COMP 285 (NC A&T, Spr '22) Homework 2

Due. Tuesday, February 1st, 2022 @ 11:59 PM!

Homework Expectations: Please see the Homework part of the Course Website (comp285.ml/policies) for guidance on what we are looking for in homework solutions. We will grade according to these standards, and you should cite all sources you used outside course material.
What we expect: Make sure to look at the "**We are expecting**" blocks below each problem to see what we will be grading for in each problem!

Exercises The following questions are exercises. We encourage you to work with a group and discuss solutions to make sure you understand the material.
Points This assignment is graded out of 100 points. However, you can get up to 120 points if you complete everything. These are not bonus points, but rather points to help make-up any parts you miss.

Formal Fun with Recursion and Big-Oh

Written Problems The following questions are to be submitted in written/typed form to gradescope.

1 Exercise: Getting Basic with Big-Oh

(20 pt.) Dr. Wang is is preparing for a review lecture for the students who want to get a bit more confortable with big-Oh notation. given how studious you've been, she asks for your help on a few questions.

1.1 Can you identify big-Oh?

(10 pt.) Write the functions below that are $O(n^2)$?

You don't need to provde a prove or explanation, but you probably want to convince yourself you're right.

1. $g_1(n) = 4n + 10$
2. $g_2(n) = 4n^2 + 10$

3. $g_3(n) = 4n^3 + 10$
4. $g_4(n) = n \log n$
5. $g_5(n) = \sin(n) + 5$
6. $g_6(n) = 4^n$
7. $g_7(n) = (285^3)^4$

[We are expecting: A list of which functions are $O(n^2)$. No explanation is required.**]**

1.2 Formal Definition of Big-Oh

(10 pt.) Serena is having a great time on COMP 285! She wants to prove, from the definition of big-Oh, that $f(n) = O(g(n))$, where $f(n) = 6n$ and $g(n) = n^2 - 3n$.[1]

As a reminder, here is the formal definition of big-Oh is:

$$f(n) = O(g(n))$$
$$\iff$$
$$\exists c, n_0 \text{ such that } \forall n > n_0, f(n) \leq c \cdot g(n)$$

1.2.1 First proof...

(5 pt.) Serena claims she has a proof in the definition above where $c = 1$. Give a possible proof using $c = 1$.

1.2.2 Second proof...

(5 pt.) Serena realizes that she'll never get inputs to her algorithm that are smaller than 4. As such, she says she wants $n_0 = 4$. Give a possible proof using $n_0 = 4$.

[We are expecting: For both parts, a short but complete proof. That is, write out the formal definition of big-Oh plugging in the values for $f(n), g(n), n_0$ and c as given in the problem statement and that you chose.**]**

[1] For this problem, it might help you to plot the functions you're trying on Wolfram Alpha. For example, here is the plot of $6n$ and $n^2 - 3n$.

2 Interview Practice: Comparing Big-Oh, Big-Omega, and Big-Theta

(10 pt.) In industry, it's quite common that you have to compare the running times of different algorithms you're implementing [2]. In this exercise, we'll practice a few common running times (I've personally seen all of these show-up on interviews) and compare them.

For each blank below, indicate whether A_i is in O, Ω, or Θ of B_i . More than one space per row can be valid.[3]

[We are expecting: All spaces in the below table should have an 'x' if A_i is big-[COLUMN] of B_i**]**.

A	**B**	O	Ω	Θ
$\log n$	n			
$2n^3$	2^n			
$\log_{10} n$	$\log_2 n$			
$n^{0.1}$	$(0.1)^n$			
n^2	$4^{\log_2 n}$			

[2] If you'd like specific examples, come to my student hours!

[3] One way to get a sense for which function is bigger is to plot them. For example, this plots $\log_2 n$ and n from 0 to 10.

3 Exercise: Practice with Recurrence Relations

(10 pt.) After settling down after her fourth move (joining us in Greensboro), Sloane wants to practice some **recurrence relations** with you. To refresh your memory, she jots down the **Master Theorem** below:

Theorem 1 (Master Theorem)**.** *Let* $T(n) = a \cdot T\left(\frac{n}{b}\right) + O(n^d)$ *be a recurrence where* $a, b > 1$*, Then,*

$$T(n) = \begin{cases} O(n^d \log n) & \text{if } a = b^d \\ O(n^d) & \text{if } a < b^d \\ O(n^{\log_b a}) & \text{if } a > b^d \end{cases}$$

For each of the below, provide the big-Oh running time:

(a) **(4 pt.)** $T(n) = 18T\left(\frac{n}{9}\right) + O(\sqrt{n})$

(b) **(2 pt.)** $T(n) = 5T\left(\frac{n}{4}\right) + O(n^2)$

(c) **(3 pt.)** $T(n) = 16T\left(\frac{n}{2}\right) + O(n^4)$

(d) **(1 pt.)** $T(n) = 4T\left(\frac{n}{3}\right) + O(n)$

[We are expecting: Use Master Theorem to calculate the tigheter upper bound for each of these recurrence relations in big-Oh notation.**]**

4 Interview Practice: Modified MergeSort

(10 pt.) In most interviews, you won't be asked to reproduce algorithms we've learned in class. Instead, you'll be asked to implement modifications of these algorithms. In this exercise, we'll see how changing the size of the subproblems affects MergeSort.

