SIT320 - Advanced Algorithms

Credit Task 10: Algorithm Analysis

School of Information Technology, Deakin University

Overview of Learning (Module Summary)

GitHub Link
ChatGPT Link

- Algorithm complexity is an important concept in computer science, specifically when trying to design efficient algorithms. It is used to describe the performance of an algorithm in terms of the size of the input.
- We can use algorithm complexity to compare the performance of different algorithms, and understand whether an algorithm is efficient enough for a given problem.
- When looking at asymptotic complexity, we should keep in mind to supress constant factors and lower order terms, as these are not significant when the input size is large.
- Once we have a grasp of asymptotic notation it give us a common language to discuss the performance of algorithms.

Problem 1: Solve Recurrence using Substitution Method

Problem Statement

- $T(n) = \begin{cases} nT(n-1), & \text{if } n > 1 \\ 1, & \text{if } n = 1 \end{cases}$
- $T(n) = \begin{cases} T(n-1) + \log n, & \text{if } n > 1 \\ 1, & \text{if } n = 1 \end{cases}$
- $T(n) = \begin{cases} 3T(n/2) + n, & \text{if } n > 1 \\ 1, & \text{if } n = 1 \end{cases}$

Solution

Task 1 Link

Key Take aways

- Recurrences express a running time bound T(n) in terms of a number of operations performed on recursive calls of smaller inputs. Recurrences must also include a base case, which is a non-recursive expression of the running time for a small input.
- Solving recurences is usually not the hard part, but rather coming up with the recurrence in the first place.
- When given a recurence we can often spot a pattern and be fairly confident what the solution will be. However, we need to come up with a proof to show that our solution is correct.

Problem 2: Solve Recurrence using Master Theorem

Problem Statement

- $T(n) = \begin{cases} T(\sqrt{n}) + \log n, & \text{if } n > 1 \\ 1, & \text{if } n = 1 \end{cases}$

Solution

Task 2 Link

Key Take aways

- The master theorem is a powerful tool for solving recurrences that describe the running time of divide-and-conquer algorithms that have the same subproblem size in each recursive call.
- The biggest strength of the master theorem is that it provides a quick solution to a large class of recurrences in the form of T(n) = aT(n/b) + O(n^d).
- a is the number of subproblems in the recursion, b is the factor by which the input size is reduced, and d is the exponent in the running time of the "combine" step.
- The quantities a and b^d represent two opposing forces; (a) being the rate at which the recursion branches out, and (b^d) being the rate at which the work is reduced at each level of recursion.
- To solve our particular problem we had to use variable substitution. The reason for this was because our recurrence was not in the above mentioned form.

Problem 3: Solve Recurrence using Recursion Tree Method

Problem Statement

- $T(n) = 2T(n/2) + cn$

Solution

Task 3 Link

Key Take aways

- The recursion tree method is ueful for generating a sloppy, but good guess that we can then prove using the substitution method.
- With this method we are looking to keep track of the total cost of all the work done at each level of recursion.
- Nodes of the tree represent the recursive calls, and the children of each node represent the subproblems generated by the recursive call.
- We look at three major components of the recursion tree:
    - The number of subproblems in each level of recursion.
    - The size of each subproblem.

    - The cost of the work done per level.
- Work completed at level j is less than or equal to - (cn^d * [n/(b^j]^d). These variables are taking from the recurrence T(n) = aT(n/b) + O(n^d). c is a constant that represents the cost of the work done per level.

Problem 4: Complexity Analysis of Quick Sort Algorithm

Problem Statement

Analyze the time complexity of the Quicksort algorithm using the Akra-Bazzi method and substitution with induction. We are to assume that the subproblem size is unbalanced.

Solution

Task 4 Link

Link to Analysis

Key Take aways

- The main feature of Quicksort is that it has an average-case running time of O(nlogn). We choose to analyse the average-case running time because through randomisation we find that the this is the most likely case.
- Finding out that selecting a pivot could be done in constant time was a key insight. This allowed us to simplify the recurrence.
- I implented three solutions to this problem. The first two were using substitution with induction on both balanced and unbalanced subproblems. The third solution was using the Akra-Bazzi method.
- The Akra-Baazi was powerful and simple at the same time. Fortunately the integral was easy to solve, and the solution was the same as the one I got using substitution with induction.

Readings and Research

Algorithms Illuminated (Part 1): The Basics by Tim Roughgarden

- Chapter 2: Asymptotic Analysis

- Chapter 4: The Master Method
- Chapter 5: QuickSort
- Chapter 6: Liner-Time Selection
- A: Quick Review of Proofs By Induction

Introduction To Algorithms by CLRS

- Chapter 4: Divide and Conquer
- Chapter 7: Quicksort