\log_b a} = 1$. Now, the *third case* covered in Theorem 8.4 applies:

$$\alpha > \log_b a.$$

Therefore, we can conclude that $L(n) \in \Theta(f(n)) = \Theta(\sqrt{n}\,)$.

Suppose (in order to obtain an example in which the original version of the theorem is to be used) that you had failed to notice that $f(n)$ was in the form needed for the simplified version of the theorem to be applicable. You'd therefore try to apply the original version of the theorem, Theorem 8.3, instead.

In this case, since $\log_b a = 0$ once again, you can eliminate the first two cases, by noting that $f(n) \notin O(n^{\log_b a})$; only the third case is possible.

Now, the first part of the condition required for the third case is satisfied, since

$$f(n) = 5\lceil \sqrt{n}\, \rceil \in \Omega(n^{\log_b a + \epsilon}) = \Omega(n^{\epsilon}), \quad \text{for } \epsilon = 0.25.$$

Indeed, we could have chosen ϵ to be *any* positive constant that is strictly greater than zero, and less than or equal to one-half, and the above statement would hold.

We must also establish "the regularity condition." Fortunately, this isn't too difficult, for this example: Since $a = 1$ and $b = 2$,

$$\begin{aligned} af\left(\lceil \tfrac{n}{b} \rceil\right) &= f\left(\lceil \tfrac{n}{2} \rceil\right) \\ &= 5\left\lceil \sqrt{\lceil (n/2) \rceil}\, \right\rceil \\ &\leq 5\left\lceil \sqrt{(n+1)/2}\, \right\rceil \\ &\leq 5\sqrt{(n+1)/2} + 5 \\ &\leq 4\sqrt{n} \qquad \text{if } n \text{ is sufficiently large (in particular, if } n \geq 127) \\ &\leq \tfrac{4}{5} \cdot 5\lceil \sqrt{n}\, \rceil \\ &= \tfrac{4}{5} f(n). \end{aligned}$$

That is, you can satisfy the regularity condition by choosing $d = \frac{4}{5}$.

Now we can conclude from the original version of the master theorem (as well as from the simplified version, as argued above) that

$$L(n) \in \Theta(f(n)) = \Theta(\sqrt{n}\,),$$

and, since $T(n) \geq L(n)$ for every integer $n \geq 1$, we can also conclude that

$$T(n) \in \Omega(\sqrt{n}\,).$$

Once again, we can't conclude that $T(n) \in O(\sqrt{n})$ because the original recurrence contains an inequality instead of an equality and is satisfied by functions that aren't in $O(\sqrt{n})$. For example, the function $T(n) = 1000n$ satisfies the original recurrence.

A Third Example

As a final example, suppose that

$$T(n) = \begin{cases} 1 & \text{if } n \leq 1, \\ 2T\left(\lfloor \frac{n}{2} \rfloor\right) + n \log_2 n & \text{if } n \geq 2. \end{cases}$$

In this case, the recurrence has the desired form — it corresponds to the choices $a = b = 2$, $c = 1$, and $f(n) = n \log_2 n$.

The simplified version of the theorem, Theorem 8.4 is applicable — or, at least, it initially seems to be, since $f(n) \in \Theta(n^\alpha (\log_2 n)^\beta)$, for $\alpha = \beta = 1$.

However, $\log_b a = 1$ as well. Now, since $\alpha = \log_b a$, neither the first nor third cases are applicable. Unfortunately the second case is not applicable either, because $\beta \neq 0$.

If we try to use the original version of the master theorem (Theorem 8.3), instead, then we encounter a similar difficulty: Since $\log_b a = \log_2 2 = 1$,

- $f(n) = n \log_2 n \in \omega(n^{(\log_b a)-\epsilon}) = \omega(n^{1-\epsilon})$, for every positive constant ϵ, and therefore $f(n)$ is not in $O(n^{(\log_b a)-\epsilon})$, for *any* positive constant ϵ. Therefore the first case mentioned in the master theorem does not apply (you can also use the limit test to eliminate this case).

- $f(n) = n \log_2 n \in \omega(n^{\log_b a}) = \omega(n)$, so $f(n) \notin \Theta(n^{\log_b a})$, and the second case mentioned in the master theorem does not apply (you could use the limit test to eliminate this case, as well); and

- $f(n) = n \log_2 n \in o(n^{(\log_b a)+\epsilon}) = o(n^{1+\epsilon})$, for every positive constant ϵ, so $f(n)$ is not in $\Omega(n^{(\log_b a)+\epsilon})$ for *any* positive constant ϵ (even though $f(n) \in \Omega(n^{\log_b a})$). Therefore the *third* case mentioned in the master theorem does not apply, either.

So, this is an example such that the master theorem is not applicable at all, and so that it can't be used to find a tight asymptotic bound for the function defined by the recurrence.

Furthermore, it should be noted as well, that *in general,* there is no point in considering the original version of the master theorem, when you have been unable to apply the simplified version because $f(n) \in \Theta(n^\alpha (\log_2 n)^\beta)$ for constants α and β such that $\alpha = \log_b a$ but $\beta \neq 0$: You'll always discover, under these circumstances, that the *original* version of the master theorem isn't applicable, either.

Exercise: Use one or more of the techniques for solving recurrences, introduced in CPSC 413 before this, to prove that $T(n) \in \Theta(n(\log_2 n)^2)$ if $T(n)$ is as defined above.

8.8 Solving Linear Recurrences with Constant Coefficients

This section concerns another special form of recurrence that has applications in algorithm analysis and a way to solve a recurrence of this form; it might not be covered in CPSC 413 if there isn't time for it.

8.8.1 Definition and Examples

Definition 8.5. A *linear recurrence with constant coefficients* is a recurrence

$$T(i) = \begin{cases} b_i & \text{if } 0 \leq i < t, \\ a_1 T(i-1) + a_2 T(i-2) + \cdots + a_t T(i-t) & \text{if } i \geq t. \end{cases} \tag{8.10}$$

for a positive integer t and complex numbers $a_1, a_2, \ldots, a_t, b_0, b_1, \ldots, b_{t-1}$.

It will be assumed throughout this section that $a_t \neq 0$.

Note that $T(i)$ is defined as a *linear* combination of the values $T(i-1), T(i-2), \ldots, T(i-t)$ for a positive constant t, whenever $i \geq t$. The coefficients $a_1, a_2, \ldots, a_t$ in the recursive definition are required to be *constants* as well.

Example 8.6. The recurrence used to define the Fibonacci numbers,

$$F_i = \begin{cases} 0 & \text{if } i = 0, \\ 1 & \text{if } i = 1, \\ F_{i-1} + F_{i-2} & \text{if } i \geq 2, \end{cases}$$

is a linear recurrence with constant coefficients (with $t = 2$, $b_0 = 0$, $b_1 = 1$, and $a_1 = a_2 = 1$).

Example 8.7. The recurrence

$$T(i) = \begin{cases} 1 & \text{if } i = 0, \\ 1 & \text{if } i = 1, \\ 2 & \text{if } i = 2, \\ 7T(i-1) - 16T(i-2) + 12T(i-3) & \text{if } i \geq 3, \end{cases}$$

is a linear recurrence with constant coefficients (with $t = 3$, $b_0 = b_1 = 1$, $b_2 = 2$, $a_1 = 7$, $a_2 = -16$, and $a_3 = 12$).

On the other hand, the recurrences that can be solved using "the master theorem," are examples of recurrences that are *not* "linear recurrences with constant coefficients."

8.8.2 Closed Forms for Solutions of Linear Recurrences with Constant Coefficients

Definition 8.8. The *characteristic polynomial* of the recurrence given in equation (8.10) is the polynomial

$$x^t - a_1 x^{t-1} - a_2 x^{t-2} - \cdots - a_{t-1} x - a_t. \tag{8.11}$$

As you may have seen in an algebra course, every polynomial with complex numbers as coefficients can be factored into a product of *linear* polynomials with complex coefficients. In particular, if

$$f(x) = x^t - a_1 x^{t-1} - a_2 x^{t-2} - \cdots - a_{t-1}x - a_t$$

as above, then there exists some positive integer $s \leq t$, distinct complex numbers $\beta_1, \beta_2, \ldots, \beta_s$, and positive integers $\nu_1, \nu_2, \ldots, \nu_s$ such that

$$\nu_1 + \nu_2 + \cdots + \nu_s = t$$

and

$$f(x) = \prod_{i=1}^{s} (x - \beta_i)^{\nu_i}. \tag{8.12}$$

In this case, we call β_i a *root* of $f(x)$ and say that ν_i is the *multiplicity* of this root.

Note that, since t is chosen so that $a_t \neq 0$, 0 is not a root of the characteristic polynomial of a linear recurrence with constant coefficients.

The following theorem gives a way to find closed forms for functions expressed by linear recurrences with constant coefficients. It won't be proved in CPSC 413, but a more general method that could be *used* to prove it will be mentioned in the next section.

Theorem 8.9. *Suppose the characteristic polynomial (given by equation (8.11)) of the linear recurrence in equation (8.10) has the factorization shown in equation (8.12), above.*

Then there exist polynomials $q_1(x), q_2(x), \ldots, q_s(x)$ such that $q_i(x)$ has degree less than ν_i for $1 \leq i \leq s$ and such that

$$T(n) = q_1(n)\beta_1^n + q_2(n)\beta_2^n + \ldots q_s(n)\beta_s^n \tag{8.13}$$

for every integer $n \geq 0$.

8.8.3 Applying Theorem 8.9

Theorem 8.9 states a closed form for the recurrence in equation (8.10) that involves the roots of the characteristic polynomial of the recurrence. In this closed form, an "unknown" polynomial is multiplied by a power of each root; the degree of this unknown polynomial is at most the multiplicity of the corresponding root (in the characteristic polynomial), minus one.

In order to apply this theorem, to find a closed form for a linear recurrence with constant coefficients, one should

1. Compute and factor the characteristic polynomial of the linear recurrence, in order to discover the roots of this polynomial and their multiplicities;

2. Use the initial values $(T(0), T(1), \ldots, T(t-1))$ in order to solve for the "unknown" polynomials that are multiplied by powers of the roots in the closed form given in equation (8.13).

It will turn out that the number of unknown coefficients that one needs to discover, in order to perform the second step, will always be equal to the number of initial values given for the recurrence (that is, t). Furthermore, one will always be able to find these coefficients by solving a nonsingular

system of t linear equations in t unknowns, so that the second step can always be performed by setting up a system of linear equations and then using Gaussian Elimination to solve it.

Consider, for example, the "Fibonacci" recurrence, given in Example 8.6. Since $t = 2$ and $a_1 = a_2 = 1$, this recurrence has characteristic polynomial

$$f(x) = x^2 - x - 1.$$

This polynomial can be factored using the binomial theorem, to show that

$$f(x) = x^2 - x - 1 = \left(x - \frac{1+\sqrt{5}}{2}\right)\left(x - \frac{1-\sqrt{5}}{2}\right).$$

Thus the polynomial has two roots, $\beta_1 = (1+\sqrt{5})/2$ and $\beta_2 = (1-\sqrt{5})/2$, and each root has multiplicity one. It follows by Theorem 8.9 that the recurrence has a closed form

$$F_n = p_1(n)\beta_1^n + p_2(n)\beta_2^n,$$

where each unknown polynomial $p_1(n)$ and $p_2(n)$ has degree at most $1 - 1 = 0$. Thus, the polynomials are both unknown "constants," and

$$F_n = c_1\beta_1^n + c_2\beta_2^n,$$

where c_1 and c_2 are complex numbers whose values are yet to be determined.

At this point, we can make use of the initial values $F_0 = 0$ and $F_1 = 1$ to obtain two equations in the "unknowns" c_1 and c_2:

$$\begin{aligned} 0 = F_0 &= c_1\beta_1^0 + c_2\beta_2^0 = c_1 + c_2, \\ 1 = F_1 &= c_1\beta_1^1 + c_2\beta_2^1 = \left(\frac{1+\sqrt{5}}{2}\right)c_1 + \left(\frac{1-\sqrt{5}}{2}\right)c_2. \end{aligned}$$

That is,

$$\begin{bmatrix} 1 & 1 \\ \frac{1+\sqrt{5}}{2} & \frac{1-\sqrt{5}}{2} \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}.$$

Solving this system, we find that $c_1 = \frac{1}{\sqrt{5}}$ and $c_2 = -\frac{1}{\sqrt{5}}$, giving us the (known) closed form

$$F_i = \frac{1}{\sqrt{5}}\left(\left(\frac{1+\sqrt{5}}{2}\right)^i - \left(\frac{1-\sqrt{5}}{2}\right)^i\right) \quad \text{for } i \geq 0.$$

Next, consider the recurrence given in example 8.7. In this case $t = 3$, $a_1 = 7$, $a_2 = -16$, and $a_3 = 12$, so the characteristic polynomial of the recurrence is

$$f(x) = x^3 - 7x^2 + 16x - 12 = (x-2)^2(x-3),$$

which has one root $\beta_1 = 2$ with multiplicity two, and another root $\beta_2 = 3$ with multiplicity one. In this case, a solution for the recurrence has a closed form

$$T(n) = p_1(n)\beta_1^n + p_2(n)\beta_2^n,$$

where the unknown polynomial $p_1(n)$ has degree at most $2 - 1 = 1$ and the unknown polynomial $p_2(n)$ has degree at most $1 - 1 = 0$, so that

$$\begin{aligned} T(n) &= (c_1 n + c_2)\beta_1^n + c_3\beta_2^n \\ &= c_1 n 2^n + c_2 2^n + c_3 3^n, \end{aligned}$$

for unknown complex numbers c_1, c_2, and c_3.

Now we use the initial values $T(0) = 1$, $T(1) = 1$, $T(2) = 2$ to obtain three equations in these unknowns,

$$\begin{aligned} c_1 \cdot 0 \cdot 2^0 + c_2 \cdot 2^0 + c_3 \cdot 3^0 &= 1, \\ c_1 \cdot 1 \cdot 2^1 + c_2 \cdot 2^1 + c_3 \cdot 3^1 &= 1, \\ c_1 \cdot 2 \cdot 2^2 + c_2 \cdot 2^2 + c_3 \cdot 3^2 &= 2; \end{aligned}$$

that is,

$$\begin{bmatrix} 0 & 1 & 1 \\ 2 & 2 & 3 \\ 8 & 4 & 9 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 2 \end{bmatrix}.$$

Solving this system, we find that $c_1 = -\frac{3}{2}$, $c_2 = -1$, and $c_3 = 2$, so that

$$T(n) = -\tfrac{3}{2} \cdot n \cdot 2^n - 2^n + 2 \cdot 3^n \quad \text{for } n \geq 0.$$

Exercise (Yet more practice in the use of mathematical induction): Prove that the closed forms obtained in the last two examples are correct.

8.9 An Additional Method

Another powerful method for solving recurrences, which probably won't be discussed in CPSC 413, involves the application of properties of *generating functions* for recurrences. Among other things, this method can be used to *prove* Theorem 8.9.

For more information about generating functions for recurrences, and their use to solve recurrences, see, for example, Graham, Knuth, and Patashnik [4], or Sedgewick and Flajolet [14].

8.10 Exercises

1. Try to use "the iteration method" to express each of the following summations as recurrences.

   You may assume that n is a power of two for the first three parts, and you may assume that n is a power of a power of two (so that $n = 2^{(2^k)}$ for some integer k) for the last two parts of this question.

   (a) $T(1) = 1$ and $T(n) = 4T(\lfloor n/2 \rfloor) + n$ if $n \geq 2$.
   (b) $T(1) = 1$ and $T(n) = 4T(\lceil n/2 \rceil) + n^2$ if $n \geq 2$.
   (c) $T(1) = 1$ and $T(n) = 4T(\lfloor n/2 \rfloor) + n^3$ if $n \geq 2$.
   (d) $T(2) = 1$ and $T(n) = T(\lfloor \sqrt{n} \rfloor) + \log_2 n$ if $n \geq 4$.
   (e) $T(2) = 1$ and $T(n) = \lfloor \sqrt{n} \rfloor T(\lfloor \sqrt{n} \rfloor) + n$ if $n \geq 4$.

2. Now prove that each of the functions $T(n)$ defined by recurrences in the first question is a nondecreasing function (defined on the set of positive integers).

   Use this, and your answer for Question #1, to find a function $g(n)$ in closed form, and constants c_L and c_U, such that

   $$T(n) \in \Theta(g(n)) \quad \text{and} \quad c_L g(n) \leq T(n) \leq c_U g(n) \text{ for all } n \geq 4$$

   for each part of Question #1. Try to make your estimates as tight as you can (just to see how well you can do).

3. For each of the functions $T(n)$ defined in the first three parts of Question #1, give recurrences for the corresponding functions

   $$U(n) = T(n-1) \quad \text{and} \quad V(n) = T(n+1).$$

4. Now try to use the "substitution method" to answer the following question. I recommend that you attempt Question #1 *before* you read this one.

   (a) Show that if $T(n)$ is as defined in Question #1(a) then $T(n) \in \Theta(n^2)$. If time permits then you should try to do this three ways (to see what happens in each case):
      i. Try to show that $T(n) \leq cn^2$ and $T(n) \geq \hat{c}n^2$ for unknown positive constants c and $\hat{c}$. (Note though, that this attempt might not be successful!)
      ii. Consider the function $U(n) = T(n) + \frac{n}{2}$ and try to show that $U(n) \leq cn^2$ and $U(n) \geq \hat{c}n^2$ for unknown positive constants c and $\hat{c}$ instead.
      iii. Try to show that $T(n) \leq cn^2 - dn$ and $T(n) \geq \hat{c}n^2$ for unknown positive constants c and $\hat{c}$ and an unknown constant d.
   (b) Show that if $T(n)$ is as defined in Question #1(b) then $T(n) \in \Theta(n^2 \log_2 n)$.
   (c) Show that if $T(n)$ is as defined in Question #1(c) then $T(n) \in \Theta(n^3)$.
   (d) Show that if $T(n)$ is as defined in Question #1(d) then $T(n) \in \Theta(\log_2 n)$.
   (e) Show that if $T(n)$ is as defined in Question #1(e) then $T(n) \in \Theta(n \log_2 \log_2 n)$.

5. Now try to use the master theorem (Theorems 8.3 and 8.4) to solve the recurrences given in Question #1. This should be reasonably straightforward for the first three parts, but you will probably need to "simplify the recurrence" for the last two parts in order to apply the master theorem successfully.

6. Try to use the master theorem (Theorems 8.3 and 8.4) to solve the following additional recurrences.

   (a) $T(1) = 1$ and $T(n) = 4T(\lfloor n/2 \rfloor) + n \log_2 n$ if $n \geq 2$.
   (b) $T(1) = 1$ and $T(n) = 4T(\lfloor n/2 \rfloor) + n^2 \log_2 n$ if $n \geq 2$.
   (c) $T(1) = 1$ and $T(n) = 4T(\lfloor n/2 \rfloor) + n^5$ if $n \geq 2$.
   (d) $T(1) = T(2) = 1$ and $T(n) = 5T(\lfloor n/3 \rfloor) + n$ if $n \geq 3$.

7. Find closed forms for the functions given by the following linear recurrences with constant coefficients.

   (a) $T(0) = 3$, $T(1) = 4$, and $T(n) = T(n-1) + 2T(n-2)$ for $n \geq 2$.
   (b) $T(0) = 2$, $T(1) = 3$, $T(2) = 4$, and $T(n) = 6T(n-1) - 11T(n-2) + 6T(n-3)$ for $n \geq 3$. Note that 1 is a root of the characteristic polynomial of this linear recurrence.
   (c) $T(0) = T(1) = 1$, $T(2) = 5$, and $T(n) = T(n-1) + T(n-2) - T(n-3)$ if $n \geq 3$. Note that 1 is a root of the characteristic polynomial of this linear recurrence.
   (d) $T(0) = 1$, $T(1) = 1$, $T(2) = 10$, $T(3) = 30$, and

   $$T(n) = 4T(n-1) - 6T(n-2) + 4T(n-3) - T(n-4) \quad \text{if } n \geq 4.$$

   Note that 1 is a root of the characteristic polynomial of this linear recurrence, with multiplicity greater than one.

Hints for selected exercises are given in the next section; solutions for Exercises #1(c), 2(c), 3(c), 4(c), 5(b), 6(b), 6(c), and 7(c) can be found in Section 11.3.1.

8.11 Hints for Selected Exercises

Exercise #1(a): Try to establish the following pattern, which should become evident after you apply the iteration method to the given recurrence: If $h \geq 0$ and n is a power of two such that $n \leq 2^h$, then

$$T(n) = 4^h T(n/2^h) + n \sum_{i=0}^{h-1} 2^i.$$

Exercise #1(b): Try to establish the following pattern, which should become evident after you apply the iteration method to the given recurrence: If $h \geq 0$ and n is a power of two such that $n \leq 2^h$, then

$$T(n) = 4^h T(n/2^h) + hn^2.$$

Exercise #1(c): Try to establish the following pattern, which should become evident after you apply the iteration method to the given recurrence: If $h \geq 0$ and n is a power of two such that $n \leq 2^h$, then

$$T(n) = 4^h T(n/2^h) + n^3 \sum_{i=0}^{h-1} \left(\tfrac{1}{2}\right)^i.$$

Exercise #1(d): Try to establish the following pattern, which should become evident after you apply the iteration method to the given recurrence: If $h \geq 0$ and n is a power of a power of two such that $n \leq 2^{(2^h)}$, then

$$T(n) = T\left(n^{1/2^h}\right) + \log_2 n \sum_{i=0}^{h-1} \left(\tfrac{1}{2}\right)^i.$$

Exercise #1(e): Try to establish the following pattern, which should become evident after you apply the iteration method to the given recurrence: If $h \geq 0$ and n is a power of a power of two such that $n \leq 2^{(2^h)}$, then

$$T(n) = \tfrac{n}{n^{1/2^h}} T\left(n^{1/2^h}\right) + hn.$$

Exercise #5(d): Start by simplifying the recurrence, by performing a "change of variable:" Express the function in terms of $k = \log_2 n$ before you try to apply the master theorem.

Exercise #5(e): Start by simplifying the recurrence, by performing a "change of function:" Consider the function $U(n) = T(n)/n$. You might want to perform a "change of variable," to simplify the recurrence *again,* after that — see the hint for Exercise #5(d), above.

Exercise #7: You'll need to start by computing and factoring the characteristic polynomial of the linear recurrence in each case. The information given in the last few parts implies that the characteristic polynomial is divisible by $x - 1$ (or even $(x - 1)^2$), and this should help you to completely factor the polynomial.

8.12 Sample Tests

This is a long chapter, which leads into the first "algorithm design" topic, and two tests containing recurrence questions were used in both fall 1996 and fall 1997. However, the second of the tests used in fall 1997 was only used as a "practice test," as a warmup for the midterm and final examination, and it didn't count for credit.

Solutions for the following tests can be found in Section 11.3.2.

8.12.1 First Class Test for 1996

Instructions:

Attempt all questions. Write answers on the question sheets.

No aids allowed.

Duration: 50 minutes.

1. (10 marks) Suppose $T(1) = 0$ and

$$T(n) = 2T(n/2) + n \log_2 n$$

for $n \geq 2$. Use the "substitution method" to prove that $T(n) \in O(n(\log_2 n)^2)$ whenever $n = 2^k$ is a power of two.

Hint: Note that $\log_2 n \geq 1$ whenever $n \geq 2$.

2. (10 marks) Now use the "iteration method" to find an expression in closed form that is an exact solution for the recurrence given in Question #1. Once again, you may assume that $n = 2^k$ is a power of two.

Hint: If $n = 2^k$ and k is a nonnegative integer then

$$\sum_{i=0}^{k} \log_2 \left(\frac{n}{2^i}\right) = \sum_{j=0}^{k} \log_2(2^j) = \sum_{j=0}^{k} j.$$

Note: A *small* number of bonus marks will be awarded if you also use mathematical induction to prove that your solution is correct.

8.12.2 Second Class Test for 1996

Instructions:

Attempt all questions. Write answers on the question sheets.

No aids allowed.

Duration: 50 minutes

1. Suppose a recursive algorithm for multiplication of two n-digit integers

   - uses exactly c_1 steps, for some positive constant c_1, if $n < 3$, and
   - solves the problem when $n \geq 3$ by
     (a) multiplying together *five* pairs of smaller integers, using the same algorithm recursively, where the smaller "input" integers are always exactly $\lfloor \frac{n}{3} \rfloor$ digits long, and
     (b) doing extra work that requires $c_2 n$ steps, for some positive constant c_2.

   (a) (4 marks) Write down a recurrence for the number of steps used by the above multiplication algorithm (as a function of n) in the worst case.

   (b) (3 marks) Find a function $f(n)$ in closed form such that this algorithm uses $\Theta(f(n))$ steps to multiply two n-digit integers together.

2. (5 marks) Write a recursive function that takes a pointer to the root of a binary tree as input, and returns the sum of the values stored at the nodes of the tree.

   Use the following notation: if p is a pointer to a node, then

   - $*p$.left is a pointer to the left child (and is "null" if the node does not have a left child);
   - $*p$.right is a pointer to the right child (and is "null" if the node does not have a right child);
   - $*p$.value is the value stored at the node.

3. Consider two "integer division" algorithms (that compute the quotient and remainder obtained by dividing one n-digit input integer by a second n-digit input integer):

   - Algorithm A uses $T(n)$ steps, when given n-digit integers as input, where

     $$T(n) = \begin{cases} c_1 & \text{if } n < 4, \\ 7T(\lceil \frac{n}{4} \rceil) + c_2 n + c_3 & \text{if } n \geq 4, \end{cases}$$

     and where c_1, c_2, and c_3 are positive constants;
   - Algorithm B uses $\Theta(n(\log n)^2)$ steps when given n-digit integers as input.

   (a) (6 marks) Find a function $f(n)$ in closed form such that $T(n) \in \Theta(f(n))$ (where $T(n)$ is the number of steps used by Algorithm A, as above).

(b) (2 marks) Say *which* of the above two algorithms is asymptotically faster — that is, say which algorithm uses a smaller number of steps, when given n-digit integers as inputs, for all sufficiently large n.

The following approximations may be helpful as you try to answer these questions. Each is a slight *under-approximation*, but no approximation is "off" by 0.01 or more. Thus, for example, you should interpret the claim below that $\log_2 3 \cong 1.58$ to mean that $1.58 \leq \log_2 3 < 1.59$.

$\log_2 2 = 1$	$\log_3 2 \cong 0.63$	$\log_4 2 = 0.5$	$\log_5 2 \cong 0.43$	$\log_6 2 \cong 0.38$	$\log_7 2 \cong 0.35$
$\log_2 3 \cong 1.58$	$\log_3 3 = 1$	$\log_4 3 \cong 0.79$	$\log_5 3 \cong 0.68$	$\log_6 3 \cong 0.61$	$\log_7 3 \cong 0.56$
$\log_2 4 = 2$	$\log_3 4 \cong 1.26$	$\log_4 4 = 1$	$\log_5 4 \cong 0.86$	$\log_6 4 \cong 0.77$	$\log_7 4 \cong 0.71$
$\log_2 5 \cong 2.32$	$\log_3 5 \cong 1.46$	$\log_4 5 \cong 1.16$	$\log_5 5 = 1$	$\log_6 5 \cong 0.89$	$\log_7 5 \cong 0.82$
$\log_2 6 \cong 2.58$	$\log_3 6 \cong 1.63$	$\log_4 6 \cong 1.29$	$\log_5 6 \cong 1.11$	$\log_6 6 = 1$	$\log_7 6 \cong 0.92$
$\log_2 7 \cong 2.80$	$\log_3 7 \cong 1.77$	$\log_4 7 \cong 1.40$	$\log_5 7 \cong 1.20$	$\log_6 7 \cong 1.08$	$\log_7 7 = 1$
$\log_2 8 = 3$	$\log_3 8 \cong 1.89$	$\log_4 8 = 1.5$	$\log_5 8 \cong 1.29$	$\log_6 8 \cong 1.16$	$\log_7 8 \cong 1.06$

8.12.3 First Class Test for 1997

Instructions:

Attempt all questions. Write answers on the question sheets.

