Learning Guide Unit 2

Site: University of the People
Course: CS 3304-01 Analysis of Algorithms - AY2024-T5
Book: Learning Guide Unit 2

Printed by: Mejbaul Mubin
Date: Saturday, 29 June 2024, 6:42 AM

Description

Learning Guide Unit 2

Table of contents

Overview

Unit 2: Divide and Conquer Algorithms

Topics:

- Techniques for the Analysis of Algorithms
  - Lower bounds analysis
  - Summation Techniques
  - Recurrence relations
  - Estimating upper and lower bounds
- Divide and Conquer Algorithms
  - Matrix Multiplications
  - Binary Search
  - MergeSort
  - Fast Fourier Transform

Learning Objectives:

By the end of this Unit, students will be able to:

1. Recognize a good algorithm based upon its upper and lower bounds.
2. Implement a process for improving (finding a tighter) lower bound.
3. Understand techniques of lower bounds analysis for different forms of algorithms including:
   - Searching
   - Finding maximum value
   - State space algorithms.
   - Finding the ith best element
   - Optimal sorting
4. Explain and be able to apply divide and conquer algorithms.
5. Recognize and understand the operation of common divide and conquer algorithms including:
   - Matrix Multiplications
   - Binary Search
   - MergeSort
   - Fast Fourier Transform

Tasks:

- Read the Learning Guide and Reading Assignments
- Participate in the Discussion Assignment (post, comment, and rate in the Discussion Forum)
- Make entries to the Learning Journal
- Take the Self-Quiz

Introduction

More on Recurrence Relations
During this unit we will be looking at divide and conquer algorithms. A typical implementation of divide and conquer algorithms employs a recursive structure which is modeled using recurrence relations. The ability to model the complexity of an algorithm using recurrence relations is an important technique used both to assess the complexity and running time of an algorithm as well as to develop new efficient algorithms.

Lower Bounds Analysis
We will take a closer look at lower bounds. To define a lower bound we can contrast it with an upper bound. We know that an upper bound is the worst case scenario of resource consumption that is consumed with the best algorithm that we have for a particular problem. We can illustrate this concept by thinking about an algorithm that searches a simple list. Imagine if you will that we have a list containing 100 integers. Further imagine that we want to find the first instance of the number '3' if it exists within the list. If the list is sorted, then the algorithm to find this number would be relatively efficient because we would start at the beginning of the list and continue to check each entry until we found the number '3'. Given the fact that the number 3 is rather small we would likely find the number quickly.

On the other hand imagine that the list is not sorted, then the number 3 could just as easily appear at the very end of the list as it does at the beginning of the list. If the number appears at the end of the list, then it would take In this case the upper bound would be determined by the number being at the end of the list and a lower bound would be defined as the number occurring at the beginning of the list. As you can see there is a great deal of distance between the two bounds. The algorithm that does this search is not very efficient because there is so much variability in its execution. We could just as easily see the upper bound execution as the lower bound execution.

In Chapter 15 of our text, we learn that the 'tighter' the bounds or the closer that the upper bound is to the lower bound, the more efficient the algorithm. In fact we learn that when the upper bound and lower bound are the same then the algorithm cannot become any more efficient. The text defines an algorithm as a 'good' algorithm when the upper bound and lower bound are the same asymptotically.
Throughout this unit we will learn about how to determine lower bounds for algorithms designed to solve a range of problems.

In this unit we will explore a variety of approaches to determine the average cost of an algorithm. This becomes important when determining the cost or complexity of an algorithm that is iterative in nature. One technique that we want to explore in detail is amortized analysis.

Amortized analysis refers to determining the time-averaged running time for a sequence of operations. It is different from what is commonly referred to as average case analysis, because amortized analysis does not make any assumption about the distribution of the data values, whereas average case analysis assumes the data are not "bad" (e.g., some sorting algorithms do well on "average" over all input orderings but very badly on certain input orderings). That is, amortized analysis is a worst case analysis, but for a sequence of operations, rather than for individual operations. It uses the fact that we are analyzing a sequence to "spread out" the costs (think of insurance where everyone can pay a relatively modest amount despite some catastrophic costs).

The motivation for amortized analysis is to better understand the running time of certain techniques, where standard worst case analysis provides an overly pessimistic bound. Amortized analysis generally applies to a method that consists of a sequence of operations, where the vast majority of the operations are cheap, but some of the operations are expensive. If we can show that the expensive operations are particularly rare we can "charge them" to the cheap operations, and only bound the cheap operations.

The general approach is to assign an artificial cost to each operation in the sequence, such that the total of the artificial costs for the sequence of operations bounds total of the real costs for the sequence. This artificial cost is called the amortized cost of an operation. In order to analyze the running time, the amortized cost thus is a correct way of understanding the overall running time - but note that particular operations can still take longer so it is not a way of bounding the running time of any individual operation in the sequence.

There are several approaches commonly used for amortized analysis however we will look at two in particular including the aggregate method and the banker's method (tokens).

Binary Counter Example
Consider the problem of storing a very large binary counter. Say we decide to use an array, where each entry A[i] stores the i-th bit.

