SIENAcollege
Computer Science

Computer Science 385
Design and Analysis of Algorithms
Siena College
Spring 2019

Problem Set 5
Due: 4:00 PM, Thursday, March 28, 2019

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, should you choose to work with someone. 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, Wednesday, March 20, 2019. 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

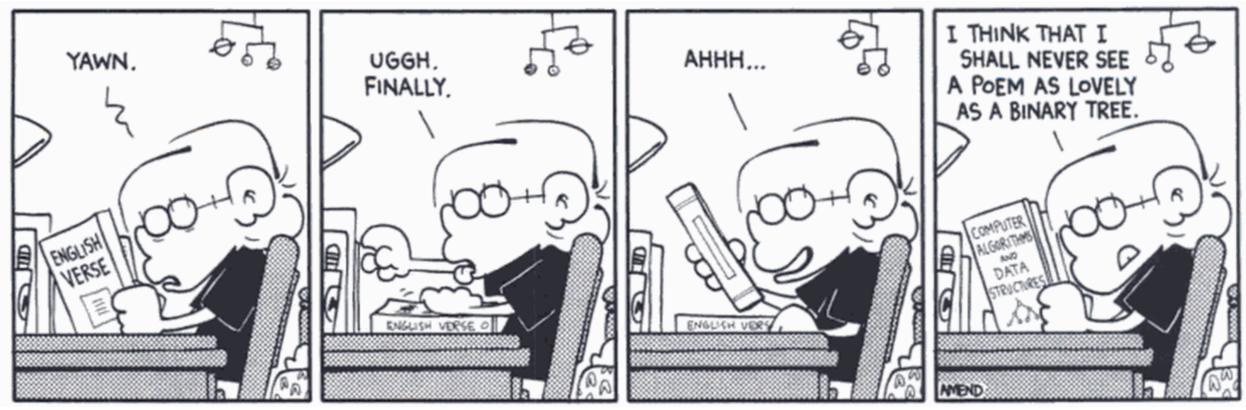
You will receive an email with the link to follow to set up your GitHub repository, which will be named `ps5-yourgitname`, for this problem set. Only one member of the group should follow the link to set up the repository on GitHub, then others will be granted write access.

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 master for 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.

Working with Binary Tree Methods



Your first tasks are mostly a reminder about binary search trees as you saw them in data structures. The `BinarySearchTree` class you will find in your GitHub starter repository has a few methods implemented only as stubs that you'll need to fill in. You'll also answer a few questions along the way.

For testing, the `main` method of that class creates two binary search trees and calls the methods you will be completing to test their functionality. The smaller tree has just 5 nodes, and you can find an ASCII-art representation of it in a comment in the constructor. The larger tree has 1000 nodes with values chosen randomly, but we will all get the same tree every time because the `Random` object is seeded with a specific value (10).

Before you begin, familiarize yourself with the data structure and the provided methods.

? Question 1:
The `numDistK(int k)` method is intended to return the number of nodes in the tree that are a distance of exactly `k` from the root. What values would you expect to get for the small BST for all `k` values below? (1 point)

```
small.numDistK(0) =

small.numDistK(1) =

small.numDistK(2) =

small.numDistK(3) =

small.numDistK(4) =
```

Question 2:
Complete the implementation of `recursiveNumDistK(int k, Node v)`, which is a recursive helper method used by `numDistK(int k)`. (3 points)

Question 3:
For a BST with n nodes and a given value of k, describe the worst case running time and its efficiency class, and briefly describe how you arrived at your answer. (1 point)

Question 4:
The next method you will be working with is `numWithinRange(int s, int e)`. This method returns the number of values in the BST which are in the range from `s` to `e`, inclusive. For the small BST, what answers would you expect from each call below? (1 point)

```
small.numWithinRange(0, 10) =

small.numWithinRange(1, 4) =

small.numWithinRange(5, 7) =

small.numWithinRange(8, 10) =
```

Question 5:
You will be completing the recursive helper method `recursiveNumWithinRange(int s, int e, Node v)`. Obviously one place you would look for a value in the desired range is in `v`'s data field. For a given value of `s`, `e`, and `v`'s data field, under what circumstances would you need to search in each subtree? (2 points)

Question 6:
Complete the implementation of `recursiveNumWithinRange`. For full credit, your code must be as efficient as possible, given your answer to the previous question. (3 points)

Question 7:
The method `removeLeaves` is intended to remove every leaf node from a given BST. How many nodes should be removed the first time this method is called on the small BST? The second time? The third time? The fourth time? (1 point)

Question 8:
Complete the implementation of the recursive helper method `recursiveRemoveLeaves(Node v)`, which should remove all of the leaves in the subtree rooted at `v`. (3 points)

Extended Empirical Analysis of Sorting Algorithms

Your task for this part of the problem set, worth a total of 40 points, is to extend your previous empirical study of sorting algorithms.

