



Computer Science 385

Design and Analysis of Algorithms

Siena College
Spring 2024

Problem Set 1

Due: 4:00 PM, Friday, February 2, 2024

You may work alone or in a group of size 2 or 3 on this assignment. However, in order to make sure you learn the material and are well-prepared for the exams, you should work through the problems on your own before discussing them with your partner(s), should you choose to work with others. In particular, the “you do these and I’ll do these” approach is sure to leave you unprepared for the exams.

All GitHub repositories must be created with all group members having write access and all group member names specified in the README.md file by 4:00 PM, Monday, January 29, 2024. This applies to those who choose to work alone as well!

There is a significant amount of work to be done here, and you are sure to have questions. It will be difficult if not impossible to complete the assignment if you wait until the last minute. A slow and steady approach will be much more effective.

Getting Set Up

In Canvas, you will find a link to follow to set up your GitHub repository, which will be named `ps1-yourgitname`, for this lab. Only one member of the group should follow the link to set up the repository on GitHub, then others should request a link to be granted write access.

Submitting

Please submit a hard copy (typeset preferred, handwritten OK but must be legible) for all written questions. Only one submission per group is needed.

Your submission requires that all required code deliverables are committed and pushed to the main branch of your repository’s origin on GitHub. If you see everything you intend to submit when you visit your repository’s page on GitHub, you’re set.

Written Questions: Discrete Math and Data Structures

Question 1: Indicate and briefly justify (in words, not mathematically) the “Big O” complexity for each of the operations below. Differentiate among best, average, and worst cases and specify under what circumstances they occur, where relevant. Specify the basic operation you are counting in each case. You may use any trustworthy resource, but any such resource must be cited. Be sure you understand any answers you need to look up, as you will see some of these or very similar questions as quiz or exam questions soon. (16 points)

For example, if the operation is “Find the largest value in an unsorted array of n integers”, your response could be:

The basic operation is an integer comparison. Since the array is unsorted, there is no way to find the largest until you have compared with every one of the n values, so the best, average, and worst cases are all $O(n)$.

- Perform a binary search in a sorted array of n integers.
- Add an element to an `ArrayList` that contains n values, using the default `add` method.
- Add an element to a singly-linked list that contains n values using the most efficient possible `add` method.
- Add an element to a sorted `ArrayList` that contains n integers, and which has capacity available to store the new value.
- Sort an array of n integers using bubble sort.
- Determine if a key exists in a binary search tree that contains n keys.
- Determine if a specific value is currently stored in a sorted array of n integers.
- And one that you might not have seen, but should be able to reason out: count the number of times a specific value occurs in a sorted array of n integers.

Question 2: For each given expression below, give one equivalent, but different looking, expression for it that you can find in the expression bank. In other words, don't say that n is equivalent to n . For logarithms, \lg indicates base 2, and otherwise the base of the log will be indicated with a subscript. (14 points)

Given expressions:

$n, n^2, n^3, \lg 2, \lg 16, \lg n, \lg n + \lg b, \lg(a/b), \lg(2^n), \lg(2^{n+1}), 2^{n+1}, 4(2^n), 2^n 2^n, (2^n)^2$

Expression bank:

$n, n + 1, n^{\lg 8}, n^{\lg 16}, \log_{10} n, \lg n, n \lg n, n \lg 2, 2^{\lg n}, 4^{\lg n}, 8^{\lg n}, \lg(ab), \lg a - \lg b, 2^{n+2}, 1, 2, 3, 4, \frac{\log_{10} n}{\log_{10} 2}, \lg(2^n), 2(2^n), 2^{n+n}, 2^{2n}, 2^{n+1}, 2^{n+2}, 4^n, (2^n)^n$

Written Questions: Summations and Counting

Question 3: Compute closed forms the following sums. Show your work. No credit if intermediate steps are not shown. Remember page 476 of your textbook! (16 points)

$$\sum_{i=1}^8 (4n + 2i + 7)$$

$$\sum_{i=0}^{n-1} (4n + 2i + 7)$$

$$\sum_{i=0}^n 2^i \cdot n$$

$$\sum_{i=0}^n 2^{i+3}$$

For the next 5 questions, refer to this Java code fragment:

```
for ( i = 1; i <= n; i++ ) {
    for ( j = 1; j <= n; j++ ) {
        for ( k = 1; k <= j; k++ ) {
            System.out.println("Hello!");
            System.out.println("Hello!");
        }
    }
}
```

Question 4: How many times will this print `Hello!` when $n=2$? (1 point)

Question 5: How many times will this print `Hello!` when $n=3$? (1 point)

