# COMP 285 (NC A&T, Fall '22) Homework 2 (80 pt)

**Formal Fun with Big-Oh and Recursion**

## ###### Due 09/15/22 @ 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 1: Big- Oh & Omega & Theta, Oh My! (20 pt)

**a. (10 pt)** Let's warm up with some Big-Oh practice! For each of the listed functions, **answer yes or no if $g_x(n) = O(n^2)$** (where `x = 1, 2 ... , 9, 10`. You don't need to prove it or provide an explanation, but you probably want to convince yourself you're right.

*Note: For this question, we are using the mathematical defintion of Big-Oh which means we must evaluate if $g_x(n)$ is upper-bounded by $n^2$.*

**Write your answers in `q1.txt`.**

**b. (10 pt)** Now let's re-introduce Big-Oh's siblings - Theta and Omega. Remember from lecture, these represent different bounds (Oh for upper, Omega for lower, Theta for tight).

In industry, it's common that you have to compare the running times of different algorithms you're implementing. 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 of the pairs of functions below, list each of the asymptotic bounds that apply to the relationship: `A = ?(B)`. When writing your answer, use 'Oh' for Big-Oh, 'Omega' for Big-Omega, and 'Theta' for Big-Theta. More than one symbol could apply.

For example, if `A = 1` and `B = n`, then you would write 'Oh' for Big-Oh because `A = O(B)` because the linear function `n` is an upper bound for the constant `1`. If `A = n` and `B = 1`, then the answer would be 'Omega' for Big-Omega because `A = G(B)` signifying B is a lower bound of A. Lastly, if `A=n` and `B=n` you would list `Oh`, `Omega`, `Theta` since B is a tight bound then, by definition, B is also an upper and lower bound for A.

As a helpful hint, one way to get a sense for which function is bigger is to plot them. For example, this graph plots $\log_2 n$ and n from 0 to 10. When in doubt, plot it out!

**Write your answers in `q1.txt`.**

*Note: For this question, we are using the mathematical defintion of Big-Oh, Big-Theta, and Big-Omega as upper, tight, and lower bounds.*

## Question 2: Code-ified Big-Oh (10 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.

**a. (3 pt)** Kyndell just wrapped up her internship at Microsoft and wanted to show you some the code she wrote!

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

Being an efficient programmer, she wants to verify the Big-Oh runtime of this snippet. She asks, "What is the tightest big-Oh running time?" Please justify your answer.

**Write your answer in `q2.txt`.**

*Hint: the answer is not O(N). Of the N elements in the vector, how many are we actually printing? How many iterations are we completing?*

**b. (3 pt)** She shows you the next snippet of code and asks you again, "What's the tightest Big-Oh runtime?" 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;
    }
}
```

**Write your answer in `q2.txt`.**

*Hint: the answer is not O(N). Of the N elements in the vector, how many are we actually printing?*

**c. (4 pt)** Impressed with the code she wrote, you start to get curious about interning at Microsoft and inquire to interview with them. Luckily, your two classmates Kyndell and Shang have your back! They show you some code they wrote for their Microsoft interviews for you to study!

```
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);
}
```

*(You know this code is from an interview setting and not used in the real world because no one uses recursion in practice!)*

What is the tightest Big-Oh runtime of this code snippet? Please justify your answer.

**Write your answer in `q2.txt`.**

# Question 3: Algorithmic Trading and Maximizing Profits (20 pt)

Kyle comes to you with his great idea for making money. He wants to leverage your knowledge of algorithms from COMP 285 to write a function that tells you when to buy, hold or sell stocks in order to maximize your profit!

Before jumping into code, you both agree to look at some data. Each of the three vectors below represent the prices of an

individual stock over the course of a single work week. (Monday, Tuesday, Wednesday, Thursday, Friday corresponding to the indices 0, 1, 2, 3, and 4).

For each example, on **which day should you buy and which day should you sell in order to maximize your profit and what is your maximum profit?**

*Note: you must first buy the stock before you can sell it! You are only permitted to buy once and sell once.*

**a. (2 pt)** AAPL = `[7, 1, 5, 3, 6]`

**b. (2 pt)** GOOGL = `[2, 5, 3, 6, 9]`

**c. (2 pt)** META = `[5, 4, 3, 8, 7]`

**Write your answers in `q3.txt`**

You realize you'll need millions and millions of training examples for your model so coming up with these by hand is out of the question.

Kyle turns to you and says you can code up a function to do the work instead!

**d. (14 pt)** Write a function `maximumProfit` that takes in a vector of integers representing stock prices and returns an integer value representing the maxmimum profit that can be achieved.

The following constraints apply: - You must first buy the stock before you can sell it. - You are only permitted to buy once and sell once. - The input vector of prices can be of any length that is greater than 1. - There might be stocks that perform too poorly to make a profit and `maximumProfit` should be able to handle that! - For example, SNAP stock lost $1.3B in value after Kylie Jenner said she wasn't using it anymore! - Let's say the SNAP stock prices during that time looked like `[10, 5, 3, 1]`. In this case, `maximumProfit` **should not perform any transactions and return 0.** - The runtime must be at least $O(n^2)$. For those who are curious, yes an $O(n)$ solution exists where n is the length of the prices vector. The $O(n)$ solution is encouraged but not required to receive credit.

**Write your solution in `answers.cpp`.**

# Question 4: How Many Sevens? (20 pt)

This problem is meant to be a refresher on recursion.

Write `howManySevens`, which takes in an integer and returns the number of times 7 appears as a digit within the number. Examples:

```
std::cout << howManySevens(123456890) << std::endl;  // should output 0
std::cout << howManySevens(777777) << std::endl;  // should output 6
std::cout << howManySevens(17374) << std::endl;  // should output 2
```

**In order to receive full credit for this problem, you must use recursion.** i.e. using `=`, `for`, `while`, etc. is prohibited.

Hint: recall the `%` and `/` operators:

```
123 % 10 // evaluates to 3
123 / 10 // evaluates to 12
```

**Write your solution in `answers.cpp`.**