**COMP 285 (NC A&T, Spr ‘22)Homework 2**

#### Due.

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

#### Homework Expectations:

Please see the [Homework](https://www.comp285.ml/homework/#general-homework-information) part of the Course Website ([comp285.ml/policies](https://www.comp285.ml/policies/#collaboration-policy-and-honor-code)) 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.

# Exercise: Getting Basic with Big-Oh

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.

## Can you identify big-Oh?

Write the functions below that are ?

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

## Formal Definition of Big-Oh

Serena is having a great time on COMP 285! She wants to prove, from the definition of big-Oh, that , where and .[[1]](#footnote-30)

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

### First proof...

Serena claims she has a proof in the definition above where . Give a possible proof using .

### Second proof...

Serena realizes that she’ll never get inputs to her algorithm that are smaller than . As such, she says she wants . Give a possible proof using .

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

In industry, it’s quite common that you have to compare the running times of different algorithms you’re implementing [[2]](#footnote-36). 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 is in , , or of . More than one space per row can be valid.[[3]](#footnote-37)

.

| **A** | **B** |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

# Exercise: Practice with Recurrence Relations

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 be a recurrence where , Then,*

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

# Interview Practice: Modified MergeSort

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.

# Industry Practice: What’s the big-Oh of code?

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.

## A loop that doubles each time!

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.

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

## Counting down by 10...

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.

void doSomethingElse(const std::vector<int> input) {  
 int z = input.size() - 1;  
 std::cout << z << std::endl;  
 while(z >= 10) {  
 std::cout << input[z] << std::endl;  
 z /= 10;  
 }  
}

## Fun with Recursion

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.

void 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);  
}

#### Coding Problems

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

# Coding

After completing the written portion of the assignment, you should submit to [Gradescope](https://www.gradescope.com/courses/350304).

You can get your starter code for the coding portion [here](https://replit.com/team/COMP285/HW2-Code).

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](https://www.gradescope.com/courses/350304).

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](https://www.gradescope.com/courses/350304). 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.

1. For this problem, it might help you to plot the functions you’re trying on Wolfram Alpha. For example, [here](https://www.wolframalpha.com/input/?i=plot+5n+and+n%5E2-3n+from+0+to+10) is the plot of and . [↑](#footnote-ref-30)
2. If you’d like specific examples, come to my student hours! [↑](#footnote-ref-36)
3. One way to get a sense for which function is bigger is to plot them. For example, [this](https://www.wolframalpha.com/input/?i=plot+log_2+n+and+n+from+0+to+10) plots and from to . [↑](#footnote-ref-37)