Question 6: Write an expression involving three summations that counts the number of times it prints `Hello!` in terms of n . There should be one summation for each loop. (3 points)

Question 7: Simplify your expression from the previous question to get a closed form. Show all work. (3 points)

Question 8: Substitute $n=2$ and $n=3$ in your expression (show your work for this, even if it seems trivial) to see if the answers match what you counted above. (1 point)

Programming Task: Generating Example Arrays

In future problem sets, you will be asked to perform empirical analyses on the sorting algorithms we will be studying. To do this, you will need to generate input arrays for the sorting algorithms. In order to test the best, worst, and average cases of some of these algorithms, you will need to generate input arrays with various characteristics. For simplicity, we will work with arrays of `int`.

- an array filled with n random values within a given range
- an array filled with n values sorted in ascending order
- an array filled with n values sorted in descending order
- an array filled with n values “nearly” sorted in ascending order

Requirements:

- You may use any programming language, but be aware that you will need to implement the empirical analysis studies later using this code, so you will either need to do those studies in the same language or rewrite these generators later in any new language you choose.
- If you use Java, implement it within a class `IntArrayPopulator` that includes `static` methods to fill a given array of `int` with numbers matching each of the above characteristics, and a `main` method that thoroughly tests these methods for various values of n and ranges. Be sure you can achieve similar functionality if you choose a different language.
- By taking the array to be filled as a parameter, you'll be able to reuse the arrays in your studies rather than re-allocating them each time.
- n will be determined by the length of the array passed to your methods.
- It is important that you generate these efficiently – for example, you should not generate sorted input by generating random input then sorting it. Generate it in sorted order right from the start.
- Your method to fill arrays with nearly sorted values should take a parameter that specifies the fraction of entries that are out of order. For example, if the parameter has a value of 0.05, approximately one out of each 20 array slots should contain a value that's out of order. There are many reasonable ways to accomplish this.
- AI assistance guideline: you may not simply enter this problem description into a generative AI system like ChatGPT to have it generate a program for you. However, you may use AI assistive tools like Copilot to aid in development. Such use must be properly acknowledged, and if used, a brief statement of how it was used must be included in the header comment at the top of your program.

Grading for the `IntArrayPopulator` will total 28 points: 4 points for the correctness of each of the 4 generator methods (including efficiency), 6 points for sufficient tests, and 6 points for design, documentation, and style.

Written Questions on Graph Representations

For the questions below, consider the graph representations discussed in class for storing directed graphs.

Suppose that the amount of memory required by the adjacency matrix graph representation is exactly $\frac{|V|^2}{8}$ bytes¹, and that the exact amount of memory required by the adjacency list graph representation is exactly $32|V| + 32|E|$ bytes².

¹This assumes each Boolean value in the array is stored as a single bit.

²This assumes 8 bytes for each reference in the `vertices[]` array, 24 bytes for each `Vertex` object (8 bytes for the head reference plus 16 bytes needed by Java to store housekeeping information about each `Vertex` object), and 32 bytes per `Edge` object (8 bytes for the integer `dest`, 8 bytes for the `next` reference, and 16 bytes to store Java housekeeping information about each `Edge` object).

Question 9: If you have a graph with 2^{21} (just over 2,000,000) vertices and each vertex has 4 outgoing edges, exactly how much memory in gigabytes is needed to store the graph using each representation? For each representation, can it fit into the 32GB of main memory which you might find in a PC today? (4 points)

Question 10: Graphs that have only a few number of edges per vertex are known as sparse graphs. A graph with a high number of edges per vertex is said to be dense. Suppose you have a complete directed graph of 2^{21} vertices. Here every vertex has a directed edge to all the other vertices. This is the “densest” graph there is! Exactly how much memory is needed to store the graph using each representation? Express your answer in gigabytes. (5 points)

Question 11: For a graph with 2^{21} vertices, where is the “break even” point, measured by $|E|$, below which the adjacency list representation is more memory efficient, and above which the adjacency matrix representation is more efficient? (5 points)

Question 12: What is the average vertex out-degree corresponding to your answer from the previous question? (3 points)

Grading

This assignment will be graded out of 100 points.

Feature	Value	Score
Q1	16	
Q2	14	
Q3	16	
Q4	1	
Q5	1	
Q6	3	
Q7	3	
Q8	1	
IntArrayPopulator correctness/efficiency	16	
IntArrayPopulator tests	6	
IntArrayPopulator design/documentation/style	6	
Q9	4	
Q10	5	
Q11	5	
Q12	3	
Total	100	