No aids allowed.

Duration: 50 minutes

1. This question concerns the definition of *recurrences*.

   (a) (1 mark) State the definition of a *recurrence.*

   (b) (2 marks) Is the following a recurrence? Why, or why not?

$$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ \frac{1}{2}\left(T(n+1) + T(n-1)\right) & \text{if } n \geq 2. \end{cases}$$

   Be specific in your answer for the last part: It isn't enough just to say that it does, or does not, satisfy the definition you gave above.

2. Suppose now that

$$T(n) = \begin{cases} 0 & \text{if } n = 1, \\ 9T\left(\lfloor \frac{n}{3} \rfloor\right) + n^4 & \text{if } n > 1. \end{cases}$$

   (a) (8 marks) Use the substitution method — **not** the master theorem — to prove that

$$T(n) \in O(n^4)$$

   assuming that n is a power of three (so that $n = 3^h$ for some integer $h \geq 0$).

   (b) (2 marks) Now show that

$$T(n) \in \Omega(n^4)$$

   as well, assuming that $T(n) \geq 0$ for every integer $n \geq 1$.
   As the marks for this part should suggest, this should be *much* easier to do than the first part.

3. Consider, once again, the recurrence used in Question #2 and the function $T(n)$ that it defines.

   (a) (5 marks) If you used the "iteration method" then you might make the following conjecture.

   Claim: If $k \geq 0$ and n is any power of three such that $n \geq 3^k$, then

$$T(n) = 9^k T\left(\frac{n}{3^k}\right) + n^4 \sum_{i=0}^{k-1} \left(\frac{1}{9}\right)^i.$$

   Prove that this claim is correct.

(b) (3 marks) Now, use the claim to find an exact closed form for $T(n)$ when n is a power of three.

4. (4 marks) Perform a "change of variable" to write a recurrence for the function $U(h) = T(n)$, where $h = \log_3 n$ (and n is a power of three, so that h is an integer), and T is as defined in Question #2. For your convenience, here is the recurrence for T once again:

$$T(n) = \begin{cases} 0 & \text{if } n = 1, \\ 9T\left(\lfloor \frac{n}{3} \rfloor\right) + n^4 & \text{if } n > 1, \end{cases}$$

8.12.4 Second Class Test for 1997

Instructions:

Attempt all questions. Write answers in the spaces provided below.

Total Marks Available: 25

No Aids Allowed

Duration: 50 minutes

Note: This will not be marked or count toward your CPSC 413 grade.

You'll need to use the following **definitions** when answering questions on this test.

A *binary heap*, of size n, is a binary tree with n nodes that is as complete as possible. The tree is filled on all levels except possibly the lowest, which is filled from the left to a point. This means that every binary heap of size n has the same shape as any other binary heap of size n.

However, a binary heap is (usually) not a binary search tree, because it *isn't* true that the value stored at a node in a heap is always greater than or equal to the value stored at the node's left child and less than or equal to the value stored at the node's right child.

Instead, a *heap order* property is always satisfied: The value stored at a node is always greater than or equal to *all of* the values stored at the children of that node. Therefore, the largest value stored in the heap is always located at the root.

An *almost heap* of size n is a binary tree with n nodes that has the same *shape* as a heap of size n. An "almost heap" might not *be* a heap, because the heap order property might not be satisfied — but you'll be able to turn an "almost heap" into a heap by rearranging the values that are stored in the "almost heap's" nodes.

Now, here are some **facts** that you may use without proving them. Note that it *is not* necessary to see the procedure *setRoot* (which these refer to) in order to use them.

- If the left and right subtrees of the root of an "almost heap" are *heaps* (and not just "almost heaps") and you call the recursive function *setRoot*, with a pointer to the root of this "almost heap" as input, then the "almost heap" will be turned into a heap when this procedure terminates.
- If $n \leq 3$ then the left and right subtrees of any "almost heap" of size n are *heaps* (and not just "almost heaps").

1. *(10 marks)* Write pseudocode for a recursive procedure *buildHeap* which takes a pointer to an "almost heap" as input and turns this "almost heap" into a heap.

   Your procedure should work from "the bottom up" in the sense that it should recursively turn the left and right subtrees of the root into heaps before it does anything else (unless, of course, the input heap has size zero).

   Your procedure can, and should, use the procedure *setRoot* as a subroutine.

   If p is a pointer to a node, please use *p.left* and *p.right* to refer to pointers to the left and right children of that node, respectively.

Note: If you think about, and you've read the instructions, this is *easy,* and your pseudocode should fit, comfortably, into the following space — with room to spare!

2. *(5 marks)* Write a recurrence for the running time $T(n)$ used by your procedure when it is given a pointer to an "almost heap" of size n as input. You should use the unit cost criterion and you should **assume** that $n = 2^k - 1$ for an integer k (that is, your recurrence only needs to be correct for integers n with this form).

   The following additional facts will be useful (and "almost true,") and you may use them without proving them:

   - If $k > 1$ and $n = 2^k - 1$ then the left and right subtrees of the root (of an "almost heap" of size n) are both "almost heaps" with size $2^{k-1} - 1 = \lfloor \frac{n}{2} \rfloor$.
   - When you call *setRoot* with a pointer to an "almost heap" of size m as input for $m \geq 1$, it uses exactly $c_1 \lceil \log_2 m \rceil + c_0$ steps before it terminates, for some positive constants c_1 and c_0 (and, it uses c_0 steps if $m = 0$).

3. *(5 marks)* Use the master theorem to prove that $T(n) \in \Theta(n)$ if $T(n)$ satisfies the recurrence you gave to answer the previous question.

4. *(5 marks)* Finally, use the result you were supposed to prove in the previous question (whether you managed to prove it or not!), and the **assumption** that the running time used by your procedure is a nondecreasing function of n, to prove that this running time is linear in n **in general**, and not just in the special case that $n = 2^k - 1$ for some integer k.

Chapter 9

Sample Midterm Tests

The midterm tests for fall 1996 and fall 1997 examined the "algorithm analysis" material included in CPSC 413 and are given below.

Solutions for these tests can be found in Section 11.4.

9.1 Midterm Test for 1996

Instructions:

Attempt all questions. Write answers on the question sheets in the space provided.

No aids allowed.

Duration: 75 minutes

1. *(5 marks — 1 mark for each part)* Say whether each of the following claims about functions of n is **true** or **false**. You do not need to prove your answers.

   (i) $\left(\frac{3}{2}\right)^n \in \Theta(2^n)$ **Answer:** _____

   (ii) $n \log_2 n \in \omega(n)$ **Answer:** _____

   (iii) $\frac{1}{1000} n^{1/10} \in O((\log_2 n)^{100})$ **Answer:** _____

   (iv) $\frac{1}{1000} n^{1/10} \in O(n^{100})$ **Answer:** _____

   (v) $n^2 + 2n \in \Theta(2n^2 + \sqrt{n})$ **Answer:** _____

2. *(10 marks)* Let $f(n) = e^n$ and let $g(n) = n$ for $n \geq 0$. Prove that $f(n) \in \omega(g(n))$.

3. *(10 marks)* Let $S_n = \sum_{i=1}^{n} 13i^{5.5} = \sum_{i=1}^{n} 13i^{11/2}$.

   Find a function $f(n)$ in closed form such that $|f(n) - S_n| \in O(n^6)$, and prove that this inequality holds for your choice of the function f.

   Generous part marks will be given if you only find a function $f(n)$ in closed form such that $S_n \in \Theta(f(n))$ (and prove that *this* relationship holds), instead.

   You may use the following facts without proving them:

   (a) If $g(x) = 13x^{5.5}$, then $g(x)$ is an *increasing* function of x (over the positive real numbers).
   (b) If $r > 1$ then $((n+1)^r - n^r) \in O(n^{r-1})$.

4. *(5 marks)* Find a function $f(n)$ in closed form such that $T(n) \in \Theta(f(n))$, where

$$T(n) = \begin{cases} 1 & \text{if } n = 1, \\ 16T\left(\lceil n/2 \rceil\right) + n^3 & \text{if } n \geq 2, \end{cases}$$

   and say **briefly** how you obtained your answer.

5. *(10 marks)* Find a closed form for the function $T(n)$, where

$$T(n) = \begin{cases} 0 & \text{if } n = 0, \\ 1 & \text{if } n = 1, \\ 2T(n-1) - T(n-2) & \text{if } n \geq 2, \end{cases}$$

   and prove that your answer is correct. You may prove that your answer is correct either by applying one of the theorems stated in class or by using mathematical induction.

9.2 Midterm Test for 1997

Instructions:

Attempt all questions. Write answers on the question sheets in the space provided.

No aids allowed.

Duration: 80 minutes

1. *(5 marks — 1 mark for each part)* Say whether each of the following claims about functions of n is **true** or **false**. You do not need to prove your answers.

   (i) $2^n \in \omega(n^{10})$ **Answer:** _____

   (ii) $\frac{1}{1000}(\log n)^{10} \in O(n^{1/100})$ **Answer:** _____

   (iii) $n \log_2 n \in \Omega(n)$ **Answer:** _____

   (iv) $n^3 + 2n^2 \in \Theta(2n^3 + \sqrt{n})$ **Answer:** _____

   (v) $1000n^{1/10} \in \Omega(n^{100})$ **Answer:** _____

2. *(8 marks)* Let $f(n) = 3n^2 + \ln n$ and let $g(n) = n^2$ for $n \geq 1$. Prove that $f(n) \in \Theta(g(n))$.

3. Suppose $n \geq 1$, $S_n = \sum_{i=1}^{n} g(i)$, where the function $g(x) = \frac{x}{\ln(2x)}$ is a nondecreasing function of x — you may use this fact without proving it in this question.

   (a) *(9 marks)* Find a function $f(n)$ in closed form such that $S_n \in \Theta(f(n))$, and prove that this holds **without** using approximation by integrals.

   (b) *(3 marks)* Now suppose $G(n)$ is a function of n such that $G'(n) = g(n)$. Use approximation by integrals to give a **lower bound** for S_n (that is, a function that is always less than or equal to S_n). Your answer should be an expression that involves $G(n)$.

4. *(5 marks)* Find a function $f(n)$ in closed form such that $T(n) \in \Theta(f(n))$, where

$$T(n) = \begin{cases} 1 & \text{if } n < 4, \\ 16T\left(\lceil n/4 \rceil\right) + 2n^2 & \text{if } n \geq 4, \end{cases}$$

   and give a **brief** proof that your answer is correct.

5. *(10 marks)* Prove that $T(n) \in O(4^n)$, where

$$T(n) = \begin{cases} 1 & \text{if } n = 0, \\ 8 & \text{if } n = 1, \\ T(n-1) + 12T(n-2) & \text{if } n \geq 2. \end{cases}$$

Hint: You should consider both the cases $n = 0$ and $n = 1$ as special cases.

Part IV

Solutions for Selected Exercises and Tests

Chapter 10

Solutions for Math Review Exercises and Tests

10.1 Proof Techniques (Review of Math 271)

10.1.1 Solution for Exercise #2 in Section 2.5

A First Solution

The first solution will depend on a fact from number theory: Since 6 is the lowest common multiple of 2 and 3, any integer k is divisible by 6 if and only if k is divisible by both 2 and 3.

Therefore, it will be sufficient to prove both that $n^3 - n$ is divisible by 2 for every integer $n \geq 0$, and that $n^3 - n$ is divisible by 3 for every integer $n \geq 0$, in order to conclude that $n^3 - n$ is always divisible by 6.

Proof that $n^3 - n$ is Always Even It is possible to prove that $n^3 - n$ is divisible by 2 (that is, that this number is even) for every integer $n \geq 0$ by observing that either n is even or n is odd, and performing a simple case analysis.

In the case that n is even, so is n^3. It follows that $n^3 - n$ is even as well, because the difference between two even numbers is always an even number.

In the case that n is odd, so is n^3. It follows that $n^3 - n$ is even, because the difference between two *odd* numbers is always even, as well.

Thus $n^3 - n$ is even in both cases, so it is even for all $n \geq 0$ (in fact for every *integer* n), as desired.

Proof that $n^3 - n$ is Always Divisible by Three The fact that $n^3 - n$ is divisible by 3 for all $n \geq 0$ will be proved by induction on n.

Basis ($n = 0$):

It is necessary to prove that $0^3 - 0$ is divisible by 3. This is clear, since $0^3 - 0 = 0 = 3 \cdot 0$.

Inductive Step:

Now let n be an arbitrarily chosen integer, and suppose (for now) that $n^3 - n$ is divisible by 3. It is necessary to prove under these assumptions that $(n+1)^3 - (n+1)$ is divisible by 3 as well.

Since $n^3 - n$ is divisible by 3, $(n+1)^3 - (n-1)$ is divisible by 3 if and only if the difference between these quantities,

$$[(n+1)^3 - (n+1)] - [n^3 - n]$$

is divisible by 3.

Now,

$$\begin{aligned}[(n+1)^3-(n+1)]-[n^3-n] &= [n^3+3n^2+3n+1-n-1]-[n^3-n] \\ &= (n^3-n^3)+3n^2+(3n-n+n)+(1-1) \\ &= 3(n^2+n),\end{aligned}$$

which is clearly divisible by 3 (since n^2+n is an integer).

Thus, $[(n+1)^3-(n+1)]$ is divisible by 3 if n^3-n is. Since n was arbitrarily chosen it follows by universal generalization that this is true for *every* nonnegative integer n, as is needed to complete the inductive step.

It now follows by induction on n that n^3-n is divisible by 3 for all $n \geq 0$.

Conclusion of Proof Since n^3-n is divisible by both 2 and 3 for all $n \geq 0$, and 6 is the lowest common multiple of 2 and 3, it follows that n^3-n is divisible by 6 for all $n \geq 0$, as well.

A Second Solution

This second proof avoids the use of the "fact from number theory" introduced at the beginning of the first, and is a proof of the claim directly by induction, instead.

Basis ($n = 0$):

It is necessary to prove that 0^3-0 is divisible by 6. This is clear, since $0^3-0 = 0 = 6 \cdot 0$.

Inductive Step:

Let n be an arbitrarily chosen integer that is greater than or equal to 0, and suppose that n^3-n is divisible by 6.

Since n^3-n is divisible by 6, $(n+1)^3-(n+1)$ is divisible by 6 as well if and only if the difference

$$[(n+1)^3-(n+1)]-[n^3-n]$$

is divisible by 6 too. As shown above, in the inductive step for the first proof, this difference is

$$3(n^2+n) = 6\left(\frac{n^2+n}{2}\right)$$

so it is necessary and sufficient to show that $(n^2+n)/2$ is an integer.

As in the beginning of the first proof, we now observe that either n is even or n is odd, and consider each case.

If n is even then so is n^2, so the sum n^2+n is even (since the sum of two even numbers is always even), and $(n^2+n)/2$ is an integer.

If n is odd then so is n^2, so once again the sum n^2+n is even (since the sum of two odd numbers is always even), and $(n^2+n)/2$ is an integer in this case, too.

Thus, $(n^2+n)/2$, implying that $[(n+1)^3-(n+1)]$ is divisible by 6, as desired.

Since the only assumption made to establish this was that n^3-n is divisible by 6, it follows that "if n^3-n is divisible by 6, then $(n+1)^3-(n+1)$ is divisible by 6."

Now, since n was arbitrarily chosen, it follows that if n^3-n is divisible by 6 then $(n+1)^3-(n+1)$ is divisible by 6 for *every* nonnegative integer n, as is needed to complete the inductive step.

Thus it follows by induction on n that n^3-n is divisible by 6 for all $n \geq 0$.

10.2 Limits and Derivatives (Review of Math 249 or 251)

10.2.1 Solution for Exercise #1(c) in Section 3.6

This limit has the form

$$\lim_{x\to 0}\frac{f(x)}{g(x)}$$

for differentiable functions $f(x) = x$ and $g(x) = \cos x - 1$, such that $\lim_{x\to 0} f(x) = \lim_{x\to 0} g(x) = 0$. L'Hôpital's rule can be used in this case (and there's no obvious common factor of $f(x)$ and $g(x)$ to be cancelled out first).

In this case $f'(x) = 1$ and $g'(x) = -\sin x$, so, by l'Hôpital's rule,

$$\lim_{x\to 0}\frac{f(x)}{g(x)} = \lim_{x\to 0}\frac{f'(x)}{g'(x)} = \lim_{x\to 0} -\frac{1}{\sin x}.$$

However, if x is small and positive then so is $\sin x$, so that $-\frac{1}{\sin x}$ is large and *negative*, and the limit approaches $-\infty$ as x approaches zero "from above" (or, "from the right"). It's easily argued (in pretty much the same way) that the limit approaches $+\infty$ as x approaches zero "from below" (or, "from the left").

Therefore $-\frac{1}{\sin x}$ doesn't *have* a limit as x approaches zero, so the limit you were asked to try to compute in this question doesn't exist, either.

10.2.2 Solution for Exercise #3 in Section 3.6

The desired identity can be derived by starting with the definition of derivative. If $h(x) = \frac{f(x)}{g(x)}$ then

$$\begin{aligned}
h'(a) &= \lim_{\delta\to 0}\frac{h(a+\delta)-h(a)}{\delta} && \text{by the definition of derivative}\\
&= \lim_{\delta\to 0}\frac{\frac{f(a+\delta)}{g(a+\delta)} - \frac{f(a)}{g(a)}}{\delta} && \text{by the definition of } h\\
&= \lim_{\delta\to 0}\frac{f(a+\delta)g(a) - f(a)g(a+\delta)}{g(a)g(a+\delta)\delta}\\
&= \lim_{\delta\to 0}\frac{f(a+\delta)g(a) - f(a)g(a) + f(a)g(a) - f(a)g(a+\delta)}{g(a)g(a+\delta)\delta}\\
&= \lim_{\delta\to 0}\frac{\left(\frac{f(a+\delta)-f(a)}{\delta}\right)g(a) - f(a)\left(\frac{g(a+\delta)-g(a)}{\delta}\right)}{g(a)g(a+\delta)}.
\end{aligned}$$

Now, since g is differentiable at a, g is also continuous at a. Thus $\lim_{\delta\to 0} g(a+\delta) = g(a)$, and it was given in the question that this value is nonzero. It follows by the sum, product, and quotient rules for limits given in the online lecture notes that the last expression in the above derivation is equal to

$$\frac{\left(\lim_{\delta\to 0}\frac{f(a+\delta)-f(a)}{\delta}\right)\cdot\left(\lim_{\delta\to 0} g(a)\right) - \left(\lim_{\delta\to 0} f(a)\right)\cdot\left(\lim_{\delta\to 0}\frac{g(a+\delta)-g(a)}{\delta}\right)}{\left(\lim_{\delta\to 0} g(a)\right)\cdot\left(\lim_{\delta\to 0} g(a+\delta)\right)},$$

since the limits given in the above expression all exist and the ones in the denominator are nonzero.

Now, by the definition of derivative,

$$\lim_{\delta\to 0}\frac{f(a+\delta)-f(a)}{\delta}=f'(a)\quad\text{and}\quad\lim_{\delta\to 0}\frac{g(a+\delta)-g(a)}{\delta}=g'(a),$$

$\lim_{\delta\to 0} g(a+\delta)=g(a)$ since g is continuous at a, and

$$\lim_{\delta\to 0} f(a)=f(a)\quad\text{and}\quad\lim_{\delta\to 0} g(a)=g(a),$$

since $f(a)$ and $g(a)$ are independent of δ. Thus the above expression is equal to

$$\frac{f'(a)g(a)-f(a)g'(a)}{(g(a))^2}$$

as desired.

10.3 Antiderivatives and Integrals (Review of Math 249 or 251, Continued)

10.3.1 Solution for Exercise #6 in Section 4.4

In order to find an antiderivative for

$$h(x)=\frac{(\ln x)^3}{x},$$

we will try to use "integration by substitution," writing $h(x)$ as

$$h(x)=f(G(x))\cdot g(x)$$

where $g(x)=G'(x)$. Since $h(x)$ involves a natural logarithm, and we know that $\int \frac{1}{x}\,dx=(\ln x)+C$, we will "guess" that $G(x)=\ln x$ so that $g(x)=\frac{1}{x}$.

Then

$$f(G(x))=\frac{h(x)}{g(x)}=(\ln x)^3=(G(x))^3,$$

so that $f(y)=y^3$.

In this case it is easy to find an antiderivative of f: We can choose $F(x)=\frac{1}{4}x^4+C$ (for any constant C), and we can conclude using integration by substitution that

$$\int h(x)\,dx=F(G(x))=\frac{1}{4}(\ln x)^4+C.$$

If we set $H(x)=\frac{1}{4}(\ln x)^4+C$ then it's easy checked (by differentiation) that

$$H'(x)=\frac{4}{4}(\ln x)^3\cdot\frac{1}{x}=h(x)$$

so that $H(x)$ really is an antiderivative of $h(x)$, as desired.

Now it follows by the Fundamental Theorem of Calculus that

$$\begin{aligned}\int_a^b\frac{(\ln x)^3}{x}\,dx&=\int_a^b h(x)\,dx\\&=H(b)-H(a)\\&=\frac{1}{4}(\ln b)^4-\frac{1}{4}(\ln a)^4.\end{aligned}$$

10.4 Solutions for Sample Tests

10.4.1 Solution for Sample Test in Section 5.1

1. Consider two algorithms, "Algorithm A" and "Algorithm B," for the same problem. Algorithm A uses $4n^2$ steps to solve a problem of size n while Algorithm B uses $100n$ steps to solve a problem of the same size.

   (a) (2 marks) For which inputs sizes (values of n) would Algorithm A be at least as fast as Algorithm B?

   Solution: Algorithm A is at least as fast as Algorithm B for input size n, for every natural number n such that $4n^2 \leq 100n$. For positive n, this inequality holds if and only if $n \leq 25$ (and the inequality clearly holds if $n = 0$ as well).

   Thus, Algorithm A is at least as fast as Algorithm B for input size n whenever $n \leq 25$.

   (b) (3 marks) Suppose Algorithm A can be used to solve a problem of size s in time t on one computer. What is the size of a problem that could be solved using the same algorithm, on a new computer that is 100 times as fast?

   Solution: Since $m = 4n^2$ steps are needed by the algorithm to solve a problem of size n, $\frac{1}{4}m = n^2$, and $n = \sqrt{\frac{1}{4}m} = \frac{1}{2}\sqrt{m}$ is the size of a problem that can be solved using m steps (assuming $\frac{1}{2}\sqrt{m}$ is an integer).

   Algorithm A uses $4s^2$ steps to solve a problem of size s on the original machine.

   Since the new machine is 100 times as fast, at least $400s^2$ steps can be performed using the new machine in the same amount of time. As argued above, a problem of size $\frac{1}{2}\sqrt{400s^2}$ can be solved using the algorithm with this number of steps, and $\frac{1}{2}\sqrt{400s^2} = 10s$.

   Thus, the faster machine can be used to solve a problem that is ten times as large, in the same amount of time.

   Note: A one-line solution giving the correct answer (without the derivation) will receive full marks for this question!

2. (5 marks) Consider the sequence of values $h_0, h_1, h_2, \ldots$, defined such that $h_0 = 2$, $h_1 = 5$, and $h_{n+2} = 5h_{n+1} - 6h_n$ for $n \geq 0$. Prove that $h_n = 2^n + 3^n$ for all $n \geq 0$.

   Solution: This can be proved using induction on n.

   Basis: Since $2^0 + 3^0 = 2 = h_0$ and $2^1 + 3^1 = 5 = h_1$, it is clear that the claim is correct for $n = 0$ and for $n = 1$.

   Inductive Step: Suppose now that $n \geq 0$ and that $h_n = 2^n + 3^n$ and $h_{n+1} = 2^{n+1} + 3^{n+1}$. Then

   $$\begin{aligned}
   h_{n+2} &= 5h_{n+1} - 6h_n && \text{(by the definition of } h_{n+2}) \\
   &= 5(2^{n+1} + 3^{n+1}) - 6(2^n + 3^n) && \text{(by the inductive hypothesis)} \\
   &= 2^n(5 \cdot 2 - 6) + 3^n(5 \cdot 3 - 6) && \text{(reordering terms)} \\
   &= 2^n(4) + 3^n(9) \\
   &= 2^{n+2} + 3^{n+2}, && \text{as required.}
   \end{aligned}$$

Since n was arbitrarily chosen, this implies that if $h_n = 2^n + 3^n$ and $h_{n+1} = 2^{n+1} + 3^{n+1}$ then $h_{n+2} = 2^{n+2} + 3^{n+2}$ for *every* integer $n \geq 0$, as is required to complete the inductive step.

It follows by induction on n that $h_n = 2^n + 3^n$ for every natural number n.

3. (10 marks) Recall that every node in a binary tree is either an *internal node* (if it has either one or two children) or a *leaf* (if it does not have any children).

   Prove that if a binary tree has exactly l leaves and i internal nodes, then $l \leq i + 1$.

   Solution: This result can be proved using induction on the size of the tree (that is, the number of nodes it contains).

   Basis: A tree with zero nodes has $l = 0$ leaves and $i = 0$ internal nodes. Clearly, $l \leq i + 1$ in this case.

   A tree with one node has $l = 1$ leaf and $i = 0$ nodes, and it is clear that $l \leq i + 1$ in this case as well.

   Inductive Step: Suppose $n > 1$ and that the claim is correct for all binary trees with (strictly) fewer than n nodes.

   Let T be a binary tree of size n, let T_L be the left subtree of the root of T, and let T_R be the right subtree of the root of T (each of T_L or T_R may be empty).

   If T_L has size n_L and T_R has size n_R then $n_L \geq 0$, $n_R \geq 0$, and $n_L + n_R + 1 = n$. If follows that $n_L \leq n - 1$ and that $n_R \leq n - 1$.

   Therefore, it follows by the inductive hypothesis that if T_L has l_L leaves and i_L internal nodes, then $l_L \leq i_L + 1$, and it also follows by the inductive hypothesis that if T_R has l_R leaves and i_R internal nodes then $l_R \leq i_R + 1$.

   Since T has size strictly greater than one, the root of T is not a leaf. Therefore, T has $l = l_L + l_R$ leaves, and $i = i_L + i_R + 1$ internal nodes (since the root of T is not a node in either T_L or T_R).

   It follows that

$$\begin{aligned} l &= l_L + l_R \\ &\leq (i_L + 1) + (i_R + 1) \\ &= (i_L + i_R + 1) + 1 \\ &= i + 1, \end{aligned}$$

   as required (for T). Since T was an "arbitrarily chosen" binary tree of size n, the desired inequality holds for *all* binary trees of size n.

   Therefore, it follows by induction on n that the claim is correct.

10.4.2 Solution for Sample Test in Section 5.2

1. (4 marks) Compute $\lim_{x\to+\infty} \frac{x \ln x}{x^{1.5}}$.