We will analyze the running time of the operation of counting using this representation, so the sequence of operations is a sequence of increments of the counter.

We will use the standard way of incrementing the counter, which is to toggle the lowest order bit. If that bit switches to a 0 we toggle the next higher order bit, and so forth until the bit that we toggle switches to a 1 at which point we stop.
A[m] A[m-1] ... A[3] A[2] A[1] A[0] cost
-- ----
0 0 0 0 0 0
1
0 0 0 0 0 1
2
0 0 0 0 1 0
1
0 0 0 0 1 1
3
0 0 0 1 0 0
1
0 0 0 1 0 1
2
0 0 0 1 1 0

When the result of the increment operation is n, the number of bits that change is at most 1+floor(lg n), that is the number of bits in the binary representation of n.

Thus in a traditional worst case analysis, the cost of counting up to n, which a sequence of n increments, is O(n log n).

But do you really think this is taking that much time? Does each increment in the sequence really cost O(log n)? How do we show its less? This is the goal of amortized analysis.

Aggregate Method.
Let's consider how often we flip each individual bit, and sum those up to bound the total, rather than individually obtaining a worst case bound for each bit. How often is A[0] toggled in counting up to n? How often is A[1] toggled, A[2] toggled, etc? Every time, every other time, every fourth time, ...
n + floor(n/2) + floor(n/4) + ...
<= n + n/2 + n/4 + ...
<= 2n

So the amortized cost of an increment is 2, and the time to count to n (n increments) is 2n=O(n).

Divide-and-Conquer Algorithms
Divide-and-conquer algorithms solve problems by breaking the larger problem to be solved into subsequently smaller problems until these smaller problems can be easily solved. The results are then combined together to solve the original problem. Divide and conquer algorithms use the following three phases:

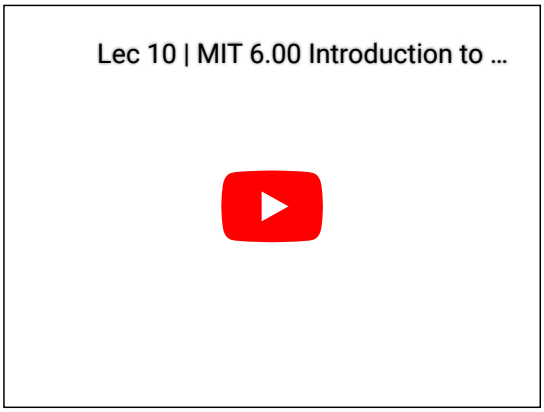
1. Divide the problem into smaller sub-problems. A sub-problem of a problem is a smaller input for the same problem. For example, for the problem of computing a closest pair of points, in a subproblem there will be fewer points but the task is still to find a closest pair of points. Generally, the input for the subproblem is a subset of the original input, but that need not be the case.
2. Conquer by recursively solving the subproblems (unless the problem size has reached a specified termination condition).
3. Combine the solutions to the subproblems to obtain the solution for the original problem.

To design a divide-and-conquer algorithm, you need to think about what subproblem solution would be helpful in computing the answer to the original problem. While many students initially feel like the “magic” is in the recursion, that step is one that can always be done. The real magic is in the combining, and finding the right subproblems to recursively solve. We now describe two sample divide-and-conquer algorithms.

For students who have access to computers with sufficient network speed, I would also recommend the following video lectures on Divide and Conquer algorithms. These lectures are entirely optional and provided as a supplemental resource for those students who wish to take advantage of them.

Santa's Dirty Socks (Divide and Conquer Algorithms)

Lecture 10: Divide and conquer methods, merge sort, exceptions by Prof. Eric Grimson, and Prof. John Guttag, MIT



Reading Assignment

Topic 1: Amortized Analysis
-Read Chapter 2, section 2.2. Divide-and-conquer Algorithms in Algorithms by S. Dasgupta, C.H. Papadimitriou, and U.V. Vazirani available at: http://www.cs.berkeley.edu/~vazirani/algorithms/chap2.pdf

-Amortized Analysis, By Charles E. Leiserson – MIT (at 19:05 minutes) http://videolectures.net/mit6046jf05_leiserson_lec13/#

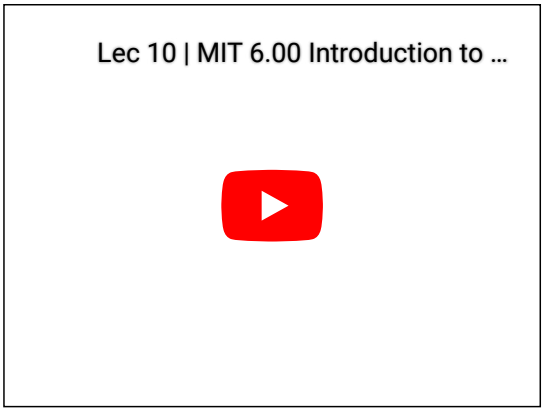
Topic 2: Lower Bounds Analysis
-Chapter 15: Lower Bounds, in A Practical Introduction to Data Structures and Algorithm Analysis by Clifford A. Shaffer.

