CMSC 132 Quiz 3 Worksheet

The next quiz for the course will be on Tue, Jul 2. The following list provides more information about the quiz:

- The quiz will be a written quiz (no computer).
- The quiz will be in lab/discussion session.
- Closed book, closed notes quiz.
- Answers must be neat and legible. **You must use pencil.**
- Check the information available at http://www.cs.umd.edu/~nelson/classes/utilities/examRules.html

The following exercises cover the material to be included in this quiz. Solutions to these exercises will not be provided, but you are welcome to discuss your solutions with the TA or instructor during office hours.

Exercises

1. Give the asymptotic bound of the following functions:

   a. $\mathbf{n}^3 + \log(\mathbf{n})$ f(**n**) = O( )

   b. $2\mathbf{n}^4 + 500\mathbf{n} + \mathbf{n}^2$ f(**n**) = O( )

   c. $\mathbf{n}\log(\mathbf{n}) + \mathbf{n}^k$ f(**n**) = O( )

   d. $\mathbf{n}^n + 200\mathbf{n} + \mathbf{n}^2$ f(**n**) = O( )

   e. $2\mathbf{n}^4 + 500\mathbf{n} + \log(\mathbf{n})$ f(**n**) = O( )

   f. **1000** f(**n**) = O( )

2. List the following big-O expressions in order of asymptotic complexity (lowest complexity first).

   O(nlog(n)) O(log(n)) $O(n^2)$ $O(n^3)$

3. Calculate the asymptotic complexity of the code snippets below (using big-O notation) with respect to the problem size **n**.

   a. f(**n**) = O( )

```
i = 1;
while (i <= n) {
   if (i % 2 == 0) {
      System.out.println(i);
   }
   i = i + 1;
}
```

   b. f(**n**) = O( )

```
for (i = 1; i < n; i *= 2) {
     System.out.println(i);
}
```

   c. f(**n**) = O( )

```
for (int i = 0; i < n; i++) {
    for (int k = n; k <= n; k++)  {
       System.out.println("Computing");
    }
}
```

4. Implement the methods below based on the following Java class definitions. For recursive methods you may only add one auxiliary function, and you may not add any instance nor static variables.

```
public class LinkedList<T extends Comparable<T>> {
     private class Node {
           private T data;
           private Node next;

           private Node(T data) {
                 this.data = data;
                 next = null;
           }
     }

     private Node head;

     public LinkedList() {
           head = null;
     }

}
```

   a. Define a constructor that takes a **TreeSet<T>** as a parameter and initializes a linked list with the elements in the set. The new list must be sorted in increasing lexicographic order.

   b. Define a **RECURSIVE** method named **size** that returns the number of elements in the list. The prototype for this method is:

```
public int size()
```

   c. Define a **RECURSIVE** method named **inRange** that returns a **HashSet<T>** with the elements in the list that in the specified range. The range includes the lower and upper bound. The prototype for this method is:

```
public HashSet<T> inRange(T lowerBound, T upperBound)
```

   d. Define a **RECURSIVE** method named **remove** that removes all instances in the list that are equal to the target parameter. The prototype for this method is:

```
public void remove(T target)
```

   e. Define a **RECURSIVE** method named **positionOfElementInList** that returns a **TreeMap<T, Integer>** that maps each element of the list to its position in the list. The prototype for this method is:

```
public TreeMap<T, Integer> positionOfElementInList()
```