   Solution #1: In this first solution a common factor (x, which is nonzero as $x \to +\infty$) will be eliminated from the numerator and denominator, and then l'Hôpital's rule will be used to complete the derivation.

$$\begin{aligned}
\lim_{x\to+\infty} \frac{x \ln x}{x^{1.5}} &= \lim_{x\to+\infty} \frac{\ln x}{x^{0.5}} && \text{(eliminating common factor } x\text{)} \\
&= \lim_{x\to+\infty} \frac{(1/x)}{0.5x^{-0.5}} && \text{(by l'Hôpital's rule)} \\
&= \lim_{x\to+\infty} \frac{1}{0.5x^{0.5}} = 0.
\end{aligned}$$

   Solution #2: In this solution the common factor won't be eliminated. Instead, l'Hôpital's rule will be used twice, in order to obtain a limit that can be computed.

$$\begin{aligned}
\lim_{x\to+\infty} \frac{x \ln x}{x^{1.5}} &= \lim_{x\to+\infty} \frac{1 + \ln x}{1.5x^{0.5}} && \text{(by l'Hôpital's rule)} \\
&= \lim_{x\to+\infty} \frac{(1/x)}{0.75x^{-0.5}} && \text{(applying l'Hôpital's rule again)} \\
&= \lim_{x\to+\infty} \frac{1}{0.75x^{0.5}} = 0.
\end{aligned}$$

   Note: In all the cases above (in both proofs), l'Hôpital's rule can be applied when it is, because the numerator and denominator of the expression to which it's applied both approach $+\infty$ as $x \to +\infty$.

2. Compute the derivative $f'(x)$ for each of the following functions f.

   (a) (3 marks) $f(x) = \ln(\ln x)$

   Solution: By the chain rule,

$$f'(x) = \frac{1}{\ln x} \cdot \frac{1}{x} = \frac{1}{x \ln x}.$$

   (b) (3 marks) $f(x) = \frac{e^x}{x \ln x}$

   Solution: This derivative can be computed by applying the quotient rule, and then applying the product rule to differentiate the denominator:

$$f'(x) = \frac{e^x(x\ln x) - e^x(1+\ln x)}{(x\ln x)^2}$$
$$= \frac{xe^x \ln x - e^x \ln x - e^x}{x^2(\ln x)^2}.$$

3. (5 marks) Compute $\int x^2 \ln x\, dx$, and use differentiation to check that your answer is correct.
Hint: $x^2 \ln x = f'(x) \cdot g(x)$, for $f'(x) = x^2$ and $g(x) = \ln x$. Consider "integration by parts."

Solution: Applying integration by parts, using $f(x) = \frac{1}{3}x^3$ and $g(x) = \ln x$ as in the hint,

$$\begin{aligned}\int x^2 \ln x\, dx &= \int f'(x) \cdot g(x)\, dx \\ &= f(x) \cdot g(x) - \int f(x) \cdot g'(x)\, dx \qquad \text{(using integration by parts)} \\ &= \frac{1}{3}x^3 \ln x - \int \frac{1}{3}x^3 \cdot \frac{1}{x}\, dx \\ &= \frac{1}{3}x^3 \ln x - \frac{1}{3}\int x^2\, dx \\ &= \frac{1}{3}x^3 \ln x - \frac{1}{9}x^3 + C\end{aligned}$$

for some (any) constant C.
Check by Differentiation: If $g(x) = \frac{1}{3}x^3 \ln x - \frac{1}{9}x^3 + C$, then

$$g'(x) = \frac{1}{3}\left(x^2 \ln x + \frac{x^3}{x}\right) - \frac{3}{9}x^2 + 0 = x^2 \ln x,$$

as desired.

4. (10 marks) Suppose

$$b(i,j) = \begin{cases} 0 & \text{if } j < 0 \text{ or } j > i, \\ 1 & \text{if } j = 0 \text{ or } j = i, \\ b(i-1, j-1) + b(i-1, j) & \text{if } 1 \le j \le i-1. \end{cases}$$

Prove that $b(i,j) = \frac{i!}{j!(i-j)!}$ for all $i, j \ge 0$ such that $0 \le j \le i$.
Hint: Consider induction on i; $0! = 1$ and $k! = \prod_{i=1}^{k} i = 1 \cdot 2 \cdot 3 \cdots k$ if $k \ge 1$.

Solution: We will use induction on i to prove that, for every integer $i \ge 0$,

$$b(i,j) = \frac{i!}{j!(i-j)!} \quad \text{for every integer } j \text{ such that } 0 \le j \le i.$$

Basis ($i = 0$): In this case there is only one integer j for which the identity must be checked, namely, $j = 0$. If $i = j = 0$ then $b(i, j) = b(0, 0) = 1$ by definition, while

$$\frac{i!}{j!(i-j)!} = \frac{0!}{0!0!} = \frac{1}{1} = 1$$

as well, so that the claim is correct when $i = 0$.

Inductive Step: Suppose now that $i \geq 0$ and

$$b(i, j) = \frac{i!}{j!(i-j)!} \quad \text{for every integer } j \text{ such that } 0 \leq j \leq i.$$

It is necessary and sufficient to prove, under this assumption, that

$$b(i+1, j) = \frac{(i+1)!}{j!(i+1-j)!} \quad \text{for every integer } j \text{ such that } 0 \leq j \leq i+1.$$

Suppose, then, that j is an integer such that $0 \leq j \leq i + 1$; then either

(a) $j = 0$,
(b) $j = i + 1$, or
(c) $1 \leq j \leq (i + 1) - 1$.

Each of these three cases will be considered separately.

First Case ($j = 0$): In this case $b(i + 1, j) = b(i + 1, 0) = 1$ by definition, while

$$\frac{(i+1)!}{j!(i+1-j)!} = \frac{(i+1)!}{1 \cdot (i+1)!} = 1$$

as well, so that the identity is correct in this case.

Second Case ($j = i + 1$): In this similar case, $b(i + 1, j) = b(i + 1, i + 1) = 1$ by definition, while (again)

$$\frac{(i+1)!}{j!(i+1-j)!} = \frac{(i+1)!}{(i+1)! \cdot 1} = 1$$

as well, so that the identity is correct in this case too.

Third Case ($1 \leq j \leq (i+1)-1$): In this final case,

$$\begin{aligned}
b(i+1,j) &= b((i+1)-1, j-1) + b((i+1)-1, j) && \text{(by definition, since } 1 \leq j \leq (i+1)-1) \\
&= b(i, j-1) + b(i,j) \\
&= \frac{i!}{(j-1)!(i-(j-1))!} + \frac{i!}{j!(i-j)!} && \text{(by the inductive hypothesis, since } 1 \leq j \leq i) \\
&= \frac{i!}{(j-1)!(i-j)!}\left(\frac{1}{(i+1)-j} + \frac{1}{j}\right) && \text{(extracting a common factor and simplifying)} \\
&= \frac{i!}{(j-1)!(i-j)!} \cdot \frac{j + (i+1-j)}{j((i+1)-j)} \\
&= \frac{i!(i+1)}{((j-1)!j)((i-j)!((i+1)-j))} && \text{(reordering factors in the denominator)} \\
&= \frac{(i+1)!}{j!((i+1)-j)!},
\end{aligned}$$

so that the identity is also correct in this last case — as is required to complete the inductive step.

If follows by induction on i that, for every integer $i \geq 0$,

$$b(i,j) = \frac{i!}{j!(i-j)!} \quad \text{for every integer } j \text{ such that } 0 \leq j \leq i.$$

Chapter 11

Solutions for Algorithm Analysis Exercises and Tests

11.1 Asymptotic Notation

11.1.1 Solutions for Selected Exercises

Solution for Exercise #2 in Section 6.10

The "limit test for $O(f)$" (Theorem 6.12) will be used to show that $n(\ln n)^2 \in O(n\sqrt{n})$.

Let $f(n) = n(\ln n)^2$ and let $g(n) = n\sqrt{n} = n^{1.5}$. It is necessary to compute the limit as n approaches $+\infty$ of $f(n)/g(n)$.

Since

$$\lim_{n\to+\infty} f(n) = \lim_{n\to+\infty} g(n) = +\infty,$$

we'll need to use either cancellation of terms or l'Hôpital's rule to compute this limit.

Two solutions will be given (just to make it clear that there's frequently more than one right way to solve a problem).

In the first solution, l'Hôpital's rule will be used. It'll turn out that we need to use it more than once.

$$\begin{aligned}
\lim_{n\to+\infty} \frac{f(n)}{g(n)} &= \lim_{n\to+\infty} \frac{f'(n)}{g'(n)} \\
&= \lim_{n\to+\infty} \frac{n(2(\ln n)(1/n)) + (\ln n)^2}{1.5n^{0.5}} \\
&= \lim_{n\to+\infty} \frac{2\ln n + (\ln n)^2}{1.5n^{0.5}}.
\end{aligned}$$

As this suggests, $f'(n) = 2\ln n + (\ln n)^2$ and $g'(n) = 1.5n^{0.5}$. Now,

$$\lim_{n\to+\infty} f'(n) = \lim_{n\to+\infty} g'(n) = +\infty,$$

so it isn't clear what the limit of the ratio $f'(n)/g'(n)$ is — but it's possible to use l'Hôpital's rule

a second time:

$$\begin{aligned}
\lim_{n\to+\infty} \frac{f(n)}{g(n)} &= \lim_{n\to+\infty} \frac{f'(n)}{g'(n)} \\
&= \lim_{n\to+\infty} \frac{f''(n)}{g''(n)} \\
&= \lim_{n\to+\infty} \frac{2/n + 2\ln n/n}{0.75n^{-0.5}} \\
&= \lim_{n\to+\infty} \frac{8\ln n + 8}{3n^{0.5}} \\
&= \lim_{n\to+\infty} \frac{\hat{f}(n)}{\hat{g}(n)},
\end{aligned}$$

where $\hat{f}(n) = 8\ln n + 8$ and $\hat{g}(n) = 3n^{0.5}$ — and where the fraction was simplified in the final steps by multiplying both numerator and denominator by $4n$.

We're still not done, since

$$\lim_{n\to+\infty} \hat{f}(n) = \lim_{n\to+\infty} \hat{g}(n) = +\infty,$$

but progress is being made (at least, the numerator and denominator have become simpler) — and it's still correct to use l'Hôpital's rule.

$$\begin{aligned}
\lim_{n\to+\infty} \frac{f(n)}{g(n)} &= \lim_{n\to+\infty} \frac{\hat{f}(n)}{\hat{g}(n)} \\
&= \lim_{n\to+\infty} \frac{\hat{f}'(n)}{\hat{g}'(n)} \\
&= \lim_{n\to+\infty} \frac{8/n}{1.5n^{-0.5}} \\
&= \lim_{n\to+\infty} \frac{16}{3\sqrt{n}} \\
&= 0.
\end{aligned}$$

We can now conclude by the limit test that $f \in O(g)$ — that is, $n(\ln n)^2 \in O(n\sqrt{n})$ — as desired.

This first solution used a combination of applications of l'Hôpital's rule and simplifications. If simplification hadn't been used between applications of l'Hôpital's rule, then the limit to be computed would have become more and more *complicated*, instead of simpler, and it's unlikely that the limit would ever have been computed.

In the second solution we'll start by using cancellation, and then we'll use l'Hôpital's rule after that. This time, let $\hat{f}(n) = (\ln n)^2$ and let $\hat{g}(n) = \sqrt{n} = n^{0.5}$. In the following derivation, we'll be able to use l'Hôpital's rule to compute the limit of $\hat{f}(n)/\hat{g}(n)$, because

$$\lim_{n\to+\infty} \hat{f}(n) = \lim_{n\to+\infty} \hat{g}(n) = +\infty.$$

Now

$$\begin{aligned}
\lim_{n\to+\infty} \frac{f(n)}{g(n)} &= \lim_{n\to+\infty} \frac{n(\ln n)^2}{n\sqrt{n}} \\
&= \lim_{n\to+\infty} \frac{n\hat{f}(n)}{n\hat{g}(n)} \\
&= \lim_{n\to+\infty} \frac{\hat{f}(n)}{\hat{g}(n)} \\
&= \lim_{n\to+\infty} \frac{\hat{f}'(n)}{\hat{g}'(n)} \\
&= \lim_{n\to+\infty} \frac{(2\ln n)/n}{0.5n^{-0.5}} \\
&= \lim_{n\to+\infty} \frac{4\ln n}{n^{0.5}} \\
&= \lim_{n\to+\infty} \frac{\bar{f}(n)}{\bar{g}(n)},
\end{aligned}$$

where $\bar{f}(n) = 4\ln n$ and $\bar{g}(n) = n^{0.5}$. In this case,

$$\lim_{n\to+\infty} \bar{f}(n) = \lim_{n\to+\infty} \bar{g}(n) = +\infty,$$

so

$$\begin{aligned}
\lim_{n\to+\infty} \frac{f(n)}{g(n)} &= \lim_{n\to+\infty} \frac{\bar{f}(n)}{\bar{g}(n)} \\
&= \lim_{n\to+\infty} \frac{\bar{f}'(n)}{\bar{g}'(n)} \\
&= \lim_{n\to+\infty} \frac{4/n}{0.5n^{-0.5}} \\
&= \lim_{n\to+\infty} \frac{8}{\sqrt{n}} \\
&= 0.
\end{aligned}$$

We can now conclude that $f \in O(g)$, as desired.

In the instructor's opinion, the second solution is the better of the two, because it begins with a simpler technique (which replaces the first application of l'Hôpital's rule, in the first solution) and leaves a slighter easier problem to be solved. The difference between the solutions would have been more dramatic — making the second solution seem even better — if the common factor to be eliminated from $f(n)$ and $g(n)$ had been a higher power of n (say, n^2) — for, then, a single application of cancellation would have replaced *several* applications of l'Hôpital's rule.

Solution for Exercise #4 in Section 6.10

This problem can be solved using the "limit test for $\Omega(f)$" (Theorem 6.15). Cancellation of a common factor of $n\ln n$ and l'Hôpital's rule will both be used to compute the desired limit.

Let $\hat{f}(n) = n$ and $\hat{g}(n) = \ln n$, so that $n^2 \ln n = (n\ln n)\hat{f}(n)$ and $n(\ln n)^2 = (n\ln n)\hat{g}(n)$. Since $n\ln n$ is strictly greater than zero whenever $n \geq 2$, we can cancel out the common factor of $n\ln n$ without changing the limit in the derivation that follows. Note as well that

$$\lim_{n\to+\infty} \hat{f}(n) = \lim_{n\to+\infty} \hat{g}(n) = +\infty,$$

so that l'Hôpital's rule can be used to compute the limit as n approaches $+\infty$ of $\hat{f}(n)/\hat{g}(n)$.

The limit of the ratio of the functions we wish to compare is

$$\begin{aligned}
\lim_{n\to+\infty} \frac{n^2 \ln n}{n(\ln n)^2} &= \lim_{n\to+\infty} \frac{(n \ln n)\hat{f}(n)}{(n \ln n)\hat{g}(n)} \\
&= \lim_{n\to+\infty} \frac{\hat{f}(n)}{\hat{g}(n)} \\
&= \lim_{n\to+\infty} \frac{\hat{f}'(n)}{\hat{g}'(n)} \\
&= \lim_{n\to+\infty} \frac{1}{1/n} \\
&= \lim_{n\to+\infty} n \\
&= +\infty.
\end{aligned}$$

It follows by the limit test for $\Omega(f)$ that $n^2 \ln n \in \Omega(n(\ln n)^2)$, as required.

Note that (as in the solutions for Exercise #2) it would also have been possible to start by using l'Hôpital's rule, instead of cancelling out a common factor first. However — as you'll discover, if you try this approach — the resulting solution would have been longer and somewhat more complicated than the one given here.

Solution for Exercise #5 in Section 6.10

If $n \geq 1$ then $n^2 \leq n^4$, $n \leq n^4$, and $1 \leq n^4$, so that if $n \geq 1$ then

$$n^4 + 3n^2 + 8n + 100 \leq n^4 + 3n^4 + 8n^4 + 100n^4 = 112n^4.$$

Therefore, if $c = 112$ and $N = 1$, then these are clearly positive constants, and

$$n^4 + 3n^2 + 8n + 100 \leq cn^4 \qquad \text{for every integer } n \geq N.$$

It follows by the definition of "$O(f)$" that $n^4 + 3n^2 + 8n + 100 \in O(n^4)$.

Note: Lots of other choices could have been made for the constants c and N, as well.

For example, if the constant $N = 4$ was used then we could have taken advantage of the fact that, for $n \geq 4$,

$$3n^2 \leq \frac{1}{2}n^4, \quad 8n \leq \frac{1}{2}n^4, \quad \text{and} \quad 100 \leq n^4$$

so that $n^4 + 3n^2 + 8n + 100 \leq 3n^4$ if $n \geq 4$. In other words we could have chosen $N = 4$ and $c = 3$ in the above argument.

How small could you have chosen c to be, using $N = 5$?

Solution for Exercise #6(a) in Section 6.10

The claim that "for all asymptotically positive functions f and g, either $f \in O(g)$, $g \in O(f)$, or both," is **false**.

Proof. It is sufficient to exhibit a pair of asymptotically positive functions f and g such that $f \notin O(g)$ and $g \notin O(f)$. One such pair was used in examples in Chapter 6 — let

$$f(n) = n^2 \qquad \text{and} \qquad g(n) = \begin{cases} n & \text{if } n \text{ is odd,} \\ n^3 & \text{if } n \text{ is even.} \end{cases}$$

These total functions are clearly asymptotically positive: $f(n) > 0$ and $g(n) > 0$ for every integer $n \geq 1$.

Suppose that $f \in O(g)$; then, by the definition of "$O(g)$," there must exist constants $c > 0$ and $N \geq 0$ such that

$$f(n) \leq cg(n) \qquad \text{for every integer } n \geq N. \tag{11.1}$$

However, if n is odd and $n > \max(0, c, N)$, then

$$f(n) - cg(n) = n^2 - cn = n(n - c) > 0,$$

since $n > 0$ and $n - c > 0$. That is, $f(n) > cg(n)$ for some integer $n > N$, which contradicts inequality (11.1). Thus, $f \notin O(g)$.

Now suppose, instead, that $g \in O(f)$; then, by the definition of "$O(f)$," there must exist constants $c' > 0$ and $N' \geq 0$ such that

$$g(n) \leq c'f(n) \qquad \text{for every integer } n \geq N'. \tag{11.2}$$

However, if n is *even* and $n > \max(0, c', N')$, then

$$g(n) - c'f(n) = n^3 - c'n^2 = n^2(n - c') > 0,$$

since $n > 0$ and $n - c' > 0$. That is, $g(n) > c'f(n)$ for some integer $n > N'$, which contradicts inequality (11.2). Thus, $g \notin O(f)$.

Since $f \notin O(g)$ and $g \notin f$, this pair of functions provides a counterexample that proves that the given claim is false.

□

Note that many other functions could have been used in this proof; these were chosen because they'd been discussed already.

Solution for Exercise #6(c) in Section 6.10

The claim that "for all asymptotically positive functions f and g, if $f \in O(g)$ and $f \notin \Theta(g)$, then $f \in o(g)$ is **false**.

Proof. It is sufficient to give two asymptotically positive functions f and g such that $f \in O(g)$ and $f \notin \Theta(g)$ but such that $f \notin o(g)$ as well.

Let

$$f(n) = \begin{cases} n & \text{if } n \text{ is even,} \\ n^2 & \text{if } n \text{ is odd,} \end{cases} \qquad \text{and} \qquad g(n) = n^2.$$

Since $n \leq n^2$ for all $n \geq 1$ it is clear that if $c = 1$ and $N = 1$ then c and N are positive constants, and that $f(n)$ and $g(n)$ are both defined, and $f(n) \leq cg(n)$, for every integer $n \geq N$. It follows by the definition of "$O(g)$" that $f \in O(g)$.

Suppose (in order to obtain a proof by contradiction) that $f \in \Theta(g)$. Then it follows by the definition of $\Theta(g)$ that $f \in \Omega(g)$ as well, so that, by the definition of $\Omega(g)$, there exist constants $c' > 0$ and N' such that $f(n)$ and $g(n)$ are both defined, and $f(n) \geq c'g(n)$, for every $n \geq N'$.

Now consider any *even* integer n such that $n > \max(N', 1/c')$; since $n > N'$, $f(n) \geq c'g(n)$, so that

$$f(n) - c'g(n) \geq 0.$$

On the other hand, since $g(n)$ is positive and $n > 1/c'$, $c' > 1/n$, and since n is even,

$$\begin{aligned} f(n) - c'g(n) &< f(n) - g(n)/n \\ &= n - n^2/n \\ &= 0, \end{aligned}$$

contradicting the previous inequality. Therefore $f \notin \Theta(g)$.

Note: This part of the proof serves to show that it can be helpful to go to the end and work backwards when stuck; I had to identify what I *wanted* to prove and then work backwards in order to decide that I needed to require that n is bigger than $1/c'$.

Finally, suppose (again, in order to obtain a proof by contradiction) that $f \in o(g)$. Then it follows by the definition of "$o(g)$" that, for every constant $c'' > 0$, there exists a constant $N'' \geq 0$ such that $f(n)$ and $g(n)$ are both defined, and $f(n) \leq c''g(n)$, for every integer $n \geq N''$.

In particular, this must be true for $c'' = 1/2$. so there must exist a constant N'' such that $f(n)$ and $g(n)$ are defined and

$$f(n) \leq \frac{1}{2}g(n)$$

for every integer $n \geq N''$.

Suppose now, that n is an *odd* integer such that $n \geq \max(1, N'')$. Clearly $n \geq N''$ so that the above inequality must hold.

However, since n is odd, $f(n) = n^2 = g(n)$; since $n \geq 1$, $n^2 > 0$, so that

$$f(n) = g(n) > \frac{1}{2}g(n),$$

contradicting the inequality that's already been established. Therefore, $f \notin o(g)$ after all.

Note: *Any* positive constant that is less than 1 could have been chosen here for c'' — there was nothing special about the choice of $1/2$.

Since $f \in O(g)$, $f \notin \Theta(g)$, and $f \notin o(g)$, it is clear that the statement, "if $f \in O(g)$ and $f \notin \Theta(g)$ then $f \in o(g)$" is false for this choice of functions — as is needed to disprove the claim. □

Solution for Exercise #6(d) in Section 6.10

The claim that "for all asymptotically positive functions f and g, $f \in O(g)$ if and only if $g \in \Omega(f)$," is **true**.

Proof. This is equivalent to the following two claims,

1. For all asymptotically positive functions f and g, if $f \in O(g)$ then $g \in \Omega(f)$, and
2. For all asymptotically positive functions f and g, if $g \in \Omega(f)$ then $f \in O(g)$,

and we will prove these two subclaims separately in order to establish the original claim.

Proof of First Subclaim: Let f and g be asymptotically positive functions and suppose $f \in O(g)$. Then there exist constants $c > 0$ and N such that $f(n)$ and $g(n)$ are both defined and

$$f(n) \leq cg(n) \qquad \text{for every integer } n \geq N. \tag{11.3}$$

Let $c' = 1/c$ and let $N' = N$; then c' and N' are constants, $c' > 0$ (since c is positive), and (dividing both sides of inequality (11.3) by c and switching sides), $f(n)$ and $g(n)$ are both defined and

$$g(n) \geq c'f(n) \qquad \text{for every integer } n \geq N'.$$

If follows by the definition of $\Omega(f)$ that $g \in \Omega(f)$. Now, since f and g were "arbitrarily chosen" asymptotically positive functions such that $f \in O(g)$, this proves the first subclaim.

Proof of Second Subclaim: Let f and g be asymptotically positive functions and suppose $g \in \Omega(f)$. Then there exist constants $c > 0$ and N such that $f(n)$ and $g(n)$ are both defined and

$$g(n) \geq cf(n) \qquad \text{for every integer } n \geq N. \tag{11.4}$$

Let $c' = 1/c$ and let $N' = N$; then c' and N' are constants, $c' > 0$, and (dividing both sides of inequality (11.4) by c and switching sides), $f(n)$ and $g(n)$ are both defined and

$$f(n) \leq c'g(n) \qquad \text{for every integer } n \geq N'.$$

If follows by the definition of $O(g)$ that $f \in O(g)$. Now, since f and g were "arbitrarily chosen" asymptotically positive functions such that $g \in \Omega(f)$, this proves the second subclaim.

Since both subclaims are true, the original claim is true as well. □

Solution for Exercise #6(f) in Section 6.10

The claim that "for all asymptotically positive functions f, g, and h, if $f \in O(h)$ and $g \in O(g)$, then $f + g \in O(h)$," is **true**.

Proof. Let f, g, and h be arbitrary asymptotically positive functions such that $f \in O(h)$ and $g \in O(h)$.

Since $f \in O(h)$ it follows by the definition of "$O(h)$" that there exist constants $c_1 > 0$ and N_1 such that $f(n)$ and $h(n)$ are both defined, and $f(n) \leq c_1 h(n)$, for every integer $n \geq N_1$.

Since $g \in O(h)$, it also follows by the definition of "$O(h)$" that there exist constants $c_2 > 0$ and $N_2 \geq 0$ such that $g(n)$ and $h(n)$ are both defined, and $g(n) \leq c_2 h(n)$, for every integer $n \geq N_2$.

Now, let $N = \max(N_1, N_2)$ and let $c = c_1 + c_2$. Clearly N is a constant since N_1 and N_2 are both constants, and c is a positive constant, since c_1 and c_2 are both positive constants.

If n is any integer such that $n \geq N$ then (by construction of N), $n \geq N_1$ and $n \geq N_2$ as well.

Therefore $(f + g)(n) = f(n) + g(n)$ is defined, since $f(n)$ and $g(n)$ are both defined, and $h(n)$ is defined as well.

Furthermore,

$$\begin{aligned} (f+g)(n) &= f(n) + g(n) \\ &\leq c_1 h(n) + c_2 h(n) \\ &= (c_1 + c_2) h(n) \\ &= ch(n). \end{aligned}$$

It therefore follows by the definition of "$O(h)$" that $f + g \in O(h)$.

Since f, g, and h were arbitrarily chosen asymptotically positive functions such that $f \in O(h)$ and $g \in O(h)$, it follows that if $f \in O(h)$ and $g \in O(h)$ then $f + g \in O(h)$, for *all* asymptotically positive functions f, g, and h, as required.

□

11.1.2 Solutions for Sample Tests

Solution for Sample Test in Section 6.12.1

1. (5 marks — 1 mark for each part) For any pair of functions $f(n)$ and $g(n)$, exactly one of the following six claims is correct.

   (i) $f(n) \in \Theta(g(n))$.
   (ii) $f(n) \in o(g(n))$.
   (iii) $f(n) \in O(g(n))$, but $f(n) \notin \Theta(g(n))$ and $f(n) \notin o(g(n))$.
   (iv) $f(n) \in \omega(g(n))$.
   (v) $f(n) \in \Omega(g(n))$, but $f(n) \notin \Theta(g(n))$ and $f(n) \notin \omega(g(n))$.
   (vi) None of the above.

   For each of the following pairs of functions $f(n)$ and $g(n)$ say which one of these claims is correct. **You do not need to prove your answer.**

   (a) $f(n) = n^3$, $g(n) = 100n^2 \ln n$ **Answer:** (iv) $f(n) \in \omega(g(n))$

   (b) $f(n) = 2^n$, $g(n) = 3^n$ **Answer:** (ii) $f(n) \in o(g(n))$

   (c) $f(n) = n + (\ln n)^2$, $g(n) = n - \sqrt{n}$ **Answer:** (i) $f(n) \in \Theta(g(n))$

   (d) $f(n) = \begin{cases} n & \text{if } n \text{ is even,} \\ n^2 & \text{if } n \text{ is odd,} \end{cases}$ $g(n) = \frac{n^2}{2}$ **Answer:** (iii) $f(n) \in O(g(n))$ but ...

   (e) $f(n) = n^{1000}$, $g(n) = \frac{1.5^n}{n^2}$ **Answer:** (ii) $f(n) \in o(g(n))$

2. (5 marks) Prove that $n^2 \in \Omega(n \ln n)$.

   Proof. The limit test for $\Omega(f)$ will be used to establish this result.

