CSC 143

Exam 2

Short Answers

1) Circle among the following the types that are defined as interfaces within the java collections framework

Set ArrayList Collection Collections Iterator Map LinkedList TreeMap

2) Complete in big O notation (e.g. O(n)) the following table for the sorting algorithms studied in class

	Worst case	**Best** case
Insert sort	**$O(n^2)$**	**O(n)**
Merge sort	**O(n log n)**	**O(n log n)**
Quick sort	**O(n log n)**	**$O(n^2)$**

3) Show that $2n^2 + 6n + 1 = O(n^2)$. That is find two constants c and n_0 such that for any $n \geq n_0$, $2n^2 + 6n + 1 \leq cn^2$

Take c = 3. To get

$2n^2 + 6n + 1 \leq 3n^2$ or

we need $6n + 1 \leq n^2$

which is true for any n >= 7

Thus, c = 3, and $n_0 = 7$

4) You are given two algorithms P and Q that solve the same problem. You know that P = O(n) and $Q = O(n^2)$. Which algorithm is the fastest for n = 100? Explain.

We can't tell. The big O notation gives information about the algorithm for large n. We know that there exists n_0 such that for any n >= n_0, P grows at most like n, and Q grows at most like n^2. However, the value of n_0 is unknown from the information given. Moreover, the big O notation gives an upper bound. It could be that Q = O(1) as well. That is we can't even tell that P is more efficient that Q for large n (in practice however, the most stringent bound is given).

5) Complete the method below that returns the number of lines in a text file. Handle possible exceptions with a **throws** statement. Don't use any method from the File class or the LineNumberReader class.
Make sure that you implement efficient streams (i.e. use wrappers)

```
public int countLines(String filename) throws IOException {

    BufferedReader in =
        new BufferedReader(new FileReader(filename));

    int count = 0;

    while (in.readLine() != null) {

        count ++;

    }

    in.close();

    return count;

}
```

6) What would be the programming consequences of making all exceptions in Java checked exceptions?

Since any non trivial piece of code may throw exceptions, the code within a method would have to be written inside a try block followed by one or more catch blocks, or the method would have to list all of the exceptions that may be thrown. This would bloat the code and make the Java language very unattractive.

7) Why does the following method within a Java class generate a compile-time error? How would you fix it?

```
public void fileOperation() {
     try {
          FileWriter out = new FileWriter ("myFile.txt");
          // code omitted …
     }
     catch (Exception e) {
          System.out.println(e.getMessage());
     }
     catch (FileNotFoundException e) {
          System.out.println(e.getMessage());
     }
}
```

Since Exception is a base class of FileNotFoundException, any exception of type FileNotFoundException can be caught by the first catch block. It is never caught by the second catch block. The order of the two catch blocks should be reversed.

8) What is the asymptotic complexity of the following method? In other words, what is foo(n) in big O notation? Make sure that you explain why.

```
public int foo(int i) {
     if (i <= 1) {
          return 1;
     }
     else {
          return foo(i/2);
     }
}
```

Take $n = 2^k$. To evaluate f(n), foo is called k + 1 times. Each call is O(1). Therefore, f(n) = O(log n).

9) Consider the method bar given below. Note the differences between foo and bar. What is bar(n) in big O notation? Make sure that you explain why.

```
public int bar(int i) {
     if (i <= 1) {
          return 1;
     }
     else { // this is not the same as foo
          int a = bar(i/2);
          int b = bar(i/2);
          return a + b;
     }
}
```

Take $n = 2^k$. bar(n) requires evaluating bar(n/2) = bar(2^{k-1}) twice. Evaluating bar(n/2) requires evaluating bar($n/4 = 2^{k-2}$) twice and so on.

The number of calls of bar is $2^0 + 2^1 + 2^2 + \ldots + 2^k = 2^{k+1} - 1 = 2n - 1$. Thus bar(n) = O(n).

10) Given the following stack S that contains strings as described below:

```
"Banana"   ← top of the stack
"Apple"
"Cherry"
```

show what the stack will look like after the following sequence of operations.

```
S.push("Kiwi");
S.push("Kiwi");
S.pop();
String fruit = S.top();
S.push("Orange");
while (!S.isEmpty() && !S.top().equals(fruit)) S.pop();
```

"Kiwi"
"Banana"
"Apple"
"Cherry"

11) Assume **head** initially points to the following list of chars:

```
head -> [a] -> [b] -> [c] -> [d]
```

where the list nodes are instances of the following class

```
public class Node {
     public char value;
     public Node next;
}
```

After the code below executes, what is printed?

```
Node ptr1, ptr2, p;
ptr1 = head;
ptr2 = head.next;
ptr1.next = ptr2.next.next;
ptr2.next.next = ptr1;
p = head;
head = head.next;
while(p!=null){
     System.out.print(p.value + " ");
     p=p.next;
}
```

A D

12) What is the postfix expression of the infix expression
`3 / A + (B - 12) % C`?
(There is only one answer)

`3 A / B 12 - C % +`

13) What is the basic difference between the Reader/Writer and InputStream/OutputStream collections of classes?