Your friend Malia gives you a new version of `MergeSort` that she coded after class. She claims this runs asymtotically better than the version we coded.

This is her code, where `merge` is left unchanged.

```
// MergeSort algorithm!
std::vector<int> mergeSortThirds(const std::vector<int>& input) {
    const std::size_t n = input.size();
    if (n <= 1) {
        return input;
    }
    const std::vector<int> left = mergeSortThirds(
        {input.begin(), input.begin() + n / 3});
    const std::vector<int> middle = mergeSortThirds(
        {input.begin() + n / 3, input.begin() + 2*n / 3})
    const std::vector<int> right = mergeSortThirds(
        {input.begin() + 2*n / 3, input.end()});
    const std::vector<int> temp = merge(L, M);
    return merge(temp, right);
}
```

Write down the recurrence relation and runtime for this version of `MergeSort`.

[We are expecting: A recurrence relation for the runtime, as well as the closed-form big-Oh runtime.[4]**]**

[4] Once you have the recurrence relation, you can probably use the Master Theorem.

5 Industry Practice: What's the big-Oh of code?

(20 pt.) In industry, you'll often times be expected to analyze existing code. In this question, you'll get a bit of practice looking at unfamiliar C++ code and identifying the running time.

5.1 A loop that doubles each time!

(5 pt.) Your friends Victor and Dezmon are working on their new internship. They show you the below code. What is the big-Oh running time? Please justify your answer.

```
1 void doSomething(const std::vector<int> input) {
2     for (int i = 1; i < input.size(); i *= 2) {
3         std::cout << input[i] << std::endl;
4     }
5 }
```

[We are expecting: You should provide a big-Oh running time $O(\cdots)$ and a short sentence explaining why this is the running time.**]**

5.2 Counting down by 10...

(5 pt.) Your friends Liam and Miah are working on their new jobs at Foogle. They show you the below code. What is the big-Oh running time? Please justify your answer.

```
1 void doSomethingElse(const std::vector<int> input) {
2     int z = input.size() - 1;
3     std::cout << z << std::endl;
4     while(z >= 10) {
5         std::cout << input[z] << std::endl;
6         z /= 10;
7     }
8 }
```

[We are expecting: You should provide a big-Oh running time $O(\cdots)$ and a short sentence explaining why this is the running time.**]**

5.3 Fun with Recursion

(10 pt.) Your friends Ryan and Jordan are working with the NSA. They show you the below code. What is the big-Oh running time? Please justify your answer.

```
std::string doSomethingSecret(const std::vector<int> input) {
    if (input.size() % 7 == 0) {
        return "Bzzzt!";
    }
    // Remove the last element in-place. This is O(1).
    input.pop_back();
    return doSomethingSecret(input);
}
```

[We are expecting: You should provide a big-Oh running time $O(\cdots)$ and a short sentence explaining why this is the running time.**]**

Coding Problems The following questions are to be submitted as a ".zip" file on Gradescope. For complete submissions instructions, please see HW1.

6 Coding

(50 pt.) After completing the written portion of the assignment, you should submit to Gradescope.

You can get your starter code for the coding portion here.

Note there are **two** coding questions. Each part is worth 20 points for correcness. Your code will also be reviewed for style (5 points) and documentation (5 ponts), for a total of 50 points for the coding portion.

Please reference the `README.md` included in your starter code for detailed instructions.

Submitting the Assignment

This assignment is a combination of written and programming questions. Both portions of the assignment should be submitted through Gradescope.

The "Homework 2: Fun with Recursion and Big-Oh" assignment is the written portion, for which you should submit a **typed** response to the non-coding questions (questions 1-5). Each response should clearly be marked with its corresponding number. You are free to use the provided templates, print the questions and write your answers, or to simply type your responses on a blank document (whatever works for you).

The "Homework 2: Coding" is the programming portion of the assignment. For this portion, download the ".zip" file from replit and upload this ".zip" file as your answer to Gradescope. You can upload the assignment as many times as you want.

To summarize, do the following:

1. Submit your **typed** responses to the non-coding portion of the assignment on Gradescope for "Homework 2: Fun with Recursion and Big-Oh"
2. Submit your **code files** as a ".zip" to Gradescope for "Homework 2: Coding".
3. Fill out this **Google Form** (https://forms.gle/ELVUbA9Tawrr4oFU8) to help me track your thoughts on the homework.