   Let $f(n) = n$ and let $g(n) = \ln n$. Then $n^2 = nf(n)$ and $n \ln n = ng(n)$. Cancellation, followed by l'Hôpital's rule, can be used to compute the desired limit. Note that

   $$\lim_{n \to +\infty} f(n) = \lim_{n \to +\infty} g(n) = +\infty,$$

so that l'Hôpital's rule is used correctly in the following derivation:

$$\begin{aligned}
\lim_{n\to+\infty} \frac{n^2}{n\ln n} &= \lim_{n\to+\infty} \frac{nf(n)}{ng(n)} \\
&= \lim_{n\to+\infty} \frac{f(n)}{g(n)} \\
&= \lim_{n\to+\infty} \frac{f'(n)}{g'(n)} \\
&= \lim_{n\to+\infty} \frac{1}{1/n} \\
&= \lim_{n\to+\infty} n \\
&= +\infty.
\end{aligned}$$

It follows that $n^2 \in \Omega(n \ln n)$, as desired. □

Note: Since $n \geq \ln n$ for every integer $n \geq 1$, $n^2 \geq n \ln n$ for all $n \geq 1$ as well, and the claim could also have been proved using the definition of "$\Omega(n \ln n)$" (using the constants $c = 1$ and $N = 1$).

3. (10 marks — 5 marks for each part) Say whether each of the following claims is true or false, and prove your answer.

   (a) For every function f such that $f(n)$ is defined and $f(n) > 0$ for all $n \in \mathbb{N}$, $f \in \Theta(f)$.

   The claim is **true**.

   Proof. Let f be any function such that $f(n)$ is defined and strictly greater than 0 for all $n \in \mathbb{N}$. Then

   $$\lim_{n\to+\infty} \frac{f(n)}{f(n)} = \lim_{n\to+\infty} 1 = 1.$$

   Since f is asymptotically positive, it follows that $f \in \Theta(f)$ (by the limit test for $\Theta(f)$). □

   Note: This could also have been proved using the definition of "$\Theta(f(n))$," since it is clear that $c_L f(n) \leq f(n) \leq c_U f(n)$ for all $n \geq N$, if you choose $c_L = c_U = 1$ and $N = 0$.

   (b) For all functions f and g such that $f(n)$ and $g(n)$ are defined for all $n \in \mathbb{N}$, and $f(n)$ and $g(n)$ are strictly greater than 0 whenever $n \geq N$ for some integer N (so that f and g are asymptotically positive), if $f \in \Omega(g)$ then $f \in \omega(g)$.

   The claim is **false**.

Proof. If is sufficient to give a pair of functions f and g such that both f and g are asymptotically positive, $f \in \Omega(g)$, and $f \notin \omega(g)$ (and, of course, to prove that these claims about f and g are true).

Let $f(n) = g(n) = n$ for all $n \in \mathbb{N}$. Then, clearly, f and g are both asymptotically positive ($f(n) = g(n) = n > 0$ for all $n \geq 1$). Since $f = g$, and $g(n) > 0$ for all $n > 0$,

$$\lim_{n \to +\infty} \frac{f(n)}{g(n)} = \lim_{n \to +\infty} 1 = 1.$$

Since f and g are asymptotically positive, and this limit exists and is a constant that is greater than zero, it follows by the limit test for $\Theta(f)$ that $f \in \Omega(g)$.

Since the limit is *finite* it follows by the limit test for $\omega(f)$ that $f \notin \omega(g)$.

Thus, this pair of functions serve as a counterexample that disproves the claim. □

Note: Any *other* pair of asymptotically positive functions f and g such that $f \in \Omega(g)$ but $f \notin \omega(g)$ could have been used as a counterexample in order to disprove this claim, as well.

Solution for Sample Test in Section 6.12.2

1. (6 marks — 1 mark for each part) For any pair of functions $f(n)$ and $g(n)$, exactly one of the following six claims is correct. (However, each claim might be true about zero, one, or several of the following pairs of functions!)

   (i) $f(n) \in \Theta(g(n))$.

   (ii) $f(n) \in o(g(n))$.

   (iii) $f(n) \in O(g(n))$, but $f(n) \notin \Theta(g(n))$ and $f(n) \notin o(g(n))$.

   (iv) $f(n) \in \omega(g(n))$.

   (v) $f(n) \in \Omega(g(n))$, but $f(n) \notin \Theta(g(n))$ and $f(n) \notin \omega(g(n))$.

   (vi) None of the above.

   For each of the following pairs of functions $f(n)$ and $g(n)$ say which one of these claims is correct. **You do not need to prove your answer.**

   (a) $f(n) = \frac{1}{10}n^2$, $g(n) = 20n$ **Answer:** (iv) $f(n) \in \omega(g(n))$.

   (b) $f(n) = n^2 + 2n + 1$, $g(n) = n^2 - 2n + 1$ **Answer:** (i) $f(n) \in \Theta(g(n))$.

   (c) $f(n) = \begin{cases} n & \text{if } n \text{ is odd,} \\ n^3 & \text{if } n \text{ is even,} \end{cases}$ $g(n) = n^2$ **Answer:** (vi) None of the above.

   (d) $f(n) = n^4$, $g(n) = 2^n$ **Answer:** (ii) $f(n) \in o(g(n))$.

   (e) $f(n) = n^2$, $g(n) = \begin{cases} n^2 + 2n + 1 & \text{if } n \text{ is odd,} \\ n \ln n & \text{if } n \text{ is even,} \end{cases}$ **Answer:** (v) $f \in \Omega(g(n))$, but ...

   (f) $f(n) = \frac{1}{1000}(\ln n)^{1000}$, $g(n) = n^{1/1000}$ **Answer:** (ii) $f(n) \in o(g(n))$.

2. (5 marks) Prove that $2n + 3\ln n \in \Theta(n)$.

Proof. The limit test for "big Theta" will be applied. Let $f(n) = 2n + 3$ and $g(n) = n$, and note that

$$\lim_{n\to+\infty} f(n) = \lim_{n\to+\infty} g(n) = +\infty,$$

so that l'Hôpital's rule can be applied to compute the limit as n approaches $+\infty$ of $f(n)/g(n)$.

$$\begin{aligned}\lim_{n\to+\infty} \frac{f(n)}{g(n)} &= \lim_{n\to+\infty} \frac{f'(n)}{g'(n)} \\ &= \lim_{n\to+\infty} \frac{2 + (3/n)}{1} \\ &= 2.\end{aligned}$$

Since the above limit is a positive constant, it follows by the limit test that $2n + 3\ln n \in \Theta(n)$. □

3. (6 marks) Say whether the following claim is true or false, and then prove your answer.

Claim: For all asymptotically positive functions f and g, if $f \in o(g)$ then $g \in \omega(f)$.

Status of Claim: The claim is **true**.

Proof #1 — Application of Definitions

Let f and g be arbitrarily chosen asymptotically positive functions, and suppose that $f \in o(g)$ but $g \notin \omega(f)$.

Since $g \notin \omega(f)$, it follows by the definition of $\omega(f)$ that there is some positive constant $c_1 > 0$, such that for every constant $N \geq 0$, there is an integer $n \geq N$ such that either $f(n)$ is undefined, $g(n)$ is undefined, or $g(n) < c_1 f(n)$ — for this is the *negation* of the condition given in the definition of $\omega(f$.

On the other hand, $f \in o(g)$ and $\frac{1}{c_1}$ is (also) a positive constant, so it follows by the definition of $o(g)$ that there exists a constant $\hat{N} \geq 0$ such that $f(n)$ and $g(n)$ are both defined and

$$f(n) \leq \tfrac{1}{c_1} g(n)$$

for every integer $n \geq \hat{N}$.

However, the above two statements contradict each other, in the sense that the constant $\hat{N}$ identified in the second statement fails to satisfy the property that is supposed to hold for *all* nonnegative constants, given in the first statement. (Note that, since $c_1 > 0$, if $f(n) \leq \frac{1}{c_1} g(n)$ then $g(n) \geq c_1 f(n)$.)

Since we have a contradiction, it *cannot* be true that $f \in o(g)$ and $g \in \omega(f)$. In other words, if $f \in o(g)$ then $g \in \omega(f)$.

Since f and g were arbitrarily chosen, this establishes the claim.

Proof #2: Application of Limit Tests

Recall that the "limit tests" for "little-oh" and "little-omega" were *necessary* as well as sufficient conditions, for asymptotic relations to hold (unlike the limit tests for "big-Oh," et cetera).

Suppose now that f and g are arbitrarily chosen asymptotically positive functions such that $f \in o(g)$. Then, by the limit test for $o(g)$,

$$\lim_{n \to +\infty} \frac{f(n)}{g(n)} = 0.$$

Note, as well, that for sufficiently large n, $f(n)$ and $g(n)$ are both defined and

$$\frac{f(n)}{g(n)} > 0,$$

since the functions f and g are both asymptotically positive.

It follows (by the definition of "limit") that, for every real number $M > 0$, there exists an integer $N \geq 0$ such that

$$0 < \frac{f(n)}{g(n)} < \frac{1}{M} \qquad \text{for every integer } n \geq N;$$

otherwise, it wouldn't be true that

$$\lim_{n \to +\infty} \frac{f(n)}{g(n)} = 0.$$

However, this implies that for every real number $M > 0$, there exists an integer $N \geq 0$ (namely, the same one as above) such that

$$\frac{g(n)}{f(n)} > M \qquad \text{for every integer } n \geq N,$$

and it follows (by the definition of "limit") that

$$\lim_{n \to +\infty} \frac{g(n)}{f(n)} = +\infty.$$

This implies, by the limit test for "little-omega," that $g \in \omega(f)$.

Therefore, if $f \in o(g)$ then $g \in \omega(f)$. Since f and g were arbitrarily chosen, this establishes the claim.

4. Suppose now that you have two algorithms, A and B, that solve the same problem. The worst case running time of algorithm A (for input size n) is $T_A(n)$, and the worst case running time of algorithm B is $T_B(n)$. $T_A \in o(T_B)$.

   (a) (2 marks) Give the definition for the "worst case running time" of an algorithm.

   Solution: This is Definition 6.1 on page 64.

(b) (2 marks) Give the definition for "$f \in o(g)$," for asymptotically positive functions f and g.

Answer: This is Definition 6.6 on page 66.

(c) (1 mark) Does the above statement about algorithms A and B imply that algorithm A will *always* solve the problem more quickly than algorithm B? That is, this does imply that algorithm A will solve the problem more quickly than algorithm B, *on every possible input*? Your answer should be either "Yes" or "No."

Answer: No, the above statement does not imply this.

(d) (3 marks) Say, in your own words, what the the above statement *does* imply about the relative performance of the two algorithms. Be as precise as you can (but try to do something different than just restating your answer for the first parts of this question).

Answer: This implies that for every sufficiently large integer n, there exists at least one input I of size n such that the time used by the algorithm B on input I is *much* larger than the time $T_A(n)$ spent by algorithm A on *any* input of size n (that is, the time spent by algorithm A on inputs of size n in the worst case). Thus $T_B(n)$, the time spent by algorithm B on inputs of size n in the worst case, is much larger than $T_A(n)$, the time spent by algorithm A on inputs of size n in the worst case, as well.
In particular, for any positive constant $\epsilon > 0$, $\frac{T_A(n)}{T_B(n)} < \epsilon$ if n is sufficiently large.
Note, though, that

- this *does not* necessarily imply that $\frac{T_A(n)}{T_B(n)} < \epsilon$ when n is small, and
- this *does not* necessarily imply that algorithm A runs more quickly than algorithm B on "most" inputs: It sometimes happens that the "expected" running time of algorithm A on inputs of size n (defined using some plausible probability distribution for the inputs of size n) is actually *larger* than the expected running time of algorithm B on inputs of size n (defined using the same probability distribution), even though the above statement about *worst case* running times holds.

Note: You certainly *wouldn't* need to make all the above points in order to get full marks for this last part of the question. On the other hand, you'd lose some marks if you made statements that are false (such as, statements that contradict the information given here).

11.2 Summations

11.2.1 Solutions for Selected Exercises

Solution for Exercise #1(d) in Section 7.12

Following the hint for this problem, we'll begin by trying to express the function whose sum is being computed in the form

$$\frac{c}{(2i+1)^2} - \frac{c}{(2i-1)^2}.$$

This is clearly equal to

$$\frac{c((2i-1)^2 - (2i+1)^2)}{(2i+1)^2(2i-1)^2} = \frac{-8ci}{(2i+1)^2(2i-1)^2},$$

and we want to equate it with

$$\frac{i}{(2i+1)^2(2i-1)^2}.$$

This can be accomplished by setting $c = -\frac{1}{8}$.

Now, we know that

$$\begin{aligned}\sum_{i=1}^{n} \frac{i}{(2i+1)^2(2i-1)^2} &= \sum_{i=1}^{n} \left(-\frac{1}{8}\left(\frac{1}{(2i+1)^2}\right) + \frac{1}{8}\left(\frac{1}{(2i-1)^2}\right)\right) \\ &= \sum_{i=1}^{n} (f(i+1) - f(i)),\end{aligned}$$

where

$$f(j) = -\frac{1}{8}\left(\frac{1}{(2j-1)^2}\right).$$

It should now be apparent that this is a telescoping sum, so we can continue by simply writing down its closed form and simplifying:

$$\begin{aligned}\sum_{i=1}^{n} \frac{i}{(2i+1)^2(2i-1)^2} &= \sum_{i=1}^{n} (f(i+1) - f(i)) \\ &= f(n+1) - f(1) \\ &= -\frac{1}{8}\left(\frac{1}{(2n+1)^2}\right) + \frac{1}{8}\left(\frac{1}{1^2}\right) \\ &= \frac{1}{8} - \frac{1}{8} \cdot \frac{1}{(2n+1)^2}.\end{aligned}$$

Solution for Exercise #3(b) in Section 7.12

This question asked for a "tight" asymptotic estimate in closed form for the sum

$$\sum_{i=1}^{n} i^4 \ln i.$$

Several of the methods for solving summations can be applied to answer this question, so several solutions will be presented (and compared) below.

Solution #1: Splitting the Sum and Bounding Terms. We'll start by "bounding the terms" and inspecting the resulting estimates. Occasionally, this is sufficient to solve a problem without "splitting the sum," so we'll try this first. Since

$$f(x) = x^4 \ln x$$

is a nondecreasing function on the positive real numbers and we are trying to bound

$$\sum_{i=1}^{n} i^4 \ln i = \sum_{i=1}^{n} f(i),$$

we have the estimates

$$nf(1) \leq \sum_{i=1}^{n} i^4 \ln i = \sum_{i=1}^{n} f(i) \leq nf(n).$$

Clearly $f(1) = 0$ (since $\ln 1 = 0$), so this gives us the bounds

$$0 \leq \sum_{i=1}^{n} i^4 \ln i \leq n^5 \ln n.$$

This is not good enough, since the lower bound, 0, clearly *does not* grow at the same rate as the upper bound, $n^5 \ln n$.

Since the simplest approach didn't work we'll continue by splitting the sum into two pieces, and bounding each piece. The function $f(n)$ grows reasonably slowly with n (it's more like a polynomial than an exponential), so we'll split the sum down the middle, as

$$\sum_{i=1}^{n} i^4 \ln i = \sum_{i=1}^{m} i^4 \ln i + \sum_{i=m+1}^{n} i^4 \ln i \qquad \text{for } m = \lfloor \tfrac{n}{2} \rfloor.$$

Let

$$S_L = \sum_{i=1}^{m} i^4 \ln i \qquad \text{and} \qquad S_U = \sum_{i=m+1}^{n} i^4 \ln i$$

for m as above. Then we can bound terms for both sums, to obtain the inequalities

$$0 \leq S_L \leq \lfloor \tfrac{n}{2} \rfloor \cdot \left(\lfloor \tfrac{n}{2} \rfloor \right)^4 \ln \left(\lfloor \tfrac{n}{2} \rfloor \right),$$

and

$$(n - \lfloor \tfrac{n}{2} \rfloor) \cdot \left(\lfloor \tfrac{n}{2} \rfloor + 1 \right)^4 \ln \left(\lfloor \tfrac{n}{2} \rfloor + 1 \right) \leq S_U \leq (n - \lfloor \tfrac{n}{2} \rfloor) \cdot n^4 \ln n.$$

Simplifying the above lower bound for S_U, we have

$$\begin{aligned} S_U &\geq (n - \lfloor \tfrac{n}{2} \rfloor) \cdot \left(\lfloor \tfrac{n}{2} \rfloor + 1 \right)^4 \ln \left(\lfloor \tfrac{n}{2} \rfloor + 1 \right) \\ &\geq \left(\tfrac{n}{2} \right) \cdot \left(\tfrac{n}{2} \right)^4 \ln \left(\tfrac{n}{2} \right) \\ &= \tfrac{n^5}{32} \ln \tfrac{n}{2} \\ &= \tfrac{n^5}{32} \ln n - \tfrac{n^5}{32} \ln 2 \\ &\geq \tfrac{1}{64} n^5 \ln n \qquad \text{if } n \geq 4. \end{aligned}$$

Since $S_L \geq 0$ it follows that $S = S_L + S_U \geq \frac{1}{64} n^5 \ln n$ if $n \geq 4$ as well. Combining this with the upper bound that we obtained without splitting the sum, we see that

$$\tfrac{1}{64} n^5 \ln n \leq \sum_{i=1}^{n} i^4 \ln i \leq n^5 \ln n \qquad \text{if } n \geq 4,$$

which implies that

$$\sum_{i=1}^{n} i^4 \ln n \in \Theta(n^5 \ln n).$$

Solution #2: Approximation by Integrals. Since the function

$$f(i) = i^4 \ln i$$

is a nondecreasing function, we can also approximate the sum by integrals.

It might not be immediately obvious how to integrate this function. Note, though, that

$$f(i) = g'(i) \cdot h(i)$$

where $g'(i) = i^4$ and $h(i) = \ln i$; using integration by parts, we have that

$$\int f(t)\,dt = \int g'(t)h(t)\,dt = g(t) \cdot h(t) - \int g(t)h'(t)\,dt.$$

Since $g'(t) = t^4$ we can choose $g(t) = \frac{1}{5}t^5$; since $h(t) = \ln t$, $h'(t) = \frac{1}{t}$, so that

$$\begin{aligned} \int f(t)\,dt &= \tfrac{1}{5}t^5 \ln t - \int \tfrac{1}{5}t^5 \cdot \tfrac{1}{t}\,dt \\ &= \tfrac{1}{5}t^5 \ln t - \tfrac{1}{5}\int t^4\,dt \\ &= \tfrac{1}{5}t^5 \ln t - \tfrac{1}{25}t^5 + C \end{aligned}$$

for a constant C.

Therefore, let $F(t) = \frac{1}{5}t^5 \ln t - \frac{1}{25}t^5$; then $F'(t) = f(t) = t^4 \ln t$.

Now, if we tried to approximate by integrals, starting with the original sum, then we'd (seemingly) obtain the bounds

$$\int_0^n f(t)\,dt \leq \sum_{i=1}^{n} f(i) \leq \int_1^{n+1} f(t)\,dt,$$

that is,

$$F(n) - F(0) \leq \sum_{i=1}^{n} i^4 \ln i \leq F(n-1) - F(1)$$

for $F(x)$ as above. Unfortunately $F(0)$ is not defined (it has "$\ln 0$" as a factor), so this is of no use.

In order to avoid this problem we'll split away the first term of the sum, to be considered separately, and bound the rest instead:

$$\int_1^n f(t)\,dt \leq \sum_{i=2}^{n} f(i) \leq \int_2^{n+1} f(t)\,dt,$$

that is,

$$F(n) - F(1) \leq \sum_{i=2}^{n} f(i) \leq F(n+1) - F(2).$$

Furthermore, it's clear that $f(1) = 1 \cdot \ln 1 = 0$, so that

$$\sum_{i=2}^{n} f(i) = \sum_{i=1}^{n} f(i),$$

so that

$$F(n) - F(1) \leq \sum_{i=1}^{n} f(i) \leq F(n+1) - F(2)$$

as well.

Now, this implies that

$$\begin{aligned}\sum_{i=1}^{n} f(i) &\geq F(n) - F(1) \\ &= \tfrac{1}{5}n^5 \ln n - \tfrac{1}{25}n^5 - \tfrac{1}{5}1^5 \ln 1 + \tfrac{1}{25}1^5 \\ &= \tfrac{1}{5}n^5 \ln n - \tfrac{1}{25}n^5 + \tfrac{1}{25},\end{aligned}$$

and

$$\begin{aligned}\sum_{i=1}^{n} f(i) &\leq F(n+1) - F(2) \\ &= \tfrac{1}{5}(n+1)^5 \ln(n+1) - \tfrac{1}{25}(n+1)^5 - \tfrac{1}{5} \cdot 2^5 \ln 2 + \tfrac{1}{25} \cdot 2^5.\end{aligned}$$

Now, it certainly isn't immediately clear from the above, but it can be shown (using calculus) that this upper bound has the form

$$\tfrac{1}{5}n^5 \ln n + l(n)$$

for some function $l(n) \in o(n^5 \ln n)$.

Thus, this method can also be used to establish that

$$\sum_{i=1}^{n} i^4 \ln i \in \Theta(n^5 \ln n) = \Theta(n^5 \log_2 n).$$

Comparison of Solutions. It's certainly possible to improve the bounds that were obtained by "splitting the sum and bounding terms." For example, by adding together the upper bounds derived for S_L and S_U, above, one can obtain a bound that's slightly more than half (actually approximately $\frac{17}{32}$ of) the upper bound that is given above.

You could also split the sum into a larger number of pieces, and bound each separately, to obtain upper and lower bounds for the total that are even closer together.

However, the method of "approximating by integrals" still produces upper and lower bounds that are closer together than anything you're likely to obtain by splitting the sum and bounding terms, no matter how many pieces you use when splitting the sum. In particular, the difference

between the upper and lower bounds, obtained using approximation by integrals, is in $o(n^5 \ln n)$, while the sum itself is in $\Theta(n^5 \ln n)$.

That is, splitting the sum and bounding terms only allowed us to conclude that

$$\sum_{i=1}^{n} i^4 \ln i \in \Theta(g(n)) \qquad \text{for } g(n) = n^5 \ln n;$$

approximating by integrals allowed us to conclude something stronger, namely that

$$\sum_{i=1}^{n} i^4 \ln i = \tfrac{1}{5} g(n) + l(n) \qquad \text{where } l(n) \in o(g(n));$$

and you aren't *ever* likely to get an approximation this fine using the first technique.

Of course, one advantage of the first method is its generality: You can apply it to bound sums of functions that aren't "nondecreasing," or that are difficult to integrate.

Solution for Exercise #4 in Section 7.12

This question asked for a proof that

$$\sum_{i=1}^{n} i^2 \in \Theta(n^3),$$

that *didn't* include an exact solution for the recurrence. "Completing and confirming a pattern" can be used accomplish this.

Outline of the Method. In order to solve this problem it is necessary to prove things, namely, that

1. $\sum_{i=1}^{n} i^2 \leq cn^3$ for some positive constant c, for all $n \geq 1$;
2. $\sum_{i=1}^{n} i^2 \geq \hat{c}n^3$ for some positive constant $\hat{c}$ and for all $n \geq 1$.

The first of these implies that

$$\sum_{i=1}^{n} i^2 \in O(n^3)$$

and the second implies that

$$\sum_{i=1}^{n} i^2 \in \Omega(n^3).$$

Both can be established using induction on n.

Proof that $\sum_{i=1}^{n} i^2 \in O(n^3)$. It is necessary and sufficient to prove the following.

Claim. There exists a positive constant c such that

$$\sum_{i=1}^{n} i^2 \leq cn^3$$

for every integer $n \geq 1$.

Proof. This will be proved induction on n.

Basis: $(n = 1)$: If $n = 1$ then

$$\sum_{i=1}^{n} i^2 = \sum_{i=1}^{1} i^2 = 1$$

and

$$cn^3 = c \cdot 1^3 = c$$

so that it is clear that

$$\sum_{i=1}^{n} i^2 \leq cn^3$$

in the case $n = 1$ **provided that** $c \geq 1$.

Inductive Step: Suppose now that $n \geq 1$ and that

$$\sum_{i=1}^{n} i^2 \leq cn^3.$$

In order to complete the inductive step it is necessary to prove, using only this assumption, that

$$\sum_{i=1}^{n+1} i^2 \leq c(n+1)^3$$

as well.

Now

$$\begin{aligned} \sum_{i=1}^{n+1} i^2 &= \sum_{i=1}^{n} i^2 + (n+1)^2 \\ &\leq cn^3 + (n+1)^2 && \text{by the inductive hypothesis} \\ &= cn^3 + n^2 + 2n + 1, \end{aligned}$$

and

$$c(n+1)^3 = cn^3 + 3cn^2 + 3cn + c.$$

The desired inequality will hold if the first of these polynomials is less than or equal to the second, and this will be the case if the respective *coefficients* are related in this way, that is, if

- $c \leq c$ (comparing the coefficients of n^3 for the polynomials),
- $1 \leq 3c$ (comparing the coefficients of n^2 for the polynomials),
- $2 \leq 3c$ (comparing the coefficients of n for the polynomials), and
- $1 \leq c$ (comparing the coefficients of 1 for the polynomials).

These aren't all *necessary* to ensure that the first polynomial in n is less than or equal to the second, but they are certainly *sufficient*, since $n \geq 0$. These are clearly all satisfied if and only if $c \geq 1$.

Thus $\sum_{i=1}^{n+1} i^2 \leq c(n+1)^3$ in this case.

Choice of Constant

We can now choose c to be any constant that is greater than or equal to 1 (in particular, $c = 1$ will do) in order to complete the proof of the claim. □

As mentioned at the beginning, this implies that

$$\sum_{i=1}^{n} i^2 \in O(n^3),$$

as desired.

Proof that $\sum_{i=1}^{n} i^2 \in \Omega(n^3)$. Now, the following must be proved.

Claim. There exists a positive constant $\hat{c}$ such that

$$\sum_{i=1}^{n} i^2 \geq \hat{c}n^3$$

for every integer $n \geq 1$.

Proof. This will also be proved by induction on n.

Basis: $(n = 1)$: If $n = 1$ then

$$\sum_{i=1}^{n} i^2 = \sum_{i=1}^{1} i^2 = 1$$

and

$$\hat{c}n^3 = \hat{c} \cdot 1 = \hat{c}$$

so that it is clear that

$$\sum_{i=1}^{n} i^2 \geq \hat{c}n^3$$

in the case $n = 1$ **provided that** $\hat{c} \leq 1$.

Inductive Step: Suppose now that $n \geq 1$ and that

$$\sum_{i=1}^{n} i^2 \geq \hat{c}n^3.$$

In order to complete the inductive step it is necessary to prove, using only this assumption, that

$$\sum_{i=1}^{n+1} i^2 \geq \hat{c}(n+1)^3$$

as well.

Now

$$\begin{aligned}
\sum_{i=1}^{n+1} i^2 &= \sum_{i=1}^{n} i^2 + (n+1)^2 \\
&\geq \hat{c}n^3 + (n+1)^2 && \text{by the inductive hypothesis} \\
&= \hat{c}n^3 + n^2 + 2n + 1,
\end{aligned}$$

and

$$\hat{c}(n+1)^3 = \hat{c}n^3 + 3\hat{c}n^2 + 3\hat{c}n + \hat{c}.$$

The desired inequality will hold if the first of these polynomials is greater than or equal to the second, and this will be the case if the respective coefficients are related in this way, that is, if

- $\hat{c} \geq \hat{c}$ (comparing the coefficients of n^3 for the polynomials),
- $1 \geq 3\hat{c}$ (comparing the coefficients of n^2 for the polynomials),
- $2 \geq 3\hat{c}$ (comparing the coefficients of n for the polynomials), and
- $1 \geq \hat{c}$ (comparing the coefficients of n for the polynomials).

Again, these aren't all necessary conditions to ensure that the first polynomial in n is greater than or equal to the second, but they are certainly sufficient, since $n \geq 0$. These are clearly all satisfied if (and only if) $\hat{c} \leq \frac{1}{3}$.

