COSC-211: DATA STRUCTURES
HW2: ASYMPTOTIC ANALYSIS

Due Thursday, March 4, 11:59pm ET

Reminder regarding intellectual responsibility: This is an individual assignment, and the work you submit should be your own. Do not look at anyone else's work, and do not show anyone your work (except for me and the course TAs).

This is a written assignment. I recommend typing up your solutions using LaTeX, which is a typesetting language that allows you to format math nicely (among many other functions). There's a lot of free software available to help you write LaTeXcode; my personal favorite is TeXstudio, available to download at https://www.texstudio.org/. You are also welcome to use your favorite word processor. If you write your solutions by hand, please be neat and legible!

1 Your Tasks

1. Rank the following functions from smallest to largest according to their big-O complexity. That is, order them based on their asymptotic growth rate. For example, you could write $n < n^3$ since $n \in O(n^3)$, but $n^3 \notin O(n)$. Some of the functions might be in the same big-O class. You do not need to do any formal proofs.

$$n^{100} \qquad \lg n \qquad 2^n \qquad 2^{\lg n} \qquad 4 \qquad \sqrt{n} \qquad n! \qquad 100n$$

2. Let $f(n) = 2n^2 + 6n - 9$. Use the definition of big-O to show that $f(n) \in O(n^2)$.

3. Java's `Queue` class provides a method called `contains`. The `contains` method takes an item as an input parameter and returns `true` if that item appears somewhere in the `Queue`, and false if not.

Suppose we implement our queue using a linked list (as in `WrapperQueue` from HW1). We will imagine that each node in our linked list contains an `E` called `data` and a pointer to another node called `nextNode`, and that the linked list starts with a node called `header`, whose `data` is null. Now suppose we add a `contains` method to our queue, as follows:

```
1 public boolean contains(E element) {
2    if (header.nextNode == null) return false;
3    Node currentNode = header.nextNode;
4    while (currentNode != null) {
5        if(currentNode.data.equals(element)) return true;
6        currentNode = currentNode.nextNode;
7    }
8    return false;
```

What is the worst case big-O runtime of `contains` in terms of n (the number of items in the queue)? What about the best case? Explain your answers.

4. Consider two different options for how one might implement a stack:

Stack version 1: items are stored in an array. To push, add the new item to the next available position in the array. To pop, remove the item in the last (used) position in the array. Don't ever resize the array; if you try to push and there's no available space, the item just doesn't get added.

Stack version 2: items are stored in an array. The push and pop operations work the same as in version 1, except that if you try to push and there's no available space in the array, create a new array whose size is one larger than the previous array, copy over all of the items from the previous array, and then add the new item to the (newly created) available position.

(a) Give the worst-case big-O runtimes for the `push` and `pop` operations for each of the above stack implementations. That is, fill in the following table of big-O runtimes. For each entry, give a short explanation of how you found your answer.

	Stack version 1	Stack version 2
`push()`		
`pop()`		

(b) Now we're going to consider the `QueueOfStacks` implementation from HW1. In our solution, the `add` operation works as follows.[1]
All of the items in our queue are stored in what we'll call the primary stack. To add an item to the queue:

1. Create a second temporary stack to hold items popped off the primary stack.
2. Pop all items off the primary stack and push them onto the temporary stack.
3. Push the new element onto the (now empty) primary stack.
4. Pop all items off the temporary stack and push them back onto the primary stack, on top of the newly added item.

And our `remove` operation works by simply popping the item on top of the primary stack.

Give the worst-case big-O runtimes for the `add` and `remove` queue operations, assuming each of the above stack implementations. That is, fill in the following table of big-O runtimes. For each entry, give a short explanation of how you found your answer.

[1] Your implementation on HW1 may have been different, and this is fine. There's more than one way to do this. Use our approach for solving this problem, even if you did something different on HW1.

	Stack version 1	Stack version 2
`add()`		
`remove()`		

2 Submit your work

Submit your work via Gradescope.

This assignment is due Thursday, March 4, 11:59pm ET.