COMP 285 (NC A&T, Fall '22) Homework 3 (100 pt)

Fun with Sorting, Data Structures, and Master Theorem

###### Due 09/27/22 @ 11:59PM ET

Submitting

In order to submit this assignment, you must download your assignment as a ZIP file and upload it to Gradescope. There will be a written component and a coding component.

For the written component, you will write your answers in the corresponding `.txt` files.

For the coding component, you will write your solution in `answers.cpp`. We will use an autograder to test your solutions - the results of which can be seen after submitting to Gradescope. Note that you have unlimited tries to submit. In order to receive full credit, you must also provide documentation on your approach, see the commented sections in the `answers.cpp` file.

Question 0: Feedback is a Gift! (Optional/Extra Credit)

We are already nearing the midpoint of the semester! In advance of the midpoint, we'd love to collect feedback on COMP 285 so far so that we can adapt, grow, and better the course for the second half.

Your inputs to this survey will not influence your grades in the class in any way. The survey data will only be used in aggregated form without identification of individuals.

You can fill out the survey here.

If >=80% of the class fills out the survey, we will add +1% to everyone's final grade.

Question 1: Interview Practice: Finding Matching Elements (10 pts)

Rob is working on a search algorithm at Foogle. He is given an array *input* of *n* integers that is sorted in ascending order and contains only unique elements. That is to say, if his array is:

$$a = [a_1, a_2, a_3, \ldots, a_{n-1}, a_n], \text{ where } a_1 < a_2 < a_3 < \ldots < a_{n-1} < a_n$$

His project is to search the above array and find the element such that its index is equal to its value, or returns that such *i* does not exist. That is to say, the algorithm finds *i* such that $i = a_i$.

a. (5 pt) Rob proposes the following pseudo-code for an algorithm that solves the problem described above.

Describe in plain English how Rob's algorithm works, and provide the big-Oh running time.

`Write your answer in q1.txt`.

b. (5 pt) After looking at Rob's proposed algorithm, you both realize you should take advantage of the fact that the input array is sorted. Give a `O(log n)` algorithm that solves the same problem (e.g., it finds the index *i* such that $i = a_i$ or returns that such *i* does not exist). If there are multiple such indices, your algorithm can return any of them. Your answer must be detailed pseudocode.

Consider a decrease-and-conquer approach, similar to binary search! If you'd like to refresh your memory on binary search, see this video.

`Write your answer in q1.txt`.

Question 2: Master Theorem Practice (20 pt)

In class, we discussed the Master Theorem, a theorem that allows us to reason about complexities of recursive functions. The purpose of this problem is to cement our understanding of how it works and what it means.

As a refresher, here is the Master Theorem: - Suppose that $a >= 1$, $b > 1$, and d are constants independent of n. - Suppose $T(n) = a * T(n/b) + O(n^d)$

Then it follows:

a. (20 pt) For each of the following runtimes $T_x(n)$ for $x = 1, 2, 3... 10$, either state the runtime or specify that the Master Theorem cannot apply.

`Write your answers in q2.txt`.

Question 3: Climb Jaden, Climb! (20 pt)

Jaden loves to hike! Being the extremely data-driven individual that he is, Jaden decides to mark his elevation at various points during his hike in various mountain ranges he encounters.

He gives his dataset to you and asks you to help him write an algorithm `findMountainTop` which, when given a vector called `elevations` (of the elevations Jaden recorded), it returns the index of the highest measured elevation.

Make sure your algorithm handles the following:

- If `elevations.size() < 3`, you should throw `std::invalid_argument` since there is no highest elevation.
- All of your inputs have an `i` with `0 < i < elevations.size() - 1` such that:
    - `elevations[0] < elevations[1] < ... < elevations[i-1] < elevations[i]`
    - `elevations[i] > elevations[i+1] > ... > elevations[eleveations.size() - 1]`

Here are a few examples:

```
 // Should return 1, since this is the index of 1.
findMountainTop({0, 1, 0});
// Should return 1, since this is the index of 2.
findMountainTop({0, 2, 1, 0});
// Should return 1, since this is the index of 10.
findMountainTop({0, 10, 4, 2});
// Should throw std::invalid_argument since there is no peak.
findMountainTop({0, 1});
```

Optimal Solution

There is a straight-forward `O(n)` solution. However, using decrease-and-conquer (think, binary search), there is also a `O(log n)` solution.