Thus $\sum_{i=1}^{n+1} i^2 \geq \hat{c}(n+1)^3$ in this case.

Choice of Constant

We can now choose $\hat{c}$ to be any positive constant that is less than or equal to $\frac{1}{3}$ (in particular, $\hat{c} = \frac{1}{3}$ will do) in order to complete the proof of this claim. □

As mentioned at the beginning, this implies that

$$\sum_{i=1}^{n} i^2 \in \Omega(n^3).$$

Conclusion. Since

$$\sum_{i=1}^{n} i^2 \in O(n^3) \qquad \text{and} \qquad \sum_{i=1}^{n} i^2 \in \Omega(n^3),$$

it follows that

$$\sum_{i=1}^{n} i^2 \in \Theta(n^3),$$

as desired.

11.2.2 Solutions for Sample Tests

Solution for Sample Test in Section 7.14.1

1. (10 marks) Find a function $f(x)$ (in closed form) such that $\sum_{i=1}^{n} i^4 - f(n) \in O(n^4)$, and prove that your answer is correct.

   Note: You may use the fact that the function $g(x) = x^4$ is an increasing function, without proving it.

   Solution: Let $f(x) = \frac{1}{5}x^5$.
   Since the function $g(x) = x^4$ is an increasing function,

   $$\int_0^n t^4\,dt \quad \leq \quad \sum_{i=1}^{4} i^4 \quad \leq \quad \int_1^{n+1} t^4\,dt.$$

   As well, since $f'(x) = g(x) = x^4$,

   $$\begin{aligned}\int_0^n t^4\,dt &= \int_0^n g(t)\,dt \\ &= f(n) - f(0) \\ &= \frac{1}{5}n^5 = f(n)\end{aligned}$$

   and

   $$\begin{aligned}\int_1^{n+1} t^4\,dt &= \int_1^{n+1} g(t)\,dt \\ &= f(n+1) - f(1) \\ &= \frac{1}{5}n^5 + (n^4 + 2n^3 + 2n^2 + n) \\ &= f(n) + n^4 + 2n^3 + 2n^2 + n,\end{aligned}$$

   so that $\sum_{i=1}^{n} i^4 - f(n) \in O(n^4)$ for $f(n) = \frac{1}{5}n^5$ (and this can be proved by approximating the given sum by an integral).

2. (10 marks) Prove that $\sum_{i=1}^{n} i \log_2 i \in \Theta(n^2 \log_2 n)$.

Hint: $\log_2(n/2) = (\log_2 n) - 1 \geq \frac{1}{2} \log_2 n$ for sufficiently large n (in particular, for $n \geq 4$).

Solution: Let $M = \max_{1 \leq i \leq n} i \log_2 i = n \log_2 n$. Then

$$\sum_{i=1}^{n} i \log_2 i \leq \sum_{i=1}^{n} M = nM = n^2 \log_2 n.$$

It is also true that $\sum_{i=1}^{n} i \log_2 i = S_L + S_U$, where

$$S_L = \sum_{i=1}^{\lfloor n/2 \rfloor} i \log_2 i, \quad \text{and} \quad S_U = \sum_{i=\lfloor n/2 \rfloor + 1}^{n} i \log_2 i.$$

Let $m_L = \min_{1 \leq i \leq \lfloor n/2 \rfloor} i \log_2 i$ and let $m_U = \min_{\lfloor n/2 \rfloor + 1 \leq i \leq n} i \log_2 i$. Then

$$m_L = 1 \log_2 1 = 0 \quad \text{and} \quad m_U = (\lfloor n/2 \rfloor + 1) \log_2(\lfloor n/2 \rfloor + 1) \geq (n/2) \log_2(n/2),$$

so

$$S_L \geq \sum_{i=1}^{\lfloor n/2 \rfloor} m_L = \lfloor n/2 \rfloor m_L = \lfloor n/2 \rfloor 0 = 0,$$

and

$$\begin{aligned} S_U &\geq \sum_{i=\lfloor n/2 \rfloor + 1}^{n} m_U = (n - \lfloor n/2 \rfloor)\, m_U \\ &\geq (n/2) m_U \\ &\geq (n/2)(n/2) \log_2(n/2) \\ &\geq (n^2/4)((\log_2 n)/2) \end{aligned}$$

for sufficiently large n. That is, $S_U \geq \frac{1}{8} n^2 \log_2 n$ for sufficiently large n, so

$$\sum_{i=1}^{n} i \log_2 i = S_L + S_U \geq 0 + \frac{1}{8} n^2 \log_2 n = \frac{1}{8} n^2 \log_2 n$$

for sufficiently large n.

Thus $\frac{1}{8} n^2 \log_2 n \leq \sum_{i=1}^{n} i \log_2 i \leq n^2 \log_2 n$ for sufficiently large n, so $\sum_{i=1}^{n} i \log_2 i \in \Theta(n^2 \log_2 n)$.

Solution for Sample Test in Section 7.14.2

1. (6 marks in total; marks for each part shown below)

   Each of the following summations is in a special form (for example, possibly, an arithmetic series) that you should be able to recognize, and whose closed form you should already know.

   Say way special form each summation has, and write down the *exact* solution of the above summation in closed form (*not* just the general solution for summations of this type). It isn't necessary to simplify your answer.

   (a) $\sum_{i=0}^{n-1} 3 \cdot 6^i$

   (1 mark:) This is a **geometric series**.

   (2 marks:) Solution in closed form: $3 \cdot \frac{6^n - 1}{6 - 1} = \frac{3}{5} 6^n - \frac{3}{5}$

   (b) $\sum_{i=1}^{n} (5^{i+1} \ln(i+1) - 5^i \ln i)$

   (1 mark:) This is a **telescoping sum**.

   (2 marks:) Solution in closed form: $5^{n+1} \ln(n+1) - 5^1 \ln 1 = 5^{n+1} \ln(n+1)$.

   Note: As stated above, you weren't required to simplify the expressions, so that "unsimplified" expression on the left, in each answer above, would be completely acceptable as your answer.

2. (1 mark) *Briefly* explain why you might need to find asymptotic bounds for summations in closed form, when estimating the worst case running times of programs from source code.

   Answer: Summations arise as the worst case running times (or upper bounds on them) for **while**, **repeat**, and **for** loops. However, it is generally difficult to compare them to other functions, so that equivalent expressions (or even asymptotic bounds) are more useful than the summations are themselves.

   Note: The first half of the above answer (describing the loops whose worst case running times are expressed as summations) will be sufficient to receive full marks for this question.

3. Suppose $f(x)$ is a nondecreasing function of x.

   (a) (3 marks) State the approximation of $\sum_{i=1}^{n} f(i)$ by integrals defined in class. That is, state the integrals that are known to be upper and lower bounds for this sum, and say which is which!

   Answer:

$$\int_0^n f(t)\,dt \leq \sum_{i=1}^{n} f(i) \leq \int_1^{n+1} f(t)\,dt$$

(so, the integral on the left is the lower bound, and the integral on the right is the upper bound).

(b) (5 marks) Now, use approximation by integrals to prove that

$$\sum_{i=1}^{n} i^{2.5} \in \Theta(n^{3.5}).$$

Note: You may assume that the function $f(x) = x^{2.5}$ is a nondecreasing function of x (defined on the nonnegative reals) without having to prove it.

Answer: Let $F(t) = \frac{1}{3.5}t^{3.5}$; then $F'(i) = i^{2.5}$.

It follows (using "approximation by integrals") that $\sum_{i=1}^{n} i^{2.5} \geq F(n) - F(0) = \frac{1}{3.5}n^{3.5}$, so that $\sum_{i=1}^{n} i^{2.5} \in \Omega(n^{3.5})$, since $\frac{1}{3.5}$ is a positive constant.

It also follows that $\sum_{i=1}^{n} i^{2.5} \leq F(n+1) - F(1) = \frac{1}{3.5}(n+1)^{3.5} - \frac{1}{3.5}$. Now since

$$\lim_{n\to+\infty} \frac{\frac{1}{3.5}(n+1)^{3.5} - \frac{1}{3.5}}{n^{3.5}} = \lim_{n\to+\infty} \frac{1}{3.5}\left(1+\frac{1}{n}\right)^{3.5} + \frac{1}{3.5n^{3.5}} = \frac{1}{3.5},$$

which is a positive constant, it is clear (by the limit test) that $\frac{1}{3.5}(n+1)^{3.5} - \frac{1}{3.5} \in O(n^{3.5})$, implying that $\sum_{i=1}^{n} i^{2.5} \in O(n^{3.5})$ as well, as is needed to complete the proof.

Therefore, $\sum_{i=1}^{n} i^{2.5} \in \Theta(n^{3.5})$.

4. (10 marks) Prove that

$$\sum_{i=1}^{n} i(\ln i)^3 \in \Theta(n^2(\ln n)^3).$$

There are several ways that you could prove this using techniques introduced in class; each will be acceptable (provided that the methods are used correctly).

If you get stuck, you can say *what* you'd need to do in order to finish (even though you couldn't manage to do it) in order to earn part marks.

Note: You may assume that the function $f(x) = x(\ln x)^3$ is a nondecreasing function of x (defined on the positive reals) without proving it.

Solution: This question could be answered using several of the methods given in class, but one was (in my opinion) much easier to use, in this case, than any of the others. A correct solution using *any* of these methods will receive full marks for this question.

Approach #1: Splitting the Sum and Bounding Terms. This was almost certainly the method from class to use, in order to answer this particular question most easily.

We'll start (as usual) by bounding terms without splitting the sum. Since it's given that the above function f is nondecreasing,

$$f(1) \le f(i) \le f(n) \qquad \text{if } 1 \le i \le n,$$

so that

$$n \cdot f(1) \le \sum_{i=1}^{n} f(i) = \sum_{i=1}^{n} i(\ln i)^3 \le n \cdot f(n).$$

Now $n \cdot f(1) = n \cdot 0$ and $n \cdot f(n) = n^2(\ln n)^3$, so this implies that

$$0 \le \sum_{i=1}^{n} i(\ln i)^3 \le n^2(\ln n)^3$$

which clearly implies that

$$\sum_{i=1}^{n} i(\ln i)^3 \in O(n^2(\ln n)^3).$$

However the above lower bound (0) isn't tight enough to allow us to conclude that $\sum_{i=1}^{n} i(\ln i)^3 \in \Omega(n^2(\ln n)^3)$, which we also need in order to finish.

Therefore we'll split the sum in order to try to improve the lower bound. Since the function f grows relatively slowly, we'll split down the middle, rewriting the sum as

$$\sum_{i=1}^{n} i(\ln i)^3 = S_L + S_U$$

for

$$S_L = \sum_{i=1}^{m} i(\ln i)^3 \qquad \text{and} \qquad S_U = \sum_{i=m+1}^{n} i(\ln i)^3,$$

where $m = \lfloor \frac{n}{2} \rfloor$.

Once again, since $1 \cdot (\ln 1)^3 = 0$,

$$S_L \ge m \cdot (1 \cdot (\ln 1)^3) = 0.$$

However,

$$\begin{aligned} S_U = \sum_{i=m+1}^{n} i(\ln i)^3 &\ge \sum_{i=m+1}^{n} (m+1)(\ln(m+1))^3 \\ &= (n-m)(m+1)(\ln(m+1))^3 \\ &\ge \left(\tfrac{n}{2}\right)(m+1)(\ln(m+1))^3 \qquad \text{since } n - m \ge \tfrac{n}{2} \\ &\ge \left(\tfrac{n}{2}\right)\left(\tfrac{n}{2}\right)\left(\ln\left(\tfrac{n}{2}\right)\right)^3, \end{aligned}$$

since $m + 1 \ge \frac{n}{2}$, $g(x) = \ln x$ is a nondecreasing function on the positive reals, and (again) the other factors are positive.

Simplifying the final expression and using the above bound for S_L, we can conclude that

$$\sum_{i=1}^{n} i(\ln i)^3 = S_L + S_U \geq S_U \geq \tfrac{1}{4}n^2\left((\ln n) - (\ln 2)\right)^3.$$

Now, there are at least two ways to finish.

One way would be to note that

$$\lim_{n\to+\infty} \frac{\frac{1}{4}n^2\left((\ln n) - (\ln 2)\right)^3}{n^2(\ln n)^3} = \lim_{n\to+\infty} \left(\tfrac{1}{4} - \tfrac{3}{4}\left(\tfrac{\ln 2}{\ln n}\right) + \tfrac{3}{4}\left(\tfrac{\ln 2}{\ln n}\right)^2 - \tfrac{1}{4}\left(\tfrac{\ln 2}{\ln n}\right)^3\right) = \tfrac{1}{4} > 0,$$

and conclude by the limit test that

$$\tfrac{1}{4}n^2\left((\ln n) - (\ln 2)\right)^3 \in \Omega(n^2(\ln n)^3),$$

so that $\displaystyle\sum_{i=1}^{n} i(\ln i)^3 \in \Omega(n^2(\ln n)^3)$ as well.

Another way would be note that if $n \geq 4$ then $\ln 2 \leq \frac{1}{2}\ln n$, so that

$$\tfrac{1}{4}n^2\left((\ln n) - (\ln 2)\right)^3 \geq \tfrac{1}{4}n^2\left(\tfrac{1}{2}\ln n\right)^3 = \tfrac{1}{32}n^2(\ln n)^3,$$

so that

$$\sum_{i=1}^{n} i^2(\ln i)^3 \geq \tfrac{1}{32}n^2(\ln n)^3$$

if $n \geq 4$, which *also* implies that $\displaystyle\sum_{i=1}^{n} i(\ln i)^3 \in \Omega(n^2(\ln n)^3)$.

In any case, once we've established that $\displaystyle\sum_{i=1}^{n} i(\ln i)^3 \in \Omega(n^2(\ln n)^3)$, we can recall that we've already proved that $\displaystyle\sum_{i=1}^{n} i(\ln i)^3 \in O(n^2(\ln n)^3)$, as well, and then conclude that

$$\sum_{i=1}^{n} i(\ln i)^3 \in \Theta(n^2(\ln n)^3),$$

as desired.

Note: This proof also required the fact that the function $g(x) = \ln x$ is a nondecreasing function on the positive reals, and you could use this without proof.

You will earn a small number of **bonus marks** if you provided a correct **proof** that this is a nondecreasing function (perhaps, by showing that its derivative is always positive) on the positive reals, when you used this fact.

Sketch of Approach #2: Approximation by Integrals. Since it was given that the function $f(x) = x(\ln x)^3$ is a nondecreasing function on the positive reals, this technique can also be applied correctly here.

(Note, though, that this technique was to be used on another question on the quiz; therefore the above function f was deliberately chosen to be difficult to integrate.)

Through the repeated use of "integration by parts," it is possible to discover that if

$$F(x) = \tfrac{1}{2}x^2(\ln x)^3 - \tfrac{3}{4}x^2(\ln x)^2 + \tfrac{3}{4}x^2(\ln x) - \tfrac{3}{8}x^2$$

then $F'(i) = f(i) = i(\ln i)^3$. While finding $F(x)$ in the first place isn't easy, it should be straightforward for you to verify that this is an antiderivative of f by differentiating to check that $F'(x) = f(x)$.

Now, the inequality that you'd probably write down next would be

$$\sum_{i=1}^{n} i(\ln i)^3 \geq F(n) - F(0),$$

but this is not helpful, since $F(0)$ is undefined (it has "$\ln 0$" as a factor).

Fortunately, $1 \cdot (\ln 1)^3 = 0$, so that

$$\sum_{i=1}^{n} i(\ln i)^3 = \sum_{i=2}^{n} i(\ln i)^3$$

and we can bound the sum on the right instead:

$$\begin{aligned}\sum_{i=2}^{n} i(\ln i)^3 &= \sum_{i=2}^{n} f(i) \\ &\geq F(n) - F(1)\end{aligned}$$

since $F'(x) = f(x)$ and the function $f(x)$ is nondecreasing on the set of real numbers greater than or equal to one. Thus

$$\sum_{i=1}^{n} i(\ln i)^3 \geq F(n) - F(1) = \tfrac{1}{2}n^2(\ln n)^3 - \tfrac{3}{4}n^2(\ln n)^2 + \tfrac{3}{4}n^2(\ln n) - \tfrac{3}{8}n^2 + \tfrac{3}{8}.$$

By using either the limit test, or bounding each of the trailing terms by a small multiple of $n^2(\ln n)^3$ when n is large (say, when $n \geq e^3$), you can use the above inequality to argue that

$$\sum_{i=1}^{n} i(\ln i)^3 \in \Omega(n^2(\ln n)^3).$$

It's also true that

$$\begin{aligned}\sum_{i=1}^{n} i(\ln i)^3 &\leq F(n+1) - F(1) \\ &= \tfrac{1}{2}(n+1)^2(\ln(n+1))^3 - \tfrac{3}{4}(n+1)^2(\ln(n+1))^2 \\ &\qquad + \tfrac{3}{4}(n+1)^2(\ln(n+1)) - \tfrac{3}{8}(n+1)^2 + \tfrac{3}{8},\end{aligned}$$

and you can either use a limit test or additional algebraic manipulation (assuming n is sufficiently large, say when $n \geq e^3$), to use this to prove that

$$\sum_{i=1}^{n} i(\ln i)^3 \in O(n^2(\ln n)^3),$$

as well.

Once you've done this, you can finish by observing that this implies that

$$\sum_{i=1}^{n} i(\ln i)^3 \in \Theta(n^2(\ln n)^3),$$

as desired.

Note: Since the previous question on the test required a description of the method "approximation by integrals," you *won't* earn a large number of part marks if you sketch the method again here — at least, not unless you managed to integrate the above function f correctly, and got stuck for some other reason!

Sketch of Approach #3: Confirming a Pattern.

This time, you'd try to prove each of the following claims:

Claim #1: There exists a positive constant $c > 0$ such that

$$\sum_{i=1}^{n} i(\ln i)^3 \leq cn^2(\ln n)^3$$

for every integer $n \geq 1$.

Claim #2: There exists a positive constant $\hat{c} > 0$ such that

$$\sum_{i=1}^{n} i(\ln n)^3 \geq \hat{c}n^2(\ln n)^3$$

for every integer $n \geq 1$.

As shown below, proving the first claim isn't too difficult.

Proof of Claim #1: By induction on n.

Basis: ($n = 1$). In this case

$$\sum_{i=1}^{n} i(\ln i)^3 = 1 \cdot (\ln 1)^3 = 0$$

and

$$cn^2(\ln n)^3 = c(\ln 1)^3 = 0,$$

so the desired inequality holds in this case for *any* choice of the constant c.

Inductive Step: Suppose $n \geq 1$ and that

$$\sum_{i=1}^{n} i(\ln i)^3 \leq cn^2(\ln n)^3.$$

Then

$$\begin{aligned}
\sum_{i=1}^{n+1} i(\ln i)^3 &= \sum_{i=1}^{n} i(\ln n)^3 + (n+1)(\ln(n+1))^3 \\
&\leq cn^2(\ln n)^3 + (n+1)(\ln(n+1))^3 && \text{by the inductive hypothesis} \\
&\leq cn^2(\ln(n+1))^3 + (n+1)(\ln(n+1))^3 && \text{since } cn^2 \geq 0 \text{ and } (\ln(n+1))^3 \geq (\ln n)^3 \\
&= (cn^2 + n + 1)(\ln(n+1))^3 \\
&\leq (cn^2 + 2cn + c)(\ln(n+1))^3 && \textbf{provided that } 2c \geq 1 \text{ and } c \geq 1 \\
&= c(n+1)^2(\ln(n+1))^3 && \text{as desired.}
\end{aligned}$$

It's easily checked that all of the above constraints on c are satisfied as long as we choose $c \geq 1$. In particular, $c = 1$ will do.

Therefore

$$\sum_{i=1}^{n} i(\ln i)^3 \leq cn^2(\ln n)^3$$

for every integer $n \geq 1$, if $c = 1$, so

$$\sum_{i=1}^{n} i(\ln i)^3 \in O(n^2(\ln n)^3).$$

The proof of the *second* claim is more difficult, because it *isn't* obvious how to replace $\ln(n+1)$ by something involving $\ln n$, when you are trying to establish the inequality in the other direction.

If you did everything above *correctly*, began a similar proof of the second claim and got stuck, and then explained that you would ideally finish by completing the second proof by induction, choosing a positive constant $\hat{c}$ satisfying all the identified constraints, and then concluding that

$$\sum_{i=1}^{n} i(\ln i)^3 \in \Omega(n^2(\ln n)^3)$$

and therefore that

$$\sum_{i=1}^{n} i(\ln i)^3 \in \Theta(n^2(\ln n)^3),$$

then you can expect to receive at least seven out of the possible ten marks for this question.

It *is* possible to complete this question using this technique — but it isn't easy. Here is one way to do this, which I couldn't possibly expect you to have discovered when writing the test!

Lemma: If $n \geq 2$ then $\ln n \geq \ln(n+1) - \frac{1}{n}$.

This lemma can be proved by inspecting the Taylor series for $h(x) = \ln(1+x)$, and noting that

$$\ln(n+1) - \ln n = \ln\left(1 + \tfrac{1}{n}\right).$$

Here is a proof of the second claim that uses this.

Basis: The case $n = 1$ is the same as it is for the first proof: One ends up with the inequality $0 \geq 0$, which is satisfied for any choice of the constant $\hat{c}$.

If $n = 2$ then

$$\sum_{i=1}^{n} i(\ln i)^3 = 2(\ln 2)^3$$

and

$$\hat{c}n^2(\ln n)^3 = 4\hat{c}(\ln 2)^3,$$

so $\sum_{i=1}^{n} i(\ln i)^3 \geq \hat{c}n^2(\ln n)^3$ in this case **provided that** $\hat{c} \leq \frac{1}{2}$.

Inductive Step: Suppose now that $n \geq 2$ and that

$$\sum_{i=1}^{n} i(\ln i)^3 \geq \hat{c}n^2(\ln n)^3.$$

Then

$$\begin{aligned}
\sum_{i=1}^{n+1} i(\ln i)^3 &= \sum_{i=1}^{n} i(\ln i)^3 + (n+1)(\ln(n+1))^3 \\
&\geq \hat{c}n^2(\ln n)^3 + (n+1)(\ln(n+1))^3 && \text{by the inductive hypothesis} \\
&\geq \hat{c}n^2\left(\ln(n+1) - \tfrac{1}{n}\right)^3 + (n+1)(\ln(n+1))^3 && \text{by the lemma, since } n \geq 2 \\
&\geq \hat{c}n^2(\ln(n+1))^3 - 3\hat{c}n(\ln(n+1))^2 \\
&\quad + 3\hat{c}(\ln(n+1)) - \hat{c}\tfrac{1}{n} + (n+1)(\ln(n+1))^3 \\
&\geq \hat{c}n^2(\ln(n+1))^3 - 3\hat{c}n(\ln(n+1))^3 \\
&\quad + 0 - \tfrac{\hat{c}}{2}(\ln(n+1))^3 + (n+1)(\ln(n+1))^3 && \text{since } n \geq 2 \text{ and } \ln(n+1) \geq 1 \\
&= \hat{c}n^2(\ln(n+1))^3 + (1 - 3\hat{c})n(\ln(n+1))^3 \\
&\quad + (1 - \tfrac{\hat{c}}{2})(\ln(n+1))^3 \\
&\geq \hat{c}n^2(\ln(n+1))^3 + 2\hat{c}(\ln(n+1))^3 \\
&\quad + \hat{c}(\ln(n+1))^3 && \textbf{if } 1 - 3\hat{c} \geq 2\hat{c} \text{ and } 1 - \tfrac{\hat{c}}{2} \geq \hat{c} \\
&= \hat{c}(n+1)^2(\ln(n+1))^3 && \text{as desired.}
\end{aligned}$$

All the constraints on $\hat{c}$ are satisfied if and only if $0 < \hat{c} \leq \frac{1}{5}$; in particular, choosing $\hat{c} = \frac{1}{5}$ will do.

It follows that

$$\sum_{i=1}^{n} i(\ln i)^3 \geq \tfrac{1}{5}n^2(\ln n)^3,$$

so that

$$\sum_{i=1}^{n} i(\ln i)^3 \in \Omega(n^2(\ln n)^3),$$

as required.

11.3 Recurrences

11.3.1 Solutions for Selected Exercises

Solution for Exercise #1(c) in Section 8.10

Derivation of a Pattern. If n is a power of two (as is given in the question), then so is $\frac{n}{2}$ provided that $n \geq 2$, and $\lfloor \frac{n}{2} \rfloor = \frac{n}{2}$ in this case. Thus, if n is a power of two and $T(n)$ is defined by the recurrence given in the question, then

$$\begin{aligned}
T(n) &= 4T\left(\tfrac{n}{2}\right) + n^3 && \text{if } n \geq 2, \\
&= 4\left(4T\left(\tfrac{n}{4}\right) + \left(\tfrac{n}{2}\right)^3\right) + n^3 && \text{if } n \geq 4, \\
&= 4^2 T\left(\tfrac{n}{4}\right) + n^3 \sum_{i=0}^{1} \left(\tfrac{1}{2}\right)^i \\
&= 4^2\left(4T\left(\tfrac{n}{8}\right) + \left(\tfrac{n}{4}\right)^3\right) + n^3 \sum_{i=0}^{1} \left(\tfrac{1}{2}\right)^i && \text{if } n \geq 8, \\
&= 4^3 T\left(\tfrac{n}{8}\right) + n^3 \sum_{i=0}^{2} \left(\tfrac{1}{2}\right)^i \cdots
\end{aligned}$$

At this point, or after you perform one or two more iterations, you might guess that if $k \geq 0$ and n is a power of two such that $n \geq 2^k$, then

$$T(n) = 4^k T\left(\tfrac{n}{2^k}\right) + n^3 \sum_{i=0}^{k-1} \left(\tfrac{1}{2}\right)^i.$$

Confirmation of a Pattern. Next, it is necessary to prove the following.

Claim. If $k \geq 0$ and n is a power of two such that $n \geq 2^k$, then

$$T(n) = 4^k T\left(\tfrac{n}{2^k}\right) + n^3 \sum_{i=0}^{k-1} \left(\tfrac{1}{2}\right)^i.$$

Proof. This will be proved by induction on k.

Basis: If $k = 0$ then

$$4^k T\left(\tfrac{n}{2^k}\right) + n^3 \sum_{i=0}^{k-1} \left(\tfrac{1}{2}\right)^i = 4^0 T\left(\tfrac{n}{2^0}\right) + n^3 \sum_{i=0}^{-1} \left(\tfrac{1}{2}\right)^i = 1 \cdot T(n) + 0 = T(n)$$

as desired.

Inductive Step: Suppose now that $k \geq 0$ and that

$$T(n) = 4^k T\left(\tfrac{n}{2^k}\right) + n^3 \sum_{i=0}^{k-1} \left(\tfrac{1}{2}\right)^i.$$

for every integer n such that n is a power of two and $n \geq 2^k$.

It is necessary (and sufficient) to prove that if n is a power of two and $n \geq 2^{k+1}$ then

$$T(n) = 4^{k+1} T\left(\tfrac{n}{2^{k+1}}\right) + n^3 \sum_{i=0}^{(k+1)-1} \left(\tfrac{1}{2}\right)^i.$$

Suppose, then, that n is a power of two and that $n \geq 2^{k+1}$. Then $n \geq 2^k$ as well, so

$$T(n) = 4^k T\left(\tfrac{n}{2^k}\right) + n^3 \sum_{i=0}^{k-1} \left(\tfrac{1}{2}\right)^i.$$

by the inductive hypothesis.

