Lab 4: Divide and Conquer Algorithms

Learning Outcomes

- Design an algorithm for a given computational problem statement
- Justify the correctness of an algorithm
- Perform asymptotic complexity analysis of the run time of an algorithm
- Generate test cases for an algorithm
- Correctly implement an algorithm from pseudocode
- Design and execute benchmarks for an algorithm

Instructions

The goal of this labs is to give you an opportunity to practice and apply what you are learning in lecture to "new" problems. In this lab, you will apply the divide and conquer strategy to create an algorithm that solves the selection problem. The selection problem can be stated as follows:

Selection Problem

> **Input:** A sequence of $n \geq 2$ numbers and an integer $1 \leq k \leq n$.
>
> **Output:** Find k^{th} smallest number.

For example, if we are given the following numbers:

[5, 1, 6, 7, 3, 4, 8]

and asked to find k = 3, we would return 4.

It is easier to see why if we sort the numbers:

[1, 3, 4, 5, 6, 7, 8]

The number 4 is the 3^{rd} smallest number. This problem commonly occurs in statistics when trying to find the medium or a percentile of a distribution.

The simplest way to solve this problem is to sort the numbers and take the element at index k – 1. From class, you know that the fastest comparative sorting algorithms require O(n log n) time. If the data are partially sorted (e.g., the first k elements are sorted), the run time can be further reduced to O(k n) or O(k log n).

It turns out that it is possible to create an even faster algorithm using a divide and conquer strategy. Hint: You randomly select a pivot value and partition the numbers around that pivot. The pivot value turns out to be the j^{th} number, so the left partition contains j elements (including the pivot) and the right partition contains n – j elements.

Submit a report containing the following:

1. A paragraph describing the approach that you used to solve the problem. Provide at least 2 illustrations that explain the approach.
2. High-level pseudocode for an algorithm that uses that rule to solve the computational problem for any input.
3. Provide an explanation and justification for why your algorithm is correct (1-3 paragraphs).
4. A table of your test cases, the answers you expect, and the answers returned by running your implementation of the algorithm.
5. Derive a recurrence relation describing the run time in terms of the number of points n. (Hint: assume that the random pivot divides the elements in half each time.)
6. Solve the recurrence relation to get a run time in terms of n in asymptotic notation.
7. Benchmark your implementation versus an approach that sorts the numbers and picks the element at index k – 1. You should include a table and graph from benchmarking different lists with different sizes of numbers. The benchmarks should support your theoretically-derived run time and provide evidence that the run time of your algorithm grows more slowly than the sorting approach.
8. Attach all of your source code and test cases in an appendix.

Submission Instructions

Submit the lab report as a PDF and all source code to Canvas.

Rubric

		Full Credit	Partial Credit	No Credit
Lab report writing and presentation quality	20%			
Solution Approach Description	10%	Approach is well described with all necessary details to implement and explain the approach to someone else.	Approach is decently described with most details necessary to implement. Explanation may be unclear in places (not constructive).	The description is insufficient to implement or explain to others.
High-level Pseudocode	10%	Pseudocode describes an algorithm which is correct for all allowed inputs.	Pseudocode describes an algorithm which is correct for most allowed inputs.	Pseudocode is not correct for most allowed inputs.
Justification of Correctness	10%	Uses techniques described in class to provide a solid and convincing argument that the algorithm is correct.	Provides a somewhat convincing argument that the algorithm is correct.	Argument contains one or more serious flaws.
Asymptotic run time analysis	10%	Analysis is correct for the provided pseudocode	Analysis is mostly correct except for minor flaws	Analysis is significantly flawed
Algorithm Implementation	10%	Implementation is faithful to the pseudocode description above and correct.	Implementation is mostly faithful to the pseudocode description above and correct for most inputs.	Implementation is not faithful to the pseudocode or not correct for some common inputs.
Test Cases	10%	Test cases consider a range of problem sizes and complexities and potential edge cases.	Limited number of test cases or only testing obvious or simple cases	No test cases
Benchmarks	10%	Benchmark experiments were set up and implemented correctly.	Benchmark experiments, implementations, or results are mostly correct.	Benchmark experiments, implementations, or results are flawed.

Algorithm run time	10%	Algorithm is faster than O(k log n), as determined by both empirical and theoretical analysis results.	Algorithm is faster than O(n log n).	Algorithm equal to or slower than O(n log n) brute-force search