Topic 3: Divide and Conquer Algorithms
-Read Chapter 2 Divide-and-conquer Algorithms in Algorithms by S. Dasgupta, C.H. Papadimitriou, and U.V. Vazirani available at http://www.cs.berkeley.edu/~vazirani/algorithms/chap2.pdf
Supplemental Materials

The following are video lectures that are available via YouTube and other sources that are related to the topics in the unit and can be used as a supplemental resource for students looking for more details or to be introduced to the same topic from another source. The use of these resources is not required and is entirely optional.

-Santa's Dirty Socks (Divide and Conquer Algorithms) -

-Lecture 10: Divide and conquer methods, merge sort, exceptions by Prof. Eric Grimson, and Prof. John Guttag, MIT



Unit 2 Optional Video Lectures

The following video lectures are optional resources that have been made available to students who can take advantage of them. These lectures are strictly optional resources. All of the information in these lectures is available in other learning resources within the course. These lectures are provided for those students who have sufficient network bandwidth and technology capabilities to take advantage of video content. These lectures cannot be used instead of the required assigned resources and there is no information that is not contained in the assigned resources. These lectures simply present some of the information in a different format.

- Unit 2 Lecture 1: Techniques for the Analysis of Algorithms
- Unit 2 Lecture 2: Estimating Upper and Lower Bounds
- Unit 2 Lecture 3: Divide and Conquer Algorithms

Discussion Assignment

Suppose you are choosing between the following three algorithms:

- Algorithm A solves problems by dividing them into five sub-problems of half the size, recursively solving each sub-problem, and then combining the solutions in linear time.
- Algorithm B solves problems of size n by recursively solving two sub-problems of size n-1 and then combining the solutions in constant time.
- Algorithm C solves problems of size n by dividing them into nine sub-problems of size n/3, recursively solving each sub-problem, and then combining the solutions in O(n2) time.

What are the running times of each of these algorithms (in big-O notation), and which would you choose?

For your discussion post, explain in your own words why you chose a particular algorithm. Include one or two examples to explain your thought process to show what is occurring and how the methodology works. Demonstrate your understanding of the intricacies of the algorithm you chose. Use APA citations and references for any sources used.

Post your response to these problems and then respond to at least 3 of your peers. Discuss whether each peer's solution is correct and why?

Learning Journal

Assignment Instructions

This assignment will assess your skills and knowledge on:

- Understanding techniques of lower bounds analysis for different forms of algorithms.
- Explaining and be able to apply divide and conquer algorithms.
- Recognizing and understand the operation of common divide and conquer algorithms.

Assignment

There are two parts A and B to this assignment, and you must submit both parts with steps and proper explanations. Answer the following questions based on the knowledge gained by you on the different applications of divide and conquer algorithm.

Part A

Sort the following list using the optimal divide-and-conquer algorithm: 36, 25, 44, 2, 8, 88, 11.

1. Which application of the divide and conquer algorithm you will use for this purpose? Explain it.
2. List and explain all the intermediary steps of the algorithm to sort the above list.
3. Explain the space and time complexities of the algorithm used by you for sorting the list.

Please note: The optimality of an algorithm is often measured with respect to certain factors, such as time complexity, space complexity, or a combination of both. An optimal algorithm should have the best possible time complexity for a given problem. Optimal algorithms often use minimal memory or have low space complexity, especially in situations where memory resources are limited.

Include the following items in your explanation:

- Propose the most appropriate algorithm.
- Submit a step-by-step solution and explain in your own words.
- Share the sorted list at the end.
- Discuss the space and time complexities of the algorithm.

Part B

You are an employee of a warehouse and have been provided with 7 identical cartons and a measuring instrument. 6 of the 7 cartons are equal in weight and 1 of the 7 given cartons has less material and thus weighs less. Your task is to find the less weighing carton in exactly two measurements.

Submit a step-by-step solution for the above problem applying the technique of divide-and-conquer.

Hints:

- The input is not a sorted.
- Consider that the cartons are labelled as C1, C2, C3, C4, C5, C6, and C7.
- You may start by dividing the 7 Cartons into three groups.

Submission Instructions

- Submit the solutions to both part A and part B in one document.
- Make sure your submission is double-spaced, using Times New Roman, 12-point font, with 1” margins.

Your submission should be clearly written, concise, and well organized, and free of spelling and grammar errors. You must review the grading rubric listed on the submission page to find out how the instructor will grade your submission.

Self-Quiz

The Self-Quiz gives you an opportunity to self-assess your knowledge of what you have learned so far.

The results of the Self-Quiz do not count towards your final grade, but the quiz is an important part of the University's learning process and it is expected that you will take it to ensure understanding of the materials presented. Reviewing and analyzing your results will help you perform better on future Graded Quizzes and the Final Exam.

Please access the Self-Quiz on the main course homepage; it will be listed inside the Unit.

Checklist

- [ ] Read the Learning Guide and Reading Assignments
- [ ] Participate in the Discussion Assignment (post, comment, and rate in the Discussion Forum)
- [ ] Make entries to the Learning Journal
- [ ] Take the Self-Quiz