Since $n \geq 2^{k+1}$, $\frac{n}{2^k} \geq 2$, so that (using the definition of T),

$$T\left(\tfrac{n}{2^k}\right) = 4T\left(\tfrac{n}{2^{k+1}}\right) + \left(\tfrac{n}{2^k}\right)^3.$$

Therefore,

$$\begin{aligned}
T(n) &= 4^k T\left(\tfrac{n}{2^k}\right) + n^3 \sum_{i=0}^{k-1} \left(\tfrac{1}{2}\right)^i && \text{by the inductive hypothesis} \\
&= 4^k \left(4T\left(\tfrac{n}{2^{k+1}}\right) + \left(\tfrac{n}{2^k}\right)^3\right) + n^3 \sum_{i=0}^{k-1} \left(\tfrac{1}{2}\right)^i \\
&= 4^{k+1} T\left(\tfrac{n}{2^{k+1}}\right) + \tfrac{n^3}{2^k} + n^3 \sum_{i=0}^{k-1} \left(\tfrac{1}{2}\right)^i \\
&= 4^{k+1} T\left(\tfrac{n}{2^{k+1}}\right) + n^3 \sum_{i=0}^{(k+1)-1} \left(\tfrac{1}{2}\right)^i, && \text{as desired,}
\end{aligned}$$

completing the inductive step.

It therefore follows by induction on k that if $k \geq 0$, and n is a power of two such that $n \leq 2^k$, then

$$T(n) = 4^k T\left(\tfrac{n}{2^k}\right) + n^3 \sum_{i=0}^{k-1} \left(\tfrac{1}{2}\right)^i,$$

as desired. □

Application of a Pattern. Since n is a power of two, $n = 2^{\log_2 n}$ and $\log_2 n$ is a nonnegative integer, so we can choose $k = \log_2 n$ and apply the pattern that's just been verified. In this case,

$$\begin{aligned}
T(n) &= 4^{\log_2 n} T\left(\tfrac{n}{2^{\log_2 n}}\right) + n^3 \sum_{i=0}^{(\log_2 n)-1} \left(\tfrac{1}{2}\right)^i && \text{(applying the pattern)} \\
&= n^2 T(1) + n^3 \sum_{i=0}^{(\log_2 n)-1} \left(\tfrac{1}{2}\right)^i \\
&= n^2 + n^3 \left(\frac{1 - \left(\frac{1}{2}\right)^{\log_2 n}}{1 - \frac{1}{2}} \right) && \text{(using the closed form for a geometric series)} \\
&= n^2 + n^3 \left(\frac{1 - \frac{1}{n}}{\frac{1}{2}} \right) \\
&= n^2 + 2n^3 - 2n^2 \\
&= 2n^3 - n^2.
\end{aligned}$$

This isn't the only approach you could take to solve this problem. As an additional exercise, simplify the recurrence by performing a "change of variable," expressing the function in terms of $k = \log_2 n$, and then try to apply the iteration method after that. You should find that the resulting solution is simpler than the one given here.

Solution for Exercise #2(c) in Section 8.10

In order to prove that the given function $T(n)$ is nondecreasing, it is necessary and sufficient to prove the following.

Claim. If

$$T(n) = \begin{cases} 1 & \text{if } n = 1, \\ 4T\left(\lfloor \frac{n}{2} \rfloor\right) + n^3 & \text{if } n \geq 2, \end{cases}$$

then $T(n+1) \geq T(n)$ for every integer $n \geq 0$.

Proof. This will be proved by induction on n.

Basis: $T(1) = 1$ and $T(2) = 4T(1) + 8 = 12$, so it is clear that $T(n+1) \geq T(n)$ when $n = 1$.

Inductive Step: Suppose now that $n \geq 1$ and that $T(m+1) \geq T(m)$ for every integer m such that $1 \leq m \leq n$. It is necessary and sufficient to prove that $T(n+2) \geq T(n+1)$ in order to complete the inductive step.

Since $n \geq 1$, $n+1$ and $n+2$ are both greater than or equal to two, so that

$$T(n+2) = 4T\left(\lfloor \tfrac{n+2}{2} \rfloor\right) + (n+2)^3$$

and

$$T(n+1) = 4T\left(\lfloor \tfrac{n+1}{2} \rfloor\right) + (n+1)^3.$$

Furthermore, since $n \geq 1$, $1 \leq \lfloor \frac{n+1}{2} \rfloor \leq n$, and either

$$\lfloor \tfrac{n+2}{2} \rfloor = \lfloor \tfrac{n+1}{2} \rfloor$$

(if n is odd), or

$$\lfloor \tfrac{n+2}{2} \rfloor = \lfloor \tfrac{n+1}{2} \rfloor + 1$$

(if n is even). In the first case,

$$T\left(\lfloor \tfrac{n+2}{2} \rfloor\right) = T\left(\lfloor \tfrac{n+1}{2} \rfloor\right)$$

because $\lfloor \frac{n+2}{2} \rfloor = \lfloor \frac{n+1}{2} \rfloor$, and in the second case

$$T\left(\lfloor \tfrac{n+2}{2} \rfloor\right) = T\left(\lfloor \tfrac{n+1}{2} \rfloor + 1\right) \geq T\left(\lfloor \tfrac{n+1}{2} \rfloor\right)$$

by the inductive hypothesis, since $1 \leq \lfloor \frac{n+1}{2} \rfloor \leq n$.

Therefore,

$$T\left(\lfloor \tfrac{n+2}{2} \rfloor\right) \geq T\left(\lfloor \tfrac{n+1}{2} \rfloor\right)$$

in every case, so that

$$\begin{aligned}
T(n+2) - T(n+1) &= \left(4T\left(\lfloor \tfrac{n+2}{2} \rfloor\right) + (n+2)^2\right) - \left(4T\left(\lfloor \tfrac{n+1}{2} \rfloor\right) + (n+1)^2\right) \\
&= 4\left(T\left(\lfloor \tfrac{n+2}{2} \rfloor\right) - T\left(\lfloor \tfrac{n+1}{2} \rfloor\right)\right) \\
&\qquad + (n^3 + 6n^2 + 12n + 8) - (n^3 + 3n^2 + 3n + 1) \\
&\geq 4 \cdot 0 + (n^3 + 6n^2 + 12n + 8) - (n^3 + 3n^2 + 3n + 1) \qquad \text{as argued above,} \\
&= 3n^2 + 9n + 7 \\
&\geq 0, \qquad\qquad \text{since } n > 0,
\end{aligned}$$

as is required to complete the inductive step.

It therefore follows by induction on n that $T(n+1) \geq T(n)$ for every integer $n \geq 1$ if T is as defined above, and this implies that T is a nondecreasing function. □

Solution for Exercise #3(c) in Section 8.10

If

$$T(n) = \begin{cases} 1 & \text{if } n = 1, \\ 4T\left(\lfloor \frac{n}{2} \rfloor\right) + n^3 & \text{if } n \geq 2, \end{cases}$$

and $U(k)$ is defined to be $T(k-1)$, then

$$\begin{aligned}
U(k) &= T(n) && \text{for } n = k-1 \text{ (so that } k = n+1) \\
&= T(k-1) && \text{replacing references to } n \\
&= \begin{cases} 1 & \text{if } k-1 = 1, \\ 4T\left(\lfloor \frac{k-1}{2} \rfloor\right) + (k-1)^3 & \text{if } k-1 \geq 2, \end{cases} && \text{applying the recurrence for } T \\
&= \begin{cases} 1 & \text{if } k = 2, \\ 4T\left(\lfloor \frac{k+1}{2} \rfloor - 1\right) + (k-1)^3 & \text{if } k \geq 3, \end{cases} && \text{by a straightforward algebraic manipulation} \\
&= \begin{cases} 1 & \text{if } k = 2, \\ 4U\left(\lfloor \frac{k+1}{2} \rfloor\right) + (k-1)^3 & \text{if } k \geq 3, \end{cases} && \text{replacing references to } T \text{ by ones for } U.
\end{aligned}$$

Thus

$$U(n) = \begin{cases} 1 & \text{if } n = 2, \\ 4U\left(\lfloor \frac{n+1}{2} \rfloor\right) + (n-1)^3 & \text{if } n \geq 3. \end{cases}$$

Similarly, if $V(k)$ is defined to be $T(k+1)$, then

$$\begin{aligned} V(k) &= T(n) && \text{for } n = k+1 \text{ (so that } k = n-1) \\ &= T(k+1) && \text{replacing references to } n \\ &= \begin{cases} 1 & \text{if } k+1 = 1, \\ 4T\left(\lfloor \frac{k+1}{2} \rfloor\right) + (k+1)^3 & \text{if } k+1 \geq 2, \end{cases} && \text{applying the recurrence for } T \\ &= \begin{cases} 1 & \text{if } k = 0, \\ 4T\left(\lfloor \frac{k-1}{2} \rfloor + 1\right) + (k+1)^3 & \text{if } k \geq 1, \end{cases} && \text{by a straightforward manipulation} \\ &= \begin{cases} 1 & \text{if } k = 0, \\ 4V\left(\lfloor \frac{k-1}{2} \rfloor\right) + (k+1)^3 & \text{if } k \geq 1, \end{cases} && \text{replacing references to } T \text{ by ones for } V. \end{aligned}$$

Thus

$$V(n) = \begin{cases} 1 & \text{if } n = 0, \\ 4V\left(\lfloor \frac{n-1}{2} \rfloor\right) + (n+1)^3 & \text{if } n \geq 1. \end{cases}$$

Solution for Exercise #4(c) in Section 8.10

Suppose, once again, that

$$T(n) = \begin{cases} 1 & \text{if } n = 1, \\ 4T\left(\lfloor \frac{n}{2} \rfloor\right) + n^3 & \text{if } n \geq 2. \end{cases}$$

Establishing the Upper Bound. It's sufficient to prove the following in order to show that $T(n) \in O(n^3)$.

Claim. There exists a positive constant c such that $T(n) \leq cn^3$ for every integer $n \geq 1$.

Proof. This will be proved by induction on n.

Basis: If $n = 1$ then $T(n) = T(1) = 1$ and $cn^3 = c$, so that $T(n) \leq cn^3$ in this case, **provided that** $c \geq 1$.

Inductive Step: Suppose now that $n \geq 1$ and that $T(m) \leq cm^3$ for every integer m such that $1 \leq m \leq n$. It is necessary and sufficient to prove that $T(n+1) \leq c(n+1)^3$ in order to complete the proof of the inductive step.

Since $n \geq 1$, $n + 1 \geq 2$, so that

$$\begin{aligned}
T(n+1) &= 4T\left(\lfloor \tfrac{n+1}{2} \rfloor\right) + (n+1)^3 && \text{by the definition of } T \\
&\leq 4c\lfloor \tfrac{n+1}{2} \rfloor^3 + (n+1)^3 && \text{by the inductive hypothesis, since } 1 \leq \lfloor \tfrac{n+1}{2} \rfloor \leq n \\
&\leq 4c\left(\tfrac{n+1}{2}\right)^3 + (n+1)^3 \\
&= \left(\tfrac{c}{2} + 1\right)(n+1)^3 && \text{since } \left(\tfrac{n+1}{2}\right)^3 = \tfrac{(n+1)^3}{8} \\
&\leq c(n+1)^3 && \textbf{provided that } \tfrac{c}{2} + 1 \leq c,
\end{aligned}$$

as is needed to complete the inductive step.

All the constraints on c can be satisfied if $c \geq 2$; in particular, choosing $c = 2$ will work.

It therefore follows by induction on n that $T(n) \leq 2n^3$ for every integer $n \geq 1$. □

Establishing the Lower Bound. It's sufficient to prove the following claim in order to establish that $T(n) \in \Omega(n^3)$.

Claim. There exists a positive constant $\hat{c}$ such that $T(n) \geq \hat{c}n^3$ for every integer $n \geq 1$.

Proof. This will be proved by induction on n.

Basis: If $n = 1$ then $T(n) = T(1) = 1$ and $\hat{c}n^3 = \hat{c}$, so $T(n) \geq \hat{c}n^3$ in this case **provided that** $\hat{c} \leq 1$.

Inductive Step: Suppose that $n \geq 1$ and that $T(m) \geq \hat{c}m^3$ for every integer m such that $1 \leq m \leq n$. It is necessary and sufficient to prove that $T(n+1) \geq \hat{c}(n+1)^3$ as well.

Since $n \geq 1$, $n + 1 \geq 2$, so

$$\begin{aligned}
T(n+1) &= 4T\left(\lfloor \tfrac{n+1}{2} \rfloor\right) + (n+1)^3 && \text{by the definition of } T \\
&\geq 4\hat{c}\lfloor \tfrac{n+1}{2} \rfloor^3 + (n+1)^3 && \text{by the inductive hypothesis, since } 1 \leq \lfloor \tfrac{n+1}{2} \rfloor \leq n \\
&\geq 0 + (n+1)^3 && \text{since } n \geq 1, \text{ so that } \lfloor \tfrac{n+1}{2} \rfloor \geq 0 \\
&\geq \hat{c}(n+1)^3 && \textbf{provided that } \hat{c} \leq 1
\end{aligned}$$

as required to complete the inductive step.

All of the constraints on $\hat{c}$ are satisfied as long as $\hat{c} \leq 1$; in particular, it is possible to choose $\hat{c} = 1$.

It therefore follows, by induction on n, that $T(n) \geq n^3$ for every integer $n \geq 1$. □

Conclusion. Since $T(n) \in O(n^3)$ and $T(n) \in \Omega(n^3)$, $T(n) \in \Theta(n^3)$, as desired.

Solution for Exercise #5(b) in Section 8.10

In this question, you were asked to use the master theorem to find an asymptotic bound in closed form for $T(n)$, where

$$T(n) = \begin{cases} 1 & \text{if } n = 1, \\ 4T\left(\lceil \frac{n}{2} \rceil\right) + n^2 & \text{if } n \geq 2. \end{cases}$$

Solution Using the "Simplified" Master Theorem. The recurrence has the form presented in the master theorem; it corresponds to the choice of constants $a = 4$, $b = 2$, $c = 1$, and the function $f(n) = n^2$.

In addition f has the form that you need in order to apply the "simplified" version of the theorem: $f(n) \in \Theta(n^\alpha (\log n)^\beta)$, for $\alpha = 2$ and $\beta = 0$.

Now since $\log_b a = \log_2 4 = 2$, $\alpha = \log_b a$ and $\beta = 0$ in this case, so that the second case covered by the master theorem applies.

Therefore, $T(n) \in \Theta(n^{\log_b a} \log_2 n) = \Theta(n^2 \log_2 n)$.

Solution Using the Original (Corrected) Master Theorem. The previous solution is completely satisfactory. However, the problem can be solved using the original version of the master theorem too.

In order to do this you'd start by recognizing that the recurrence has the required form and identifying the constants a, b, and c, and the function $f(n)$, as above.

Once again, $\log_b a = \log_2 4 = 2$, so

$$\begin{aligned} \lim_{n \to +\infty} \frac{f(n)}{n^{\log_b a}} &= \lim_{n \to +\infty} \frac{n^2}{n^2} \\ &= \lim_{n \to +\infty} 1 \\ &= 1, \end{aligned}$$

so it follows by the limit test that $f(n) \in \Theta(n^{\log_b a})$.

Therefore the second case in the master theorem is applicable, and $T(n) \in \Theta(n^{\log_b a} \log_2 n) = \Theta(n^2 \log_2 n)$.

Solution for Exercise #6(b) in Section 8.10

In this question, you were asked to use the master theorem to find an asymptotic bound in closed form for $T(n)$, where

$$T(n) = \begin{cases} 1 & \text{if } n = 1, \\ 4T\left(\lfloor \frac{n}{2} \rfloor\right) + n^2 \log_2 n & \text{if } n \geq 2. \end{cases}$$

Attempted Solution Using the "Simplified" Version. The recurrence has the form presented in the master theorem; it corresponds to the choice of constants $a = 4$, $b = 2$, $c = 1$, and the function $f(n) = n^2 \log_2 n$.

In addition, f has the form that you need in order to try to apply the "simplified" version of the theorem: $f(n) \in \Theta(n^\alpha (\log n)^\beta)$, for $\alpha = 2$ and $\beta = 1$.

Now since $\log_b a = \log_2 4 = 2$, $\alpha = \log_b a$, so that the first and third cases of the simplified theorem are not applicable. However, $\beta \neq 0$, so that the second case covered by the simplified theorem is not applicable, either.

Therefore, the simplified version of the master theorem cannot be used to solve this problem.

Attempted Solution Using the Original Theorem. Since $a = 2$, $b = 4$, and $f(n) = n^2 \log_2 n$,

$$\begin{aligned}\lim_{n\to+\infty} \frac{f(n)}{n^{\log_b a}} &= \lim_{n\to+\infty} \frac{n^2 \log_2 n}{n^2} \\ &= \lim_{n\to+\infty} \log_2 n \\ &= +\infty.\end{aligned}$$

Therefore it follows by the limit test that $f(n) \in \omega(n^{\log_b a})$. This implies that $f(n)$ is not in $O(n^{(\log_b a)-\epsilon})$ for any positive constant, so that the first case is not applicable, and it also implies that $f(n)$ is not in $\Theta(n^{\log_b a})$, so that the second case is not applicable either.

However, let ϵ be any positive constant; then

$$\begin{aligned}\lim_{n\to+\infty} \frac{f(n)}{n^{(\log_b a)+\epsilon}} &= \lim_{n\to+\infty} \frac{n^2 \log_2 n}{n^{2+\epsilon}} \\ &= \lim_{n\to+\infty} \frac{\log_2 n}{n^{\epsilon}} \\ &= \lim_{n\to+\infty} \frac{(\log_2 e) \ln n}{n^{\epsilon}} \\ &= \lim_{n\to+\infty} \frac{(\log_2 e)/n}{\epsilon n^{\epsilon-1}} && \text{by l'Hôpital's rule} \\ &= \tfrac{\log_2 e}{\epsilon} \lim_{n\to+\infty} \frac{1}{n^{\epsilon}} \\ &= 0,\end{aligned}$$

since $\epsilon > 0$. It follows that $f(n) \in o(n^{(\log_b a)+\epsilon})$, and therefore that $f(n)$ is *not* in $\Omega(n^{(\log_b a)+\epsilon})$, for *every* positive constant ϵ.

Therefore the third case of the (original) master theorem is not applicable either.

Thus the (original version of the) master theorem cannot be applied in order to solve this problem, either.

An Optional Exercise. Use the substitution method to confirm that $T(n) \in \Theta(n^2(\log_2 n)^2)$.

Solution for Exercise #6(c) in Section 8.10

In this question you were asked to use the master theorem to find an asymptotic bound in closed form for $T(n)$, where

$$T(n) = \begin{cases} 1 & \text{if } n = 1, \\ 4T\left(\lfloor \frac{n}{2} \rfloor\right) + n^5 & \text{if } n \geq 2. \end{cases}$$

Solution Using the "Simplified" Master Theorem. The recurrence has the form presented in the master theorem; it corresponds to the choice of constants $a = 4$, $b = 2$, $c = 1$, and the function $f(n) = n^5$.

In addition, f has the form that you need in order to apply the "simplified" version of the theorem: $f(n) \in \Theta(n^\alpha (\log n)^\beta)$, for $\alpha = 5$ and $\beta = 0$.

In this case, $\log_b a = \log_2 4 = 2$, so $\alpha > \log_b a$, and the third case of the simplified master theorem is applicable.

Therefore, $T(n) \in \Theta(f(n)) = \Theta(n^5)$.

Solution Using the "Original" Master Theorem. The above proof is completely acceptable, but the problem can be solved using the original version of the master theorem too.

The argument presented below is a reasonably "short" version, in the sense that it doesn't include a use of a limit test to disqualify cases (in order to determine which case is the only one that could apply). See the solution for Exercise #6(b), above, for an example of this.

We'd begin, as above, by noting that the recurrence has the required form, and that it corresponds to the choice of constants $a = 4$, $b = 2$, $c = 1$, and the function $f(n) = n^5$.

Let $\epsilon = 1$. Then $(\log_b a) + \epsilon = (\log_2 4) + 1 = 3$, so

$$\begin{aligned}\lim_{n\to+\infty} \frac{f(n)}{n^{(\log_b a)+\epsilon}} &= \lim_{n\to+\infty} \frac{n^5}{n^3} \\ &= \lim_{n\to+\infty} n^2 \\ &= +\infty,\end{aligned}$$

and it follows by the limit test that $f(n) \in \Omega(n^{(\log_b a)+\epsilon})$.

In order to conclude that the third case of the original (corrected) master theorem is applicable, it is necessary to confirm that the "regularity" condition holds — that is, that there exists a constant d such that $d < 1$ and

$$af\left(\tfrac{n}{b}\right) \leq df(n)$$

for all sufficiently large n. Since $f(n)$ is asymptotically positive (indeed, in this particular problem, $f(n) > 0$ whenever $n > 0$), this is equivalent to the condition that

$$\frac{af\left(\frac{n}{b}\right)}{f(n)} \leq d < 1$$

for all sufficiently large n.

Now, since $a = 4$, $b = 2$, and $f(n) = n^5$, if $n > 0$ then

$$\begin{aligned}\frac{af\left(\frac{n}{b}\right)}{f(n)} &= \frac{4\left(\frac{n}{2}\right)^5}{n^5} \\ &= \frac{4\left(\frac{n^5}{32}\right)}{n^5} \\ &= \frac{\left(\frac{n^5}{8}\right)}{n^5} \\ &= \tfrac{1}{8},\end{aligned}$$

so the regularity condition is satisfied if we choose $d = \frac{1}{8}$. (It's also satisfied if we choose d to be any *other* positive constant such that $\frac{1}{8} \leq d < 1$, as well.)

Since all the required conditions for it are satisfied, the third case given in the master theorem is applicable.

Therefore, $T(n) \in \Theta(f(n)) = \Theta(n^5)$.

Solution for Exercise #7(c) in Section 8.10

In this question you were first asked to try to find an expression in closed form (which is also a function of n) that is equal to $T(n)$ for all n, for $T(n)$ as defined by the recurrence

$$T(n) = \begin{cases} 1 & \text{if } n = 0, \\ 1 & \text{if } n = 1, \\ 5 & \text{if } n = 2, \\ T(n-1) + T(n-2) - T(n-3) & \text{if } n \geq 3. \end{cases}$$

This is a linear recurrence with constant coefficients, so you should try to apply Theorem 8.9 (on page 141) to solve this problem.

The above recurrence has characteristic polynomial

$$x^3 - x^2 - x + 1$$

(note the signs of the coefficients of x^2, x, and 1!). It was given in the question that 1 is a root of this polynomial, and this implies that the polynomial is divisible by $x - 1$. Indeed, if we divide the characteristic polynomial by $x - 1$ and then use the binomial theorem to factor the resulting quotient, we will discover that

$$\begin{aligned} x^3 - x^2 - x + 1 &= (x-1)(x^2-1) \\ &= (x-1)(x-1)(x+1) \\ &= (x+1)(x-1)^2. \end{aligned}$$

Thus the characteristic polynomial has two roots, $\beta_1 = -1$ (with multiplicity $\nu_1 = 1$) and $\beta_2 = 1$ (with multiplicity $\nu_2 = 2$). According to Theorem 8.9, it follows that

$$T(n) = c_{1,0}\beta_1^n + (c_{2,1}n + c_{2,0})\beta_2^n = c_{1,0}(-1)^n + c_{2,1}n + c_{2,0} = \begin{cases} c_{2,1}n + c_{2,0} + c_{1,0} & \text{if } n \text{ is even,} \\ c_{2,1}n + c_{2,0} - c_{1,0} & \text{if } n \text{ is odd.} \end{cases}$$

Now, we must use the initial values $T(0) = 1$, $T(1) = 1$, and $T(2) = 5$ in order to find the unknown values $c_{1,0}$, $c_{2,1}$, and $c_{2,0}$. Note that

$$\begin{aligned} T(0) &= 1 = c_{1,0} + 0 \cdot c_{2,1} + c_{2,0}, \\ T(1) &= 1 = -1 \cdot c_{1,0} + c_{2,1} + c_{2,0}, \qquad \text{and} \\ T(2) &= 5 = c_{1,0} + 2 \cdot c_{2,1} + c_{2,0}. \end{aligned}$$

Thus,

$$\begin{bmatrix} 1 & 0 & 1 \\ -1 & 1 & 1 \\ 1 & 2 & 1 \end{bmatrix} \cdot \begin{bmatrix} c_{1,0} \\ c_{2,1} \\ c_{2,0} \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 5 \end{bmatrix}.$$

We will use Gaussian Elimination to solve this system. First, we add the first equation from the second, and subtract the first equation from the third, in order to obtain the system

$$\begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 2 \\ 0 & 2 & 0 \end{bmatrix} \cdot \begin{bmatrix} c_{1,0} \\ c_{2,1} \\ c_{2,0} \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 4 \end{bmatrix}.$$

We next subtract two times the second equation from the third in order to obtain the following (equivalent) triangular system:

$$\begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 2 \\ 0 & 0 & -4 \end{bmatrix} \cdot \begin{bmatrix} c_{1,0} \\ c_{2,1} \\ c_{2,0} \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 0 \end{bmatrix}.$$

Now we use "back substitution" to solve this system. The third equation is "$-4c_{2,0} = 0$," which implies that $c_{2,0} = 0$. The second equation is "$c_{2,1} + 2c_{2,0} = 2$," which now implies that $c_{2,1} = 2 - 2 \cdot 0 = 2$. Finally, the first equation is "$c_{1,0} + 0 \cdot c_{2,1} + c_{2,0} = 1$," which implies that $c_{1,0} = 1 - 0 \cdot 2 - 0 = 1$. Thus,

$$T(n) = 1 \cdot (-1)^n + 2n + 0 = \begin{cases} 2n+1 & \text{if } n \text{ is even,} \\ 2n-1 & \text{if } n \text{ is odd.} \end{cases}$$

If we made no arithmetic errors along the way, then this solution is correct. Of course, you could use mathematical induction (yet again) to confirm that the function $T(n)$ that was given in the original problem really *is* equal to the one given here, as a check for any such mistakes.

11.3.2 Solutions for Sample Tests

Solution for Sample Test in Section 8.12.1

1. (10 marks) Suppose $T(1) = 0$ and

$$T(n) = 2T(n/2) + n \log_2 n$$

for $n \geq 2$. Use "the substitution method" to prove that $T(n) \in O(n(\log_2 n)^2)$ whenever $n = 2^k$ is a power of two.

Hint: Note that $1 \leq \log_2 n$ whenever $n \geq 2$.

Solution: We will use mathematical induction (on $k = \log_2 n$) to show that $T(n) \leq cn(\log_2 n)^2$ for every power n of two, for some constant c.

Basis: If $k = 0$ then $n = 1$. Since $T(1) = 0$ and $c \cdot 1 \cdot (\log_2 1)^2 = c \cdot 1 \cdot 0 = 0$, $T(1) \leq c \cdot 1 \cdot (\log_2 1)^2$ for *all* c.

Inductive Step: Suppose now that $k > 0$, $n = 2^k$, and that $T(m) \leq cm(\log_2 m)^2$ for $m = 2^h$,

for every integer h such that $0 \leq h < k$. Now

$$\begin{aligned}
T(n) &= 2T(n/2) + n\log_2 n && \text{(since } n \geq 2 \text{ and is a power of two)} \\
&\leq 2[c(n/2)(\log_2(n/2))^2] + n\log_2 n && \text{(by the inductive hypothesis)} \\
&= cn[(\log_2 n)^2 - 2\log_2 n + 1] + n\log_2 n \\
&\leq cn[(\log_2 n)^2 - \log_2 n] + n\log_2 n && \text{(since } n \geq 2\text{, and assuming } c > 0\text{)} \\
&= cn(\log_2 n)^2 + (1-c)n\log_2 n \\
&\leq cn(\log_2 n)^2 && \text{(provided that } c \geq 1\text{).}
\end{aligned}$$

Now if we choose $c = 1$ (or, indeed, if we choose c to be any constant that is greater than or equal to 1) then c satisfies all the constraints that were identified in the inductive proof.

