COSC-211: DATA STRUCTURES
HW3: ASYMPTOTIC ANALYSIS

Due Friday, February 16, 9:00am

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. Please type up your solutions. I recommend 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 free to use your favorite word processor.

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 $f(n) = 2n^3 + 4n - 17$. Prove that $f(n) \in O(n^3)$.

3. Let $f(n) = \frac{1}{3} n \lg n$. Is $f(n) \in O(n)$? Prove or disprove.

4. Prove that $O(n) \subseteq O(n^2)$. [Hint: what you are trying to show here is that *for all* functions $f(n)$ that are in the set $O(n)$, $f(n)$ must also be in the set $O(n^2)$.]

5. In class, we have discussed both an array-based implementation and a Vector-based implementation for the Queue ADT. Some advantages of the Vector implementation include that it's easy to code up using the methods from the Vector class and that it allows us to use generics. We claimed that a disadvantage is that it's slower. Your job in this problem is to use asymptotic analysis to justify this claim.

(a) Fill in the following table with the worst-case and best-case asymptotic runtime for both the array-based wraparound Queue implementation (the code is posted on the course web page) and the Vector-based Queue implementation (the code was provided as part of HW2, and the source code for the underlying Vector methods is at the end of this assignment for reference). Give a short (1-sentence) explanation of each entry.

Operation	Array: Worst Case	Array: Best Case	Vector: Worst Case	Vector: Best Case
enqueue				
dequeue				
size				
isEmpty				

(b) Use your answers from part (a) to either support or refute the claim that the Vector-based implementation is slower than the array-based implementation.

2 Submit your work

Type up your solutions and bring a hard copy to class.

This assignment is on Friday, February 16, 9:00am.

Vector Methods

Below I have reproduced some of the source code for Java's Vector class. The full source code can be found at:
http://hg.openjdk.java.net/jdk7/jdk7/jdk/file/00cd9dc3c2b5/src/share/classes/java/util/Vector.java.

I encourage you to go take a look at it. You will find that the methods there look a bit different than what appears below; this is because I have cleaned up some of the methods to make them easier for you to read. Come talk to me if you're curious about something you see in the true source code that I omitted!

You will see variables `elementData` and `elementCount` referred to in the methods below. These are fields in the Vector class that represent the array of data and the number of elements currently stored in that array respectively.

```
public int size() {
   return elementCount;
}
```

```
public boolean isEmpty() {
   return elementCount == 0;
}
```

```
public boolean add(E e) {
   grow(elementCount + 1);
   elementData[elementCount++] = e;
   return true;
}
```

```
 private void grow(int minCapacity) {
   int oldCapacity = elementData.length;
   int newCapacity = 2 * oldCapacity;

   // This copies the contents of elementData into a new array,
   // element by element
   elementData = Arrays.copyOf(elementData, newCapacity);
}
```

```
public E remove(int index) {
   if (index >= elementCount)
      throw new ArrayIndexOutOfBoundsException(index);
   E oldValue = elementData[index];

   int numMoved = elementCount - index - 1;
   if (numMoved > 0) {
      // Shifts the values in positions [index+1...elementCount-1]
      // in elementData left by one position, closing the "hole"
      // left by removing the element at position index
            System.arraycopy(elementData, index+1, elementData,
                             index, numMoved);
   }
   elementData[--elementCount] = null;

   return oldValue;
}
```