- Add some advanced sorting algorithms to your empirical study: mergesort, quicksort, and heapsort. For quicksort, you will need to consider three versions: the standard version that chooses the pivot value as the first element of the subarray being sorted, a version that chooses a pivot randomly from among all of the elements of the subarray being sorted, and the "median of three" pivot choice which chooses the pivot as the middle value of the first, middle, and last elements of the subarray. A good implementation will support all three without replicating large amounts of code by having the latter two move their chosen pivot to the first position then applying the standard quicksort.
- You are permitted to use or base your code on existing algorithm implementations. For any code that is not entirely your own work, you must be sure you are not violating any copyrights, and you must cite the source(s) both in your code and in your writeup.
- Expectations are the same as they were for the earlier study in terms of coding the algorithms, running tests to gather timings and comparison counts, presenting those timings in tabular and graph formats, and a writeup that states your expectations for the results, what you observed, and how these align. For each algorithm and problem size, you should test with random data, sorted data, reverse sorted data, and nearly sorted data.
- As before, avoid recursive implementations for efficiency purposes. This is especially true for quicksort when the worst-case behaviors come up. You can probably get away with recursion for mergesort.
- Run each case several times, as you did before. Choose your range of problem sizes so that the largest cases run for 5-10 minutes at most. Choose a range of 6 to 10 powers of two for problem sizes. You will have a large number of runs to make and it will take a significant amount of time even with this restriction. Plan accordingly.
- Just stating "the graph looks like $O(n \log n)$" isn't good enough to match your timing and comparison count results to the theory. Show how as n changes, the trend grows like $n \log n$ or n^2 or whetever is appropriate.

Written Problems

In college admissions, one measure of similarity between schools is the number of "cross admits" (*i.e.*, students who applied to and were offered admission to both). A large number of cross admits would mean the schools are seen by college-bound students as being similar. Your task is to take the lists of students admitted to college A and B and count the number of names that appear in both. Your input consists of two arrays: `A[0...n-1]` contains the names of n students admitted to college A, and `B[0...m-1]` contains the names of m students admitted to college B. For example, for the two unreasonably small arrays below, 2 should be returned as the answer, because students "D. Chambers" and "I. Jones" were the only ones in both arrays. You may assume there are no duplicate names in the arrays, and that no two students exist with the same name.

```
A[0...7] = {"S. Cooper", "D. Chambers", "H. Stone", "M. Brady",
            "A. Fonzarelli", "I. Jones", "P. Venkman", "L. Organa"}

B[0...4] = {"I. Jones", "M. McFly", "F. Gump", "D. Chambers", "M. Simpson"}
```

Question 9:

You could easily write a brute force algorithm solving this problem in $\Theta(nm)$ time. But the number of students in both arrays is very large, and so an asymptotically more efficient algorithm is required. Write such an algorithm. (8 points)

ALGORITHM NUMCROSSADMITS(A, B)
//Input: $A[0..n-1]$, admits to college A
//Input: $B[0..m-1]$, admits to college B
//Output: the numbers of names that appear in both A and B

Question 10:

Give the Θ efficiency class of your algorithm and briefly explain how you arrived at this bound. (3 points)

Major League Baseball teams are deep into spring training and will be starting the season on the day this problem set it due, so let's consider a problem involving baseball statistics. Suppose you have a large array of numbers representing the batting averages of all players in a league who played in some minimum number of games, *e.g.*:

$$A = .312, .288, .185, .300, .276, .297, .307, .330.$$

Your task is to find the pair of batting averages that are closest together in value.

Question 11:
Develop pseudocode for an algorithm to solve this problem the operates in better than $\Theta(n^2)$ time. (8 points)

Question 12:
Give the Θ efficiency class of your algorithm and briefly explain how you arrived at this bound. (3 points)

Read Section 8.1 of Levitin to learn all about the robot coin collection problem. There, you will find a bottom-up dynamic programming solution to this problem.

Question 13:
Complete the pseudocode below that uses a **recursive exhaustive search algorithm** (**not** dynamic programming) to solve the robot coin collection. (6 points)

ALGORITHM RobotCoinCollection(r, c, C)
//Input: r, c, starting row and column position
//Input: $C[1..n, 1..m]$ matrix of 0 and 1 values indicating if a coin exists at each position
//Output: the max number of coins collected when the robot reaches (n, m)

Question 14:
Rewrite your algorithm from above so that it uses **top-down dynamic programming** to get a recursive algorithm that is more efficient. (8 points)

ALGORITHM RobotCoinCollection($r, c, C, sols$)
//Input: r, c, starting row and column position
//Input: $C[1..n, 1..m]$ matrix of 0 and 1 values indicating if a coin exists at each position
//Input: $sols[1..n, 1..m]$ initialized to all -1
//Output: the max number of coins collected when the robot reaches (n, m)

Question 15:
Give the Θ efficiency class of your top-down dynamic programming algorithm and briefly explain how you arrived at this bound. (3 points)

Read Section 8.3 of Levitin to learn all about optimal binary search trees.

Question 16:
Answer Levitin Exercise 8.3.1, p. 303. (3 points)

Question 17:
Answer Levitin Exercise 8.3.5, p. 303. Justify your answer. (3 points)