Therefore, $T(n) \leq cn(\log_2 n)^2$ for any constant $c \geq 1$ and for every power n of two, so $T(n) \in O(n(\log_2 n)^2)$, as desired.

2. (10 marks) Now use "the iteration method" to find an expression in closed form that is an exact solution for the recurrence given in Question 1. Once again, you may assume $n = 2^k$ is a power of two.

Hint: If $n = 2^k$ and k is a nonnegative integer then $\displaystyle\sum_{i=0}^{k} \log_2\left(\frac{n}{2^i}\right) = \sum_{j=0}^{k} \log_2(2^j) = \sum_{j=0}^{k} j$.

Note: A *small* number of bonus marks will be awarded if you also use mathematical induction to prove that your solution is correct.

Solution: If $n = 2^k \geq 1$ is a power of two then

$$\begin{aligned}
T(n) &= n\log_2 n + 2T(\tfrac{n}{2}) \\
&= n\log_2 n + 2[(\tfrac{n}{2})\log_2(\tfrac{n}{2}) + 2T(\tfrac{n}{4})] \\
&= n\log_2 n + n\log_2(\tfrac{n}{2}) + 4T(\tfrac{n}{4}) \\
&= n\log_2 n + n\log_2(\tfrac{n}{2}) + 4[\tfrac{n}{4}\log_2(\tfrac{n}{4}) + 2T(\tfrac{n}{8})] \\
&= n\log_2 n + n\log_2(\tfrac{n}{2}) + n\log_2(\tfrac{n}{4}) + 8T(\tfrac{n}{8}).
\end{aligned}$$

It is possible that a pattern is clear at this point; otherwise, one could continue to expand and collect terms a few more times, until it seemed clearer that

$$\begin{aligned}
T(n) &= n\log_2 n + n\log_2(\tfrac{n}{2}) + n\log_2(\tfrac{n}{4}) + \cdots + n\log_2(\tfrac{n}{2^i}) + 2^{i+1}T(\tfrac{n}{2^{i+1}}) \\
&= \left(\sum_{j=0}^{i} n\log_2(\tfrac{n}{2^j})\right) + 2^{i+1}T(\tfrac{n}{2^{i+1}})
\end{aligned} \tag{11.5}$$

for $0 \le i < k = \log_2 n$. In particular, when $i = k - 1$,

$$\begin{aligned}
T(n) &= \left(\sum_{j=0}^{k-1} n \log_2(\tfrac{n}{2^j})\right) + 2^k T(\tfrac{n}{2^k}) \\
&= n\left(\sum_{j=0}^{k-1} \log_2(\tfrac{n}{2^j})\right) + nT(1) \\
&= n\left(\sum_{j=0}^{k} \log_2(\tfrac{n}{2^j})\right) && \text{(since } T(1) = 0 = \log_2 1\text{)} \\
&= n\left(\sum_{i=0}^{k} \log_2(2^i)\right) = n\left(\sum_{i=0}^{k} i\right) && \text{(using the hint)} \\
&= n\left(\frac{k(k+1)}{2}\right) \\
&= \tfrac{1}{2}nk^2 + \tfrac{1}{2}nk = \tfrac{1}{2}n(\log_2 n)^2 + \tfrac{1}{2}n\log_2 n.
\end{aligned}$$

Thus (one might guess that) $T(n) = \frac{1}{2}n(\log_2 n)^2 + \frac{1}{2}n\log_2 n$ whenever n is a power of two.

Additional Material, for Bonus Marks: Up to three bonus marks will be awarded for a correct proof that the above closed form is correct — that is, for a proof that $T(n) = \frac{1}{2}n(\log_2 n)^2 + \frac{1}{2}n\log_2 n$, whenever $n \ge 1$ is a power of two.

Basis: If $n = 1$ then $T(n) = T(1) = 0$, and $\frac{1}{2}n(\log_2 n)^2 + \frac{1}{2}n\log_2 n = \frac{1}{2}\cdot 1 \cdot 0^2 + \frac{1}{2}\cdot 1 \cdot 0 = 0$ as well, so the closed form is correct when $n = 1$.

Inductive Step: Suppose now that $n \ge 2$ is a power of two, and that $T(m) = \frac{1}{2}m(\log_2 m)^2 + \frac{1}{2}m\log_2 m$ whenever m is a power of two and $1 \le m < n$. Then

$$\begin{aligned}
T(n) &= 2T(n/2) + n\log_2 n && \text{(since } n \ge 2 \text{ and } n \text{ is a power of two)} \\
&= 2[\tfrac{1}{2}(n/2)(\log_2(n/2))^2 + \tfrac{1}{2}(n/2)\log_2(n/2)] + n\log_2 n \\
& && \text{(by the inductive hypothesis)} \\
&= \tfrac{1}{2}n[(\log_2 n - 1)^2 + (\log_2 n - 1) + 2\log_2 n] && \text{(collecting terms)} \\
&= \tfrac{1}{2}n[(\log_2 n)^2 - 2\log_2 n + 1 + \log_2 n - 1 + 2\log_2 n] \\
&= \tfrac{1}{2}n[(\log_2 n)^2 + \log_2 n] \\
&= \tfrac{1}{2}n(\log_2 n)^2 + \tfrac{1}{2}n\log_2 n,
\end{aligned}$$

as desired.

Therefore, it follows by induction on n that $T(n) = \frac{1}{2}n(\log_2 n)^2 + \frac{1}{2}n\log_2 n$ whenever $n \ge 1$ and n is a power of two.

Note: The only part of the derivation that really needs to be confirmed (by proof) is the pattern given in equation (11.5). Thus, full bonus marks will also be given for a proof by induction that equation (11.5) is correct for $0 \le i < k = \log_2 n$.

Solution for Sample Test in Section 8.12.2

1. Suppose a recursive algorithm for multiplication of two n-digit integers

   - uses exactly c_1 steps, for some positive constant c_1, if $n < 3$, and
   - solves the problem when $n \geq 3$ by
     (a) multiplying together *five* pairs of smaller integers, using the same algorithm recursively, where the smaller "input" integers are always exactly $\lfloor \frac{n}{3} \rfloor$ digits long, and
     (b) doing extra work that requires exactly $c_2 n$ steps for some positive constant c_2.

   (a) (4 marks) Write down a recurrence for the number of steps used by the above multiplication algorithm (as a function of n) in the worst case.

   Solution: It follows from the description of the problem that the number of steps used when the inputs are n-digit integers is

   $$T(n) = \begin{cases} c_1 & \text{if } n < 3 \\ 5T(\lfloor \frac{n}{3} \rfloor) + c_2 n & \text{if } n \geq 3. \end{cases}$$

   (b) (3 marks) Find a function $f(n)$ in closed form such that this algorithm uses $\Theta(f(n))$ steps to multiply two n-digit integers together.

   Solution: This has the form needed for the simplified master theorem (Theorem 8.4 on page 132) to be applicable. It corresponds to the choice of constants $a = 5$, $b = 3$, $\alpha = 1$, and $\beta = 0$.
   Now, since $\alpha = 1 < \log_3 5 = \log_b a$, the first case described in this theorem is applicable, so $T(n) \in \Theta(n^{\log_b a}) = \Theta(n^{\log_3 5})$. Thus we can set $f(n) = n^{\log_3 5}$.

2. (5 marks) Write a recursive function that takes a pointer to the root of a binary tree as input, and returns the sum of the values stored at the nodes of the tree.

   Use the following notation: if p is a pointer to a node, then

   - $*p$.left is a pointer to the left child (and is "null" if the node does not have a left child);
   - $*p$.right is a pointer to the right child (and is "null" if the node does not have a right child);
   - $*p$.value is the value stored at the node.

   Solution: A recursive function sum_of_nodes, which takes a pointer to the root of the binary tree as input and has the sum as its value (that is, the value returned), is as follows.

```
function sum_of_nodes(p : ↑node)

if (p = null) then
    return 0
else
    return (*p.value + sum_of_nodes(*p.left) + sum_of_nodes(*p.right))
end if
```

3. Consider two "integer division" algorithms (that compute the quotient and remainder obtained by dividing one n-digit input integer by a second n-digit input integer):

   - Algorithm A uses $T(n)$ steps, when given n-digit integers as input, where

$$T(n) = \begin{cases} c_1 & \text{if } n < 4, \\ 7T(\lceil \frac{n}{4} \rceil) + c_2 n + c_3 & \text{if } n \geq 4, \end{cases}$$

     and where c_1, c_2, and c_3 are positive constants;
   - Algorithm B uses $\Theta(n(\log n)^2)$ steps when given n-digit integers as input.

   (a) (6 marks) Find a function $f(n)$ in closed form such that $T(n) \in \Theta(f(n))$ (where $T(n)$ is the number of steps used by Algorithm A, as above).

   Solution: Once again, the simplified master theorem is applicable — this time, with $a = 7$, $b = 4$, $\alpha = 1$, and $\beta = 0$. Since $\alpha = 1 < \log_4 7 = \log_b a$, the first case mentioned in the theorem is applicable, so that $T(n) \in \Theta(n^{\log_b a}) = \Theta(n^{\log_4 7})$. Thus we can set $f(n) = n^{\log_4 7}$.

   (b) (2 marks) Say *which* of the above two algorithms is asymptotically faster — that is, say which algorithm uses a smaller number of steps, when given n-digit integers as inputs, for all sufficiently large n.

   Solution: Since $(\log n)^2 \in o(n^c)$ for any constant $c > 0$, $(\log n)^2 \in o(n^{(\log_4 7)-1})$, and $n(\log n)^2 \in o(n^{\log_4 7})$. Thus, algorithm B is asymptotically faster than algorithm A.

Solution for Sample Test in Section 8.12.3

1. This question concerns the definition of *recurrences*.

   (a) (1 mark) State the definition of a *recurrence.*

   Answer: A recurrence is an equation or inequality that describes a function in terms of its values on smaller inputs.

   (b) (2 marks) Is the following a recurrence? Why, or why not?

$$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ \frac{1}{2}\left(T(n+1) + T(n-1)\right) & \text{if } n \geq 2. \end{cases}$$

   Be specific in your answer for the last part: It isn't enough just to say that it does, or does not, satisfy the definition you gave above.

   Answer: *No,* this is *not* a recurrence, because its description of $T(n)$ (for the case $n \geq 1$) refers to the value $T(n+1)$ of the function at a *larger* value, $n+1$, as well as smaller values.

2. Suppose now that

$$T(n) = \begin{cases} 0 & \text{if } n = 1, \\ 9T\left(\lfloor \frac{n}{3} \rfloor\right) + n^4 & \text{if } n > 1. \end{cases}$$

(a) (8 marks) Use the substitution method — **not** the master theorem — to prove that

$$T(n) \in O(n^4)$$

assuming that n is a power of three (so that $n = 3^h$ for some integer $h \geq 0$).

Solution:

Claim: There exists a constant $c > 0$ such that $T(n) \leq cn^4$ whenever $n \geq 1$ and n is a power of three.

Proof of Claim: By induction on $h = \log_3 n$.

Basis ($h = 0$): If $h = 0$ then $n = 3^h = 1$, so that $T(n) = T(1) = 0$ and $cn^4 = c$. It follows that $T(n) \leq cn^4$ in this case, for **any** constant c such that $c > 0$.

Inductive Step: Suppose now that $h \geq 0$ and that $T(n) \leq cn^4$ for $n = 3^h$. It is necessary and sufficient to prove, using only this assumption, that $T(\hat{n}) \leq c\hat{n}^4$ as well, for $\hat{n} = 3^{h+1}$. Since $h \geq 0$, $\hat{n} = 3n \geq 3$, so

$$\begin{aligned}
T(\hat{n}) &= 9T\left(\lfloor \tfrac{\hat{n}}{3} \rfloor\right) + \hat{n}^4 && \text{by the definition of } T\text{, since } \hat{n} > 1 \\
&= 9T(n) + \hat{n}^4 && \text{since } \hat{n} = 3n \\
&\leq 9cn^4 + \hat{n}^4 && \text{by the inductive hypothesis (stated above)} \\
&= \left(\tfrac{c}{9} + 1\right)\hat{n}^4 && \text{since } \hat{n} = 3n\text{, so that } \hat{n}^4 = 81n^4 \\
&\leq c\hat{n}^4 && \textbf{provided that } \tfrac{c}{9} + 1 \leq c\text{, that is, } c \geq \tfrac{9}{8}
\end{aligned}$$

as required to complete the inductive hypothesis.

All the constraints that were identified for c in the above proof are satisfied as long as $c \geq \frac{9}{8}$; in particular, choosing $c = \frac{9}{8}$ will work.

It therefore follows (by induction on h) that the claim is correct.

Clearly, since $T(n) \leq \frac{9}{8}n^4$ whenever n is a power of three, $T(n) \in O(n^4)$ whenever n is a power of three, as desired.

(b) (2 marks) Now show that

$$T(n) \in \Omega(n^4)$$

as well, assuming that $T(n) \geq 0$ for every integer $n \geq 1$.

As the marks for this part should suggest, this should be *much* easier to do than the first part.

Solution: Since $T(m) \geq 0$ for every integer $m \geq 1$, if $n \geq 3$ then

$$\begin{aligned}
T(n) &= 9T\left(\lfloor \tfrac{n}{3} \rfloor\right) + n^4 && \text{by the definition of } T \\
&\geq 0 + n^4 && \text{by the above assumption} \\
&= n^4.
\end{aligned}$$

It follows that $T(n) \in \Omega(n^4)$.

3. Consider, once again, the recurrence used in Question #2 and the function $T(n)$ that it defines.

   (a) (5 marks) If you used the "iteration method" then you might make the following conjecture.

   Claim: If $k \geq 0$ and n is any power of three such that $n \geq 3^k$, then

   $$T(n) = 9^k T\left(\tfrac{n}{3^k}\right) + n^4 \sum_{i=0}^{k-1} \left(\tfrac{1}{9}\right)^i.$$

   Prove that this claim is correct.

   Proof: By induction on k.

   Basis ($k = 0$): If $k = 0$ then

   $$9^k T\left(\tfrac{n}{3^k}\right) + n^4 \sum_{i=1}^{k-1} \left(\tfrac{1}{9}\right)^i = 9^0 T\left(\tfrac{n}{3^0}\right) + n^4 \cdot \sum_{i=1}^{-1} \left(\tfrac{1}{9}\right)^i = 1 \cdot T(n) + n^4 \cdot 0 = T(n),$$

   as required.

   Inductive Step: Suppose, now, that $k \geq 0$ and that

   $$T(n) = 9^k T\left(\tfrac{n}{3^k}\right) + n^4 \sum_{i=0}^{k-1} \left(\tfrac{1}{9}\right)^i$$

   for every power n of three such that $n \geq 3^k$. It is necessary and sufficient to prove, using only the above assumption, that

   $$T(n) = 9^{k+1} T\left(\tfrac{n}{3^{k+1}}\right) + n^4 \sum_{i=0}^{(k+1)-1} \left(\tfrac{1}{9}\right)^i$$

   for every power n of three such that $n \geq 3^{k+1}$.
   Suppose, then, that $n = 3^h$ is a power of three and that $n \geq 3^{k+1}$ (in other words, h is an integer and $h \geq k + 1$).
   Then $n \geq 3^k$ as well, so

   $$\begin{aligned}
   T(n) &= 9^k T\left(\tfrac{n}{3^k}\right) + n^4 \sum_{i=0}^{k-1} \left(\tfrac{1}{9}\right)^i && \text{by the above inductive hypothesis} \\
   &= 9^k \left(9T\left(\tfrac{n}{3^{k+1}}\right) + \left(\tfrac{n}{3^k}\right)^4\right) + n^4 \sum_{i=0}^{k-1} \left(\tfrac{1}{9}\right)^i && \text{since } \tfrac{n}{3^k} \text{ is a power of three and } \tfrac{n}{3^k} \geq 3 \\
   &= 9^{k+1} T\left(\tfrac{n}{3^{k+1}}\right) + \tfrac{n^4}{9^k} + \sum_{i=0}^{k-1} \left(\tfrac{1}{9}\right)^i \\
   &= 9^{k+1} T\left(\tfrac{n}{3^{k+1}}\right) + n^4 \sum_{i=0}^{(k+1)-1} \left(\tfrac{1}{9}\right)^i, && \text{as desired.}
   \end{aligned}$$

   It therefore follows by induction on k that the claim is correct.

(b) (3 marks) Now, use the claim to find an exact closed form for $T(n)$ when n is a power of three.

Solution: Since we wish to eliminate references to the function T on the right hand side, we should choose k so that the value of $T\left(\frac{n}{3^k}\right)$ is given explicitly in the recurrence for T; that is, we should choose k so that

$$\tfrac{n}{3^k} = 1.$$

Solving for k, we would find that $k = \log_3 n$.
Clearly — if n is a power of three and $n \geq 1$ — this implies that $k \geq 0$ and that $n = 3^k \geq 3^k$, so that the claim implies that

$$\begin{aligned}
T(n) &= 9^k T\left(\tfrac{n}{3^k}\right) + n^4 \sum_{i=0}^{k-1} \left(\tfrac{1}{9}\right)^i \\
&= 9^{\log_3 n} T\left(\tfrac{n}{3^{\log_3 n}}\right) + n^4 \sum_{i=0}^{(\log_3 n)-1} \left(\tfrac{1}{9}\right)^i \\
&= n^2 T(1) + n^4 \left(\frac{1 - \left(\frac{1}{9}\right)^{\log_3 n}}{1 - \frac{1}{9}}\right) \\
&= n^2 \cdot 0 + n^4 \left(\frac{1 - \frac{1}{n^2}}{1 - \frac{1}{9}}\right) \\
&= \tfrac{9}{8} n^4 - \tfrac{9}{8} n^2.
\end{aligned}$$

4. (4 marks) Perform a "change of variable" to write a recurrence for the function $U(h) = T(n)$, where $h = \log_3 n$ (and n is a power of three, so that h is an integer), and T is as defined in Question #2. For your convenience, here is the recurrence for T once again:

$$T(n) = \begin{cases} 0 & \text{if } n = 1, \\ 9T\left(\lfloor \frac{n}{3} \rfloor\right) + n^4 & \text{if } n > 1, \end{cases}$$

Solution:

$$\begin{aligned}
U(h) &= T(n) && \text{for } n = 3^h \text{ and } h = \log_3 n \\
&= T(3^h) \\
&= \begin{cases} 0 & \text{if } 3^h = 1, \\ 9T\left(\lfloor \frac{3^h}{3} \rfloor\right) + \left(3^h\right)^4 & \text{if } 3^h > 1, \end{cases} \\
&= \begin{cases} 0 & \text{if } h = 0, \\ 9T\left(3^{h-1}\right) + 81^h & \text{if } h > 0, \end{cases} \\
&= \begin{cases} 0 & \text{if } h = 0, \\ 9U(h-1) + 81^h & \text{if } h > 0. \end{cases}
\end{aligned}$$

Solution for Sample Test in Section 8.12.4

Several definitions and facts were given in the test and are used in the following sections; please refer to Section 8.12.4 for this material.

1. *(10 marks)* Write pseudocode for a recursive procedure *buildHeap* which takes a pointer to an "almost heap" as input and turns this "almost heap" into a heap.

   Your procedure should work from "the bottom up" in the sense that it should recursively turn the left and right subtrees of the root into heaps before it does anything else (unless, of course, the input heap has size zero).

   Your procedure can, and should, use the procedure *setRoot* as a subroutine.

   If p is a pointer to a node, please use *p.left* and *p.right* to refer to pointers to the left and right children of that node, respectively.

   Solution:

   procedure *buildHeap*(p)
   buildHeap$(p.left)$
   buildHeap$(p.right)$
   setRoot(p)
   end procedure

2. *(5 marks)* Write a recurrence for the running time $T(n)$ used by your procedure when it is given a pointer to an "almost heap" of size n as input. You should use the unit cost criterion and you should **assume** that $n = 2^k - 1$ for an integer k (that is, your recurrence only needs to be correct for integers n with this form).

   The following additional facts will be useful (and "almost true,") and you may use them without proving them:

   - If $k > 1$ and $n = 2^k - 1$ then the left and right subtrees of the root (of an "almost heap" of size n) are both "almost heaps" with size $2^{k-1} - 1 = \lfloor \frac{n}{2} \rfloor$.
   - When you call *setRoot* with a pointer to an "almost heap" of size m as input for $m \geq 1$, it uses exactly $c_1 \lceil \log_2 m \rceil + c_0$ steps before it terminates, for some positive constants c_1 and c_0 (and, it uses c_0 steps if $m = 0$).

   Solution:

   $$T(n) = \begin{cases} c_2 & \text{if } n < 3 \\ 2T\left(\lfloor \frac{n}{2} \rfloor\right) + c_1 \lceil \log_2 n \rceil + c_0 & \text{if } n \geq 3 \end{cases}$$

   in the special case that $n = 2^k - 1$ for an integer k, and for a positive constant c_2 and the constants c_0 and c_1 mentioned in the above "facts."

   It should be clear that $T(n)$ is *less than or equal to* the recursive expression on the right hand side, above. It's less obvious, but also true, that $T(n)$ is *equal* to an expression in this form, because the left and right subheaps can be "arbitrary" (or "any") subheaps of the given size.

3. *(5 marks)* Use the master theorem to prove that $T(n) \in \Theta(n)$ if $T(n)$ satisfies the recurrence you gave to answer the previous question.

   Solution: The above recurrence has the form that is considered in the master theorem and corresponds to the choice of constants $a = c_2$, $b = 2$, $c = 2$, and the function $f(n) = \lceil \log_2 n \rceil + c_0$. Furthermore, the simplified version of the master theorem can be used, because $f(n) \in \Theta(n^\alpha (\log_2 n)^\beta)$ where $\alpha = 0$ and $\beta = 1$.

   In this case $\log_b a = \log_2 2 = 1 > \alpha$, so the first case covered by the (simplified) version of the master theorem applies. Therefore

$$T(n) \in \Theta(n^{\log_b a}) = \Theta(n).$$

4. *(5 marks)* Finally, use the result you were supposed to prove in the previous question (whether you managed to prove it or not!), and the **assumption** that the running time used by your procedure is a nondecreasing function of n, to prove that this running time is linear in n **in general**, and not just in the special case that $n = 2^k - 1$ for some integer k.

   Solution: It follows from the above result that there is some constant N, and positive constants d_0 and d_1, such that if $n = 2^k - 1$ for any integer k, and $n \geq N$, then $d_0 n \leq n \leq d_1 n$.

   Now let $\hat{N}$ be the smallest integer such that $\hat{N} = 2^k - 1$ for some integer k and $\hat{N} \geq N$.

   Suppose n is an integer such that $n \geq \hat{N}$. Then there exist integers $\hat{h}$ and $\hat{k}$ such that $\hat{h} = 2^{\hat{k}} - 1$ and $\hat{N} \leq \hat{n} \leq n < 2\hat{n}$.

   Since $\hat{n} \geq \hat{N}$ and $2\hat{n} \geq \hat{N}$, $T(\hat{n}) \geq d_0 \hat{n}$ and $T(2\hat{n}) \leq d_1(2\hat{n})$. Furthermore it is given that T is a nondecreasing function.

   Therefore,

$$\begin{aligned} T(n) &\geq T(\hat{n}) \\ &\geq d_0 \hat{n} \\ &\geq \tfrac{d_0}{2} n \\ &= \hat{d}_0 n \qquad \text{where } \hat{d}_0 = \tfrac{d_0}{2} \text{ is a positive constant,} \end{aligned}$$

   and

$$\begin{aligned} T(n) &\leq T(2\hat{n}) \\ &\leq d_1(2\hat{n}) \\ &\leq 2d_1 n \\ &= \hat{d}_1 n \qquad \text{where } \hat{d}_1 = 2d_1 \text{ is a positive constant.} \end{aligned}$$

   Therefore $\hat{d}_0 n \leq T(n) \leq \hat{d}_1 n$ for every integer $n \geq \hat{N}$, so $T(n) \in \Theta(n)$, as desired.

11.4 Midterm Tests

11.4.1 Solution for Sample Test in Section 9.1

1. *(5 marks — 1 mark for each part)* Say whether each of the following claims about functions of n is **true** or **false**. You do not need to prove your answers.

   (i) $\left(\frac{3}{2}\right)^n \in \Theta(2^n)$ **Answer: false**

   (ii) $n \log_2 n \in \omega(n)$ **Answer: true**

   (iii) $\frac{1}{1000} n^{1/10} \in O((\log_2 n)^{100})$ **Answer: false**

   (iv) $\frac{1}{1000} n^{1/10} \in O(n^{100})$ **Answer: true**

   (v) $n^2 + 2n \in \Theta(2n^2 + \sqrt{n})$ **Answer: true**

2. *(10 marks)* Let $f(n) = e^n$ and let $g(n) = n$ for $n \geq 0$. Prove that $f(n) \in \omega(g(n))$.

   Solution: Since $f'(n) = e^n$, $g'(n) = 1$, and $\lim_{n \to +\infty} f(n) = \lim_{n \to +\infty} g(n) = +\infty$,

   $$\begin{aligned} \lim_{n \to +\infty} \frac{f(n)}{g(n)} &= \lim_{n \to +\infty} \frac{f'(n)}{g'(n)} && \text{(by l'Hôpital's rule)} \\ &= \lim_{n \to +\infty} \frac{e^n}{1} \\ &= +\infty. && \text{(since } e > 1\text{)} \end{aligned}$$

   It follows that $f(n) \in \omega(g(n))$, as desired.

3. *(10 marks)* Let $S_n = \sum_{i=1}^{n} 13 i^{5.5} = \sum_{i=1}^{n} 13 i^{11/2}$.

   Find a function $f(n)$ in closed form such that $|f(n) - S_n| \in O(n^6)$, and prove that this inequality holds for your choice of the function f.

   Generous part marks will be given if you only find a function $f(n)$ in closed form such that $S_n \in \Theta(f(n))$ (and prove that *this* relationship holds), instead.

   You may use the following facts without proving them:

   (a) If $g(x) = 13x^{5.5}$, then $g(x)$ is an *increasing* function of x (over the positive real numbers).
   (b) If $r > 1$ then $((n+1)^r - n^r) \in O(n^{r-1})$.

Solution, for Full Marks:

Let $g(x) = 13x^{11/2}$, so that $S_n = \sum_{i=1}^{n} g(i)$, and let $f(x) = 2x^{6.5} = 2x^{13/2}$. Then $f'(x) = g(x)$, and it is given that $g(x)$ is an increasing function of x (over the positive real numbers), so that S_n can be approximated by integrals:

$$\int_0^n g(t)\,dt \quad \leq \quad S_n = \sum_{i=1}^{n} g(i) \quad \leq \quad \int_1^{n+1} g(t)\,dt.$$

Since $f'(x) = g(x)$, the lower bound given in the above equation is

$$\int_0^n g(t)\,dt = f(n) - f(0) = 2n^{13/2} - 2 \cdot 0^{13/2} = 2n^{13/2},$$

while the upper bound is

$$\int_1^{n+1} g(t)\,dt = f(n+1) - f(1) = 2(n+1)^{13/2} - 2.$$

Now,

$$\begin{aligned}
|f(n) - S_n| &= S_n - f(n) \\
&\leq (f(n+1) - f(1)) - f(n) \\
&= 2(n+1)^{13/2} - 2 - 2n^{13/2} \\
&< 2((n+1)^{13/2} - n^{13/2}) \\
&\in O(n^{11/2}) && \text{(since it is given that } (n+1)^{13/2} - n^{13/2} \in O(n^{11/2})) \\
&\subseteq O(n^6) && \text{(since } 11/2 \leq 6.)
\end{aligned}$$

Thus, if $f(n) = 2n^{13/2}$, then $f(n)$ satisfies all the conditions given in the question.

