COSC-211: DATA STRUCTURES
HW3: ASYMPTOTIC ANALYSIS
Solutions

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.

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

Solution: From smallest to largest: $O(4) \subset O(\lg n) \subset O(\sqrt{n}) \subset O(100n) = O(2^{\lg n}) \subset n^{100} \subset O(2^n) \subset O(n!)$

2. Let $f(n) = 3n^2 + 5n - 12$. Prove that $f(n) \in O(n^2)$.

Solution: Let $c = 4$ and $n_0 = 5$. For $n \geq n_0 = 5$, we have $n^2 \geq 25$. Plugging into $f(n)$, we have that for $n \geq n_0 = 5$, $3n^2 + 5n - 12 \geq 3 \cdot 25 + 5 \cdot 5 - 12 = 88 > 0$. We also have:

$$\begin{aligned} 3n^2 + 5n - 12 &< 3n^2 + 5n \\ &\leq 3n^2 + n \cdot n \\ &= 4n^2 \\ &= cn^2, \end{aligned}$$

where the first inequality results from adding 12, thereby increasing the expression, and the second line results from the fact that $n \geq 5$. We have shown that there exist positive constants c and n_0, namely $c = 4$ and $n_0 = 5$, such that for all $n \geq n_0$, $0 \leq f(n) \leq cg(n)$. Hence $f(n) \in O(g(n))$.

3. Prove that $O(n) \subseteq O(n^2)$. The notation $\subseteq$ means "is a subset of or equal to". That is, we want to show that every function in $O(n)$ must also be in $O(n^2)$. [Hint: this is a "for all" claim. Start by considering an arbitrary function $f(n)$, where all you know about $f(n)$ is that it's in $O(n)$.]

Solution: We begin with an arbitrary function $f(n) \in O(n)$. From this, the definition of $O(n)$ tells us that there exist positive constants c and n_0 such that for all $n \geq n_0$, $0 \leq f(n) \leq cn$. Since $n_0 > 0$ and n_0 is an integer, we can also say that $n \leq n^2$ for all $n \geq n_0$. Hence $cn \leq cn^2$ for any positive constant c, and in particular for the constant c that we used in the definition of $O(n)$ above. Putting this together, we now have $0 \leq f(n) \leq cn \leq cn^2$ for all $n \geq n_0$. But this is exactly the definition of $O(n^2)$! So we now know that $f(n)$ must also be in $O(n^2)$.

Note that this argument didn't require us to choose a specific function $f(n)$. The argument works for *any* function that happens to be in $O(n)$.

4. Java's `Stack` class provides a method called `search`. The `search` method takes an `Object` as an input parameter and returns and `int` representing the location of that `Object` within the

stack (the top of the stack is 1), or -1 if the `Object` isn't in the stack.

Suppose we add a `search` method to our array-based `Stack` of `Book` objects and implemented it as follows:

```
1 public int search(Book b) {
2     for (int i = 0; i < top; i++) {
3        if (booklist[i].equals(b)) return i+1;
4     }
5     return -1;
```

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

Solution: The `search` function is $O(n)$ in the worst case and $O(1)$ in the best case.

Worst case: As soon as the `if` statement in line 3 evaluates to `true` (i.e., we find the book) the method returns, so our worst case is when the input `b` isn't in the stack. In this case, the body of the `for` loop takes constant time (the call to `equals` in line 3, plus the comparison `i < top` and the increment `i++` in line 2, all constant time operations). The loop executes n times—`i` runs from 0 to `top`. So the entire loop in lines 2-4 takes $O(1) * O(n) = O(n)$ time. The return statement in line 5 takes constant time, so overall we have $O(n) + O(1) = O(n)$.

Best case: The best case is that the book we're looking for is on the bottom of the stack (in position 0). In this case, we find the book on the first iteration of the for loop (when `i` is 0). All we do is set `i = 0`, compare `i < top`, index into `booklist`, call `equals`, and return. All constant work: the total runtime is $O(1)$.