Reader and Writer are base classes for character streams. InputStream and OutputStream are base classes for byte streams.

14) Check the most efficient list implementation (array or linked list) for the following problems? If both implementations are as efficient, check both of them.

	Array implementation	Linked list implementation
Random access of an item (e.g. accessing element i of the list for any i)	**X**	
Insertion/deletion of an item (assumes that the item has been located)		**X**
Binary search	**X**	
Simulation of a stack		**X**
Simulation of an unbounded queue*.		**X**

An unbounded queue is a list that has no limit on the number of items it contains. Queue operations are adding at the end of the queue, and removing from the front of the queue.

Programming questions

1) Consider an expression that contains grouping symbols. The grouping symbols are **( ), [ ]**. Expressions are considered to be balanced if their grouping symbols follow the following rules

   a) each left symbol must be followed by a right symbol of the same kind with some data in between (ie. you can't have an empty pair like [])

   b) if pairs are nested , one pair must be completely nested within another.

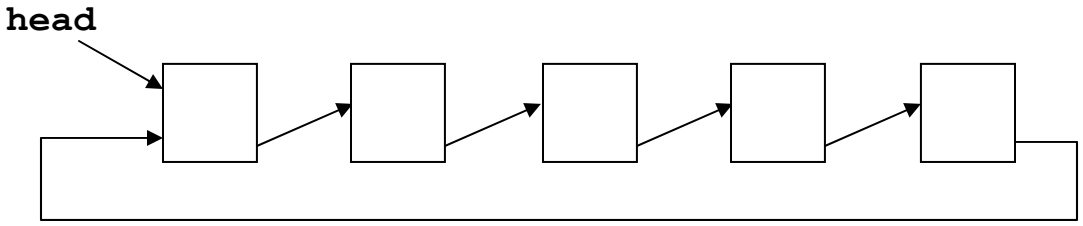
   For example, "[)", ")a(", and "([a)]" are not balanced.
   "(a)", "([a b])((c + d))", and "abc" are balanced.

   Write a method isBalanced that returns true if the given string is balanced, and false otherwise. Use a character stack in your implementation (assume that you have access to a CharacterStack class).

```
public boolean isBalanced(String s) {

     if (s == null) {
          throw new IllegalArgumentException("Null string");
     }

     CharacterStack stack = new CharacterStack ();

     // Your code goes here
         boolean hasData = false;

         for (int i = 0; i < s.length(); i ++) {
             char c = s.charAt(i);

             if (c == '(') {
                 stack.push(c);
                 hasData = false;
             }
             else if (c == '[') {
                 stack.push(c);
                 hasData = false;
             }
             else if (c == ')') {
                 if (stack.isEmpty()) return false;
                 if (stack.pop() != '(') return false;
                 if (!hasData) return false;
             }
             else if (c == ']') {
                 if (stack.isEmpty()) return false;
                 if (stack.pop() != '[') return false;
```

```
            if (!hasData) return false;
        }
        else {
            hasData = true;
        }
    }
    return hasData && stack.isEmpty();
}
```

1) In a circular linked list, the last node references the first node so that every node has a successor. Pictorially, a circular linked list looks like this:



Consider a class CircularLinkedList that uses the above data structure for a list. Implement an iterator within that class. You are given the structure on the next page. Just fill in the code. Do not use any helper methods from the java collections framework. Access the nodes directly.

Don't forget to throw a NoSuchElementException in next if necessary

```
public class CircularLinkedList implements List {

     private Node head;
     private int size;

     private class Node {
          Object item;
          Node next;
          Node(Object i) { item = i; }
     }

     // most of class not listed


     public Iterator iterator() {  return new Itr(this);  }

     /*********************************************
     *  ADD YOUR CODE TO THE CLASS ON THE NEXT PAGE
     *********************************************/
```

```
    private class Itr implements Iterator {

        // add your instance variable(s)
        Node p;
        boolean done;

        public Itr(CircularLinkedList list ) {
            p = list.head;
        }

        public boolean hasNext() {
            return !( p == null || done) ;
        }

        public Object next() {
            if (!hasNext()) {
                throw new NoSuchElementException();
            }
            else {
                Object item = p.item;
                p = p.next;
                done = (p == head);
                return item;
            }
        }

        public void remove() {  // don't  code this one
            throw new UnsupportedOperationException();
        }
    }
}
```