Implement the easier of the two and submit your assignment. I then encourage you to try and implement the `O(log n)` solution.

a. (20 pt) **`Write your solution in answers.cpp`**.

Question 4: Balanced Parentheses Pt. 2 (10 pt)

In class, we went over the balanced parentheses problem. As a refresher, we wrote a function that, given a string containing just char '(' and char ')', returned whether or not the parentheses were balanced. By balanced, we meant that every open parentheses had a matching closing parentheses. For example:

- `()` -> returned `true`

- `)(` -> returned `false`
- `()(()())` -> returned `true`
- `(()` -> returned `false`

If you recall, we used a stack to keep track of the open parentheses. You can find the code we wrote together in `answers.cpp`. For this question, you will build upon what we wrote together.

a. (6 pt) Alex comes to you and says, "Wait! But what about the other open/close parentheses characters?" You realize that Alex is right - our implementation isn't well-generalized for `{}` and `[]`! Alex shows you the following test cases:

- `({}[])()` -> should return `true`
- `{}[(){]` -> should return `false`
- `[[]{}])` -> should return `false`
- `{[]}({})[]` -> should return `true`

You both realize you can modify the current implementation (`balancedParenthesesV2`) with only a handful of lines of code so you both get to work!

`Write your solution in answers.cpp` .

b. (4 pt) What is the most optimal Big-Oh time complexity of this algorithm and what is the space complexity of this algorithm? Assume `N` represents the length of the input string.

`Write your answer in q4.txt` .

Question 5: Finding The Mode! (20 pt)

Given an array of integers of size `n`, return the most common element.

The most common (mode) element is the element that appears more than `⌊n / 2⌋` times. You may assume that the most common element always exists in the array.

A few examples:

```
 // Returns 3 since it occurs 3 > ⌊3 / 2⌋ = 1 times.
findTheMode({3, 2, 3});

// Returns 2 since it occurs 4 > ⌊7 / 2⌋ = 3.
findTheMode({2, 2, 1, 1, 1, 2, 2});

// Returns 1 since it occurs 1 > ⌊1 / 2⌋ = 0
findTheMode({1});
```

Your solution can only use at most one for-loop.

In general, during an interview, its actually okay to start with a "less optimal solution" and implement that (as long as it works), then simply discuss some ideas as to how you'd implement a more efficient solution. So, you can start with a double for-loop solution to have a working solution; however, that should be your starting point. Your final submission can only rely on one for-loop.

a. (15 pt) **`Write your solution in answers.cpp`** .

b. (5 pt) What is the Big-Oh runtime of your algorithm and what is the space complexity of your implementation?

`Write your answer in q5.txt` .

Question 6: Behavorial Interviews, Career Prep Module #2 (20 pt)

Thank you Denzel for the presentation on Behavorial Interviews!

Let's put this into practice as part of our career prep work! Remember the **STAR method**: - *Situation*: Set the scene and give the necessary details for your example - *Task*: Describe what your responsibility was in the situation. - *Action*: Explain exactly what steps you took to address it - *Result*: Share what outcomes your actions achieved and provide any data driven results.

Answer each of the following common interview questions using the STAR method. **Focus on concision; you should only need 1-3 sentences for each letter in STAR.**

1. (4 pt) Tell me about a time you convinced someone to change their mind.
2. (4 pt) Tell me about a time you made a mistake.
3. (4 pt) Tell me about a time you resolved some kind of conflict.
4. (4 pt) Tell me about a time you received constructive feedback and how you dealt with it.
5. (4 pt) Tell me about a time you were a leader or took the initiative to solve a problem.

The goal is that by the end of this assignment you'll have a cheat sheet you can reference and be equipped with before any behavorial interview you walk into.

I recommend saving your responses to these questions in your own document and adding to it as you begin/continue interviewing. You'll be asked many different behavorial questions and keeping track of your ideal answers (with STAR framework) in a document is a great/easy way to stay prepared.

`Write your answers in q6.txt`.