Less Satisfactory Solution, Worth "Generous Part Marks:"

A looser approximation of the sum can be obtained by "splitting the sum" and "bounding terms."

First, note that since $13 = g(1) \leq g(i) \leq g(n) = 13n^{11/2}$ if $1 \leq i \leq n$,

$$S_n = \sum_{i=1}^{n} g(i) \leq \sum_{i=1}^{n} g(n) = ng(n) = 13n^{13/2}.$$

This implies that $S(n) \in O(n^{13/2})$. In order to show that $S(n) \in \Omega(n^{13/2})$, as well, note that

$$S_n = S_L + S_U, \qquad \text{for} \quad S_L = \sum_{i=1}^{\lfloor n/2 \rfloor} g(i), \quad \text{and} \quad S_U = \sum_{\lfloor n/2 \rfloor + 1}^{n} g(i).$$

Let $m_L = \min_{1 \le i \le \lfloor n/2 \rfloor} g(i)$. Then $m_L = g(1) = 13$, and (since $n > 0$ and $m_L > 0$)

$$\begin{aligned} S_L &= \sum_{i=1}^{\lfloor n/2 \rfloor} g(i) \\ &\ge \sum_{i=1}^{\lfloor n/2 \rfloor} m_L \\ &= \left\lfloor \frac{n}{2} \right\rfloor \cdot m_L \\ &= \left\lfloor \frac{n}{2} \right\rfloor \cdot 13 \\ &\ge 13 \left(\frac{n}{2} - 1 \right) = \frac{13}{2} n - 13. \end{aligned}$$

Now, let $m_U = \min_{\lfloor n/2 \rfloor + 1 \le i \le n} g(i)$. Then $m_U = g(\lfloor n/2 \rfloor + 1) = 13(\lfloor n/2 + 1 \rfloor)^{11/2} \ge 13(n/2)^{11/2}$, and (since $n > 0$ and $m_U > 0$)

$$\begin{aligned} S_U &= \sum_{i=\lfloor n/2 \rfloor + 1}^{n} g(i) \\ &\ge \sum_{i=\lfloor n/2 \rfloor + 1}^{n} m_U \\ &= \left(n - \left\lfloor \frac{n}{2} \right\rfloor \right) m_U \\ &= \left\lceil \frac{n}{2} \right\rceil m_U \\ &\ge \left(\frac{n}{2} \right) \cdot 13 \cdot \left(\frac{n}{2} \right)^{11/2} \\ &\frac{13}{2^{13/2}} \cdot n^{13/2}. \end{aligned}$$

Therefore,

$$S_n = S_L + S_U \ge \frac{13}{2} n - 13 + \frac{13}{2^{13/2}} \cdot n^{13/2} \in \Omega(n^{13/2}).$$

Since $S_n \in O(n^{13/2})$ and $S_n \in \Omega(n^{13/2})$, $S_n \in \Theta(n^{13/2})$ (and we can set $f(n) = n^{13/2}$ in order to earn "generous part marks" for this question).

4. *(5 marks)* Find a function $f(n)$ in closed form such that $T(n) \in \Theta(f(n))$, where

$$T(n) = \begin{cases} 1 & \text{if } n = 1, \\ 16T\left(\lceil n/2 \rceil\right) + n^3 & \text{if } n \ge 2, \end{cases}$$

and say **briefly** how you obtained your answer.

Solution: This recurrence has the form

$$T(n) = \begin{cases} c_0 & \text{if } n < a, \\ aT(\lceil n/b \rceil) + g(n) & \text{if } n \geq a, \end{cases}$$

where c_0, a, and b are positive constants and where $g(n)$ is a function of n in closed form (in particular, where $c_0 = 1$, $a = 16$, $b = 2$, and $g(n) = n^3$). Thus, it is possible that "the master theorem" can be used to find an asymptotic approximation for $T(n)$. Furthermore, since $g(n) \in \Theta(n^\alpha (\log n)^\beta)$ for $\alpha = 3$ and $\beta = 0$, the "simplified master theorem" can be used.

Since $\alpha = 3 < 4 = \log_2 16 = \log_b a$, the first cased described in this theorem is applicable, and this implies that $T(n) \in \Theta(n^{\log_b a}) = \Theta(n^4)$.

Thus, the master theorem implies that if $f(n) = n^4$ then $T(n) \in \Theta(f(n))$.

Comments on Alternative Solutions: "In principle," you could use other methods from CPSC 413 to solve this problem, but this would not be nearly as easy to do as it was to apply the master theorem:

It is possible to use "the iteration method" to establish that $T(n) = 2n^4 - n^3$, whenever n is a power of two. However, it would be necessary to prove, somehow, that $T(n) \in \Theta(n^4)$ for all other positive integers n as well, before you could correctly conclude that "$T(n) \in \Theta(n^4)$" (*and* consider this to have been proved). In particular, if you concluded that $T(n) \in \Theta(n^4)$ after having obtained the closed form for powers of two (and without doing anything else), then your answer was incomplete.

It should be noted as well that it is necessary to "guess a pattern" when using the iteration method, in order to express the recurrence as a summation. In order to give a complete "proof" of correctness of the resulting closed form — even just for powers of two — either the "pattern" you guessed, or the closed form you obtained from it, must be verified (probably, using induction on $k = \log_2 n$).

Now, suppose that you got as far as obtaining a closed form for $T(n)$ when n is a power of two (and proving that this is correct).

One way to complete a proof that $T(n) \in \Theta(n^4)$ is to show (by induction) that $T(n)$ is an *increasing* function — that is, if m and n are integers and $1 \leq m \leq n$, then $T(m) \leq T(n)$. This proof by induction would not be very difficult (but, the special case $n = 2$ would need to be treated carefully). At that point, you could note that if $n \geq 2$ and $k = \lfloor \log_2 n \rfloor$, then $n/2 < 2^k \leq n < 2^{k+1} \leq 2n$, 2^k and 2^{k+1} are powers of two, and

$$(2 \cdot 2^{4k} - 2^{3k}) = T(2^k) \leq T(n) \leq T(2^{k+1}) = (2 \cdot 2^{4(k+1)} - 2^{3(k+1)}).$$

Using *this,* you could establish (for example) that $\frac{1}{32} n^4 \leq T(n) \leq 16n^4$ for $n \geq 32$, which would complete the proof that $T(n) \in \Theta(n^4)$.

Another way to complete the proof (after finding the closed form for $T(n)$ when n is a power of two) would be to "guess" that $T(n) \in \Theta(n^4)$ in general and then try to use "the substitution method" to confirm this. You would end using the substitution method twice — once, to prove that $T(n) \in O(n^4)$, and then again, to prove that $T(n) \in \Omega(n^4)$. The application of the substitution method to prove that $T(n) \in \Omega(n^4)$ is straightforward. Unfortunately, the application of this method to show that $T(n) \in O(n^4)$ would be more difficult: A first attempt

to prove that $T(n) \in \alpha n^4$ for some unknown value α would fail, because it would impossible to prove the "inductive step" required. A second attempt, to prove that $T(n) \leq \alpha n^4 - \beta n^3$ *for all* $n \geq 1$, would also fail, because you would not be able to find constants α and β satisfying all the constraints obtained during the attempt.

Now, note that it would be good enough to confirm that $T(n) \leq \alpha n^4 - \beta n^3$ "for all $n \geq n_0$," for some positive constant n_0. In order to prove this, you could try to choose a suitable integer n_0, and then

(a) confirm that the inequality is satisfied for the "special cases" $n = n_0,\ n_0 + 1,\ n_0 + 2, \ldots, 2n_0 - 2$ (in the "basis" part of your proof); and, then,

(b) try to prove that the inequality is also satisfied for $n \geq 2n_0 - 1$ in the "inductive step."

You would end up using the recursive definition for $T(n)$ in the inductive step; now, provided that the above values were all checked in the "basis," you could safely assume that $T(\lceil n/2 \rceil) \leq \alpha(\lceil n/2 \rceil)^4 - \beta(\lceil n/2 \rceil)^3$ in the inductive step when you needed to.

If you pick n_0 to be too large, then there will so many special cases that need to be checked in the basis that you would not have time to finish. However, if you pick n_0 to be too *small*, then you will discover that the constraints you've identified for α and β can't all be satisfied.

At this point, I have not completed a proof along these lines. However, I *suspect* that this can be made to work if you choose $n_0 = 5$, and that it won't work if n_0 is chosen to be smaller than that.

In any case, this would take *much* more time than you could afford to spend on a single midterm question, and it was *expected* that you would apply the master theorem, and write down the solution (almost) immediately after that, instead.

5. *(10 marks)* Find a closed form for the function $T(n)$, where

$$T(n) = \begin{cases} 0 & \text{if } n = 0, \\ 1 & \text{if } n = 1, \\ 2T(n-1) - T(n-2) & \text{if } n \geq 2, \end{cases}$$

and prove that your answer is correct. You may prove that your answer is correct either by applying one of the theorems stated in class or by using mathematical induction.

There are at least three different ways to solve this problem, using techniques introduced in CPSC 413. Of these three, the third is most likely the one to be attempted by students who worked on Problem Set #5, and is the most reliable, in the sense that it is the only one that does not require you to guess a pattern after examining a finite number of cases (and then use induction to prove that your guess is correct).

Solution #1: In this solution, we will compute $T(0)$, $T(1)$, $T(2)$, $T(3)$, and so on, until a pattern becomes clear. We will then guess a solution and then try to use mathematical induction in order to prove that it is correct.

Now,

$$\begin{aligned}
&T(0) = 0,\\
&T(1) = 1,\\
&T(2) = 2T(1) - T(0) = 2 \cdot 1 - 0 = 2,\\
&T(3) = 2T(2) - T(1) = 2 \cdot 2 - 1 = 3,\\
&T(4) = 2T(3) - T(2) = 2 \cdot 3 - 2 = 4,
\end{aligned}$$

and it would seem to be reasonable to conjecture at this point that $T(n) = n$ for every integer $n \geq 0$. In fact, this is correct, as will be shown below by induction on n:

Basis: It is given that $T(0) = 0$ and $T(1) = 1$, so $T(n) = n$ for $n = 0$ and $n = 1$.

Inductive Step: Now suppose that $n \geq 2$ and that $T(m) = m$ for every integer m such that $0 \leq m \leq n - 1$. Then

$$\begin{aligned}
T(n) &= 2T(n-1) - T(n-2) && \text{(by definition, since } n \geq 2)\\
&= 2(n-1) - (n-2) && \text{(by the inductive hypothesis)}\\
&= 2n - 2 - n + 2 = n. && \text{(as desired)}
\end{aligned}$$

It follows by induction on n that $T(n) = n$ for every integer $n \geq 0$.

Solution #2: In this solution, we will try to use "the iteration method." While we won't end up expressing the recursively defined function $T(n)$ as a summation, we *will* find a way to express $T(n)$ in terms of $T(1)$ and $T(0)$, and this will allow us to find a closed form for $T(n)$: For large n,

$$\begin{aligned}
T(n) &= 2T(n-1) - T(n-2)\\
&= 2(2T(n-2) - T(n-3)) - T(n-2)\\
&= 3T(n-2) - 2T(n-3)\\
&= 3(2T(n-3) - T(n-4)) - 2T(n-3)\\
&= 4T(n-3) - 3T(n-4)\\
&= 4(2T(n-4) - T(n-5)) - 3T(n-4)\\
&= 5T(n-4) - 4T(n-5).
\end{aligned}$$

At this point, it might seem reasonable to conjecture that "for every integer i such that $1 \leq i \leq n - 1$,

$$T(n) = (i+1)T(n-i) - iT(n-i-1)."$$

In fact, this conjecture is correct. We will prove this using induction on i.

Basis: If $i = 1$ and $1 \leq i \leq n - 1$, then $n \geq 2$, so that

$$\begin{aligned}
T(n) &= 2T(n-1) - T(n-2) && \text{(by definition, since } n \geq 2)\\
&= (i+1)T(n-i) - iT(n-i-1). && \text{(since } i = 1)
\end{aligned}$$

On the other hand, if $i = 1$ and i is *not* between 1 and $n-1$ then the claim is also true (since "false implies anything"). Thus, the result is correct if $i = 1$.

Inductive Step: Suppose now that $i \geq 2$ and that, for every integer j such that $1 \leq j \leq i-1$, the claim holds — that is, "if $1 \leq j \leq n-1$ then

$$T(n) = (j+1)T(n-j) - jT(n-j-1)."$$

Now, the claim is trivially true if i is not less than or equal to $n-1$ (again, since "false implies anything,"), so we may assume that $2 \leq i \leq n-1$. In this case, $1 \leq i-1 \leq n-2 < n-1$, so

$$\begin{aligned}
T(n) &= iT(n-(i-1)) - (i-1)T(n-(i-1)-1) && \text{(by the inductive hypothesis)} \\
&= iT(n-i+1) - (i-1)T(n-i) \\
&= i(2T(n-i) - T(n-i-1)) - (i-1)T(n-i) && \text{(by definition, since } n-i+1 \geq 2) \\
&= (2i-i+1)T(n-i) - iT(n-i-1) && \text{(reordering terms)} \\
&= (i+1)T(n-i) - iT(n-i-1). && \text{(as desired)}
\end{aligned}$$

It follows by induction on i that if $1 \leq i \leq n-1$ then $T(n) = (i+1)T(n-i) - iT(n-i-1)$.

In particular, when $n \geq 2$ and $i = n-1$, then

$$T(n) = ((n-1)+1)T(n-(n-1)) - (n-1)T(n-(n-1)-1) = nT(1) - (n-1)T(0) = n.$$

Since $T(n) = n$ when $n = 0$ and when $n = 1$ as well, it follows that $T(n) = n$ for every integer $n \geq 0$.

Solution #3: Now, note that the given recurrence is a "linear recurrence with constant coefficients." The characteristic polynomial of this recurrence is

$$x^2 - 2x + 1 = (x-1)^2,$$

so it has a single root, $\beta_1 = 1$, with multiplicity $\nu_1 = 2$.

By a theorem given in class, it follows that the function $T(n)$ has a closed form

$$T(n) = (c_1 n + c_0)\beta_1^n = c_1 n + c_0$$

for unknowns c_1 and c_0. This equation holds when $n = 0$ and when $n = 1$, so that

$$0 \cdot c_1 + c_0 = T(0) = 0,$$

and

$$1 \cdot c_1 + c_0 = T(1) = 1.$$

The first of these equations implies that $c_0 = 0$, and the second equation then implies that $c_1 = 1$. Therefore,

$$T(n) = n \qquad \text{for every integer } n \geq 0.$$

11.4.2 Solution for Sample Test in Section 9.2

1. *(5 marks — 1 mark for each part)* Say whether each of the following claims about functions of n is **true** or **false**. You do not need to prove your answers.

   (i) $2^n \in \omega(n^{10})$ **Answer: True**

   (ii) $\frac{1}{1000}(\log n)^{10} \in O(n^{1/100})$ **Answer: True**

   (iii) $n \log_2 n \in \Omega(n)$ **Answer: True**

   (iv) $n^3 + 2n^2 \in \Theta(2n^3 + \sqrt{n})$ **Answer: True**

   (v) $1000n^{1/10} \in \Omega(n^{100})$ **Answer: False**

   Comment: Parts (a) and (b), above, could have been answered by recalling and applying the "rule of thumb" that any fixed power of a logarithm grows asymptotically strictly more slowly than any fixed power of n, provided that the *exponent* in the power of n is strictly greater than zero. Similarly, part (e) could be answered using the rule of thumb that n^{c_1} grows asymptotically strictly more slowly than n^{c_2} whenever c_1 and c_2 are constants and $c_1 < c_2$. The remaining parts could be decided and answered by using limit tests.

2. *(8 marks)* Let $f(n) = 3n^2 + \ln n$ and let $g(n) = n^2$ for $n \geq 1$. Prove that $f(n) \in \Theta(g(n))$.

   Solution:

$$\begin{aligned}
\lim_{n\to+\infty} \frac{f(n)}{g(n)} &= \lim_{n\to+\infty} \frac{3n^2 + \ln n}{n^2} \\
&= \lim_{n\to+\infty} \frac{6n + \frac{1}{n}}{2n} && \text{by l'Hôpital's rule} \\
&= \lim_{n\to+\infty} \left(\frac{6n}{2n} + \frac{\frac{1}{n}}{2n} \right) \\
&= \lim_{n\to+\infty} \left(3 + \frac{1}{2n^2} \right) \\
&= 3.
\end{aligned}$$

   Since the above limit exists and is a positive constant, it follows that $f(n) \in \Theta(g(n))$.

   Comments: L'Hôpital's rule was applied correctly in the above derivation because

$$\lim_{n\to+\infty} f(n) = \lim_{n\to+\infty} g(n) = +\infty,$$

   $f'(n) = 6n + \frac{1}{n}$, and $g'(n) = 2n$. Alternate solutions (including the use of the definition of "$\Theta(g(n))$" instead of the limit test) would also be completely acceptable.

3. Suppose $n \geq 1$, $S_n = \sum_{i=1}^{n} g(i)$, where the function $g(x) = \frac{x}{\ln(2x)}$ is a nondecreasing function of x — you may use this fact without proving it in this question.

   (a) *(9 marks)* Find a function $f(n)$ in closed form such that $S_n \in \Theta(f(n))$, and prove that this holds **without** using approximation by integrals.

   Solution: Since it is given that $g(x)$ is a nondecreasing function,

$$S_n = \sum_{i=1}^{n} g(i) \leq \sum_{i=1}^{n} g(n) = ng(n) = \tfrac{n^2}{\ln(2n)} \leq \tfrac{n^2}{\ln n}.$$

   It's also true that

$$S_n = \sum_{i=1}^{n} g(1) \geq \sum_{i=1}^{n} g(1) = ng(1) = \tfrac{n}{\ln 2},$$

   but this is not sufficient to answer this question because $\frac{n}{\ln 2} \notin \Theta\left(\frac{n^2}{\ln n}\right)$.

   However, this *does* prove that $S_n \in O\left(\frac{n^2}{\ln n}\right)$.

   To continue, let's split the sum and bound terms for both of the partial sums we get by splitting: Let $S_n = S_L + S_U$ where

$$S_L = \sum_{i=1}^{m} g(i) \qquad \text{and} \qquad S_U = \sum_{i=m+1}^{n} g(i),$$

   where (as usual) $m = \lfloor \frac{n}{2} \rfloor$.

   Once again, $g(x)$ is a nondecreasing function of x, so

$$S_L = \sum_{i=1}^{m} g(i) \geq \sum_{i=1}^{m} g(1) = mg(1) = \frac{\lfloor \frac{n}{2} \rfloor}{\ln 2} \geq 0,$$

   and

$$\begin{aligned}
S_U &= \sum_{i=m+1}^{n} g(i) \\
&\geq \sum_{i=m+1}^{n} g(m+1) && \text{since the function } g \text{ is nondecreasing} \\
&= (n-m)g(m+1) \\
&\geq \left(\tfrac{n}{2}\right) g(m+1) && \text{since } n-m \geq \tfrac{n}{2} \text{ and } g(m+1) \geq 0 \\
&\geq \left(\tfrac{n}{2}\right) g(n/2) && \text{since } g \text{ is nondecreasing, } m+1 \geq \tfrac{n}{2}, \text{ and } \tfrac{n}{2} \geq 0 \\
&= \left(\tfrac{n}{2}\right) \left(\tfrac{n/2}{\ln n}\right) \\
&= \tfrac{n^2}{4 \ln n} \\
&= \tfrac{1}{4} \left(\tfrac{n^2}{\ln n}\right).
\end{aligned}$$

Therefore

$$S_n = S_L + S_U \geq 0 + \tfrac{1}{4}\left(\tfrac{n^2}{\ln n}\right) = \tfrac{1}{4}\left(\tfrac{n^2}{\ln n}\right),$$

which implies that $S_n \in \Omega\left(\frac{n^2}{\ln n}\right)$ (since $\frac{1}{4}$ is a positive constant).

Therefore $S_n \in \Theta\left(\frac{n^2}{\ln n}\right)$, so $S_n \in \Theta(f(n))$ where $f(n) = \frac{n^2}{\ln n}$.

Comments: The approximations used above (for example, using 0 as a lower bound for S_L) were chosen to obtain a lower bound that was "as clearly" in $\Theta(f(n))$ as possible — namely, a constant multiple of $f(n)$. You could have made different approximations — for example, avoiding "approximations" whenever possible — and obtained slightly *different* upper and lower bounds as a result. If this happened and you still gave a correct argument that your bounds were tight (possibly mentioning or using the limit test, to prove this in the end), then this will be perfectly acceptable (if done correctly).

It would also be acceptable to use a completely different technique — probably, guessing the final answer and then using the "substitution method" (involving a pair of proofs by induction, with unknown constants included in the proofs), if you were using a technique that had been introduced in the course.

However, this might be more difficult to do correctly, since it requires that you "guess" the final answer instead of having it derived as part of the answer, and since the required proofs by induction might be longer and messier than the solution that's given above.

(b) *(3 marks)* Now suppose $G(n)$ is a function of n such that $G'(n) = g(n)$. Use approximation by integrals to give a **lower bound** for S_n (that is, a function that is always less than or equal to S_n). Your answer should be an expression that involves $G(n)$.

Solution: Since $g(x)$ is a nondecreasing function,

$$\sum_{i=1}^{n} g(i) \geq \int_0^n g(t)\,dt.$$

It's given that $G'(x) = g(x)$, and it follows by the fundamental theorem of calculus that

$$\int_0^n g(t)\,dt = G(n) - G(0).$$

Therefore

$$G(n) - G(0)$$

is a lower bound for S_n.

4. *(5 marks)* Find a function $f(n)$ in closed form such that $T(n) \in \Theta(f(n))$, where

$$T(n) = \begin{cases} 1 & \text{if } n < 4, \\ 16T\left(\lceil n/4 \rceil\right) + 2n^2 & \text{if } n \geq 4, \end{cases}$$

and give a **brief** proof that your answer is correct.

Solution: The above recurrence has the form that is required if the **master theorem** is to be used, and it corresponds to the choice of constants $a = 16$, $b = 4$, and $c = 1$, and to the function $f(x) = 2x^2$.

Furthermore, $f(x) = x^\alpha(\log_2 x)^\beta$ for constants $\alpha = 2$ and $\beta = 0$, so this function is in the form that's required if the "simplified" master theorem is to be used.

In this case, $\alpha = 2$ and $\log_b a = \log_4 16 = 2$ (since $16 = 4^2$), so $\alpha = \log_b a$. Since $\beta = 0$, the second case given in the theorem is applicable.

Therefore, $T(n) \in \Theta(n^{\log_b a} \log_2 n) = \Theta(n^2 \log_2 n)$.

Comment: It will also be perfectly acceptable if you used the (corrected) *original* master theorem instead of the simplified theorem in order to solve this problem.

The correct use of other techniques, such as the iteration method or the substitution method, would be acceptable too. However, they weren't required and (as the number of marks for the question should have suggested), time wasn't really allowed for this.

5. *(10 marks)* Prove that $T(n) \in O(4^n)$, where

$$T(n) = \begin{cases} 1 & \text{if } n = 0, \\ 8 & \text{if } n = 1, \\ T(n-1) + 12T(n-2) & \text{if } n \geq 2. \end{cases}$$

Hint: You should consider both the cases $n = 0$ and $n = 1$ as special cases.

Solution: The "substitution method" will be used.

Claim. There exists a positive constant $c > 0$ such that $T(n) \leq c4^n$ for every integer $n \geq 1$.

Proof of Claim. This will be proved by induction on n. As suggested by the "hint," it will be convenient to treat the cases $n = 0$ and $n = 1$ as special cases, so both will be considered in the "basis."

Basis: If $n = 0$ then $T(n) = T(0) = 1$ and $c4^n = c4^0 = c$, so $T(n) \leq c4^n$ in this case **provided that** $c \geq 1$.

If $n = 1$, then $T(n) = T(1) = 8$ and $c4^n = c4^1 = 4c$, so $T(n) \leq c4^n$ in this case provided that $4c \geq 8$; that is, provided that $c \geq 2$.

Inductive Step: Suppose now that $n \geq 0$ and that $T(n) \leq c4^n$ and that $T(n+1) \leq c4^{n+1}$. Then

$$\begin{aligned} T(n+2) &= T((n+2)-1) + 12T((n+2)-2) && \text{since } n+2 \geq 2 \\ &= T(n+1) + 12T(n) \\ &\leq c4^{n+1} + 12c4^n && \text{by the inductive hypothesis} \\ &= (4c + 12c)4^n && \text{since } 4^{n+1} = 4 \cdot 4^n \\ &= (16c)4^n \\ &= c \cdot 4^2 \cdot 4^n = c4^{n+2}, \end{aligned}$$

as desired (**without** requiring any additional conditions on c).

The above conditions on c can be satisfied as long as c is chosen to be any positive constant that is greater than or equal to 2. In particular, if $c = 2$ then all these conditions are satisfied. It therefore follows by induction on n that the claim is correct (and, furthermore, that we can choose the constant c mentioned in the claim to be 2). □

It clearly follows from the claim (and the definition of $O(4^n)$) that $T(n) \in O(4^n)$, as desired.

Comment: As for the other questions, other correct solutions that used different techniques will be perfectly acceptable — but the substitution method was probably the one that was easiest to use correctly in this case. The fact that you were *given* an asymptotic bound in closed form should have been an indication that the substitution method was a good method to try first.

The iteration method could be used, "in principle," but you'd very likely need to discover an exact solution in closed form, in order to make this work. Since $T(n) = \frac{11}{7} \cdot 4^n - \frac{4}{7} \cdot (-3)^n$ and there's nothing in the above question that might suggest that "$(-3)^n$" should be involved at all, this would probably be quite difficult to do. (And, as usual, part marks will be awarded for reasonable attempts.)

Bibliography

[1] Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.

[2] Gilles Brassard and Paul Bratley. *Fundamentals of Algorithmics*. Prentice Hall, 1996.

[3] Thomas H. Cormen, Charles E. Leiserson, and Ronald L Rivest. *Introduction to Algorithms*. McGraw-Hill/MIT Press, 1990.

[4] Ronald L. Graham, Donald E. Knuth, and Oren Patashnik. *Concrete Mathematics: A Foundation for Computer Science*. Addison-Wesley, second edition, 1994.

[5] Ralph P. Grimaldi. *Discrete and Combinatorial Mathematics: An Applied Introduction*. Addison-Wesley, second edition, 1989.

[6] Ellis Horowitz, Sartaj Sahni, and Sanguthevar Rajasekaran. *Computer Algorithms/C++*. Computer Science Press, 1996.

[7] Donald E. Knuth. *The Art of Computer Programming, Volume 1: Fundamental Algorithms*. Addison-Wesley, third edition, 1997.

[8] Donald E. Knuth. *The Art of Computer Programming, Volume 2: Seminumerical Algorithms*. Addison-Wesley, third edition, 1997.

[9] Donald E. Knuth. *The Art of Computer Programming, Volume 3: Sorting and Searching*. Addison-Wesley, second edition, 1998.

[10] Richard Neapolitan and Kumarss Naimipour. *Foundations of Algorithms Using C++ Pseudocode*. Jones and Bartlett, second edition, 1997.

[11] G. Polya. *How to Solve It*. Princeton University Press, second edition, 1957.

[12] Gregory J. E. Rawlins. *Compared to What? An Introduction to the Analysis of Algorithms*. Computer Science Press, 1992.

[13] Robert Sedgewick. *Algorithms in C, Parts 1-4: Fundamentals, Data Structures, Sorting, Searching*. Addison-Wesley, 1998.

[14] Robert Sedgewick and Philippe Flajolet. *An Introduction to the Analysis of Algorithms*. Addison-Wesley, 1996.

[15] Daniel Solow. *How to Read and Do Proofs*. Wiley, second edition, 1990.

Index