CMPSCI 187 (Fall 2018) Lab 05: Midterm Review

This lab reviews some topics on the midterm. To work on the assignment:

- Go to ***File -> Make a Copy*** to make an editable copy of this Google Doc for your account
- Follow the instructions to complete the assignment
- When you are done, go to ***File -> Download As -> PDF Document***
- Log in to Gradescope and submit your PDF

Section A: Multiple Choice

One correct answer per question. Select your choice by making that option **bold**

1. **What's the output of the following code?**

```
int x = 4, y = 5;
float z = x / y;
System.out.println(z);
```

(a) 1.25
(b) 1.0
(c) 0.8
(d) 0.0

2. **If class `Triangle` implements the `Geometry` interface, and class `RightTriangle` extends `Triangle`, which of the following would <u>not</u> compile?**

(a) `Geometry g = new Geometry();`
(b) `Geometry g = new Triangle();`
(c) `Geometry g = new RightTriangle();`
(d) `Triangle t = new RightTriangle();`

3. **Assume that `curr` is a variable that points to a node in the middle of a long linked list. Which of the following inserts a new node `nn` into the linked list <u>after</u> the `curr` node?**

(a) `nn.setLink(curr);  curr = nn;`
(b) `nn.setLink(curr);  curr.setLink(nn);`
(c) `nn.setLink(curr.getLink());  curr = nn;`
(d) `nn.setLink(curr.getLink());  curr.setLink(nn);`

4. **Which of these are in increasing order of complexity?**
(a) O(n) < O(log n) < O(n^2) < O(n^3) < O(2^n)
(b) O(n) < O(log n) < O(2^n) < O(n^2) < O(n^3)
(c) O(log n) < O(n) < O(2^n) < O(n^2) < O(n^3)
(d) O(log n) < O(n) < O(n^2) < O(n^3) < O(2^n)

5. **What is the Big-O cost of the following code (`n` is a large positive integer)?**

```
int count = 0;
for (int i = 1; i < n; i *= 2) {
      for (int k = i; k <= i + 8; k ++ ) {
            count ++;
      }
}
```

(a) O(1)
(b) O(log n)
(c) O(n)
(d) O(n log n)

(End of page 2)

Section B: Arrays

1. Complete the following method which reverses the order of elements stored in an `int` array `a`. For example, if `a[] = {1, 2, 3, 4, 5}`, calling `reverse(a)` will make it become `{5, 4, 3, 2, 1}`. If you declare any new variables, they must be of type `int`. You are <u>NOT</u> allowed to create any new method, any new array, or use any object type (such as Stack or ArrayList). The running time of your method should be no more than O(n).

<u>Remember</u>: during the real midterm, you must write down code correctly without the help of a Java compiler. Therefore when working on these programming questions, you should treat them as a real exam and should not rely on a Java compiler to tell you if your code is correct or not.

```
public void reverse(int[] a) {
	int n = a.length;
	for (int i = 0; i < n/2; i++)
	{
		int temp = a[i];
		a[i] = a[n-1-i];
		a[n-1-i] = temp;
	}
}
```

(End of page 3)

2. Complete the method below to perform a circular left-shift of elements stored in `a`**. For example, if** `a[] = {1, 2, 3, 4, 5}`**, calling** `circularLeftShift(a, 1)` **will circular left-shift the array once, making it** `{2, 3, 4, 5, 1}`**. In other words, the first element is shifted to the last position, and all other elements are shifted to the left by one position. Calling** `circularLeftShift(a, k)` **(where** `k > 0`**) is equivalent to repeat** `circularLeftShift(a, 1) k` **times. If you declare any new variables, they must be of type** `int`**. You are <u>NOT</u> allowed to create any new method, any new array, or use any object type (such as Stack or ArrayList). The running time of your method should be no more than O(k*n) (i.e. linear with respect to k times n).**

```
public void circularLeftShift(int[] a, int k) {
      int n = a.length;
      k = k % n;  // modulo n so that k is always less than n
      if(k <= 0) return;

      int temp = a[0];
      for (int i = 0; i < n-1; i++)
            a[i] = a[i+1];
      a[n-1] = temp;

      circularLeftShift(a, k-1);
}
```

(End of page 4)

Section 3: Linked Lists

```
public class LLStringNode {
      private String data;
      private LLStringNode next;
      public String getData()                   { return data; }
      public void setData(String data)          { this.data = data; }
      public LLStringNode getNext()             { return next; }
      public void setNext(LLStringNode next)    { this.next = next; }
}
```

Given the above definition of a Linked List node, complete the following LinkedListStrings class. Specifically, complete the `add` **and** `elementAt` **methods. Do NOT use iterators. Do NOT create new methods.**

```
public class LinkedListStrings {
      private LLStringNode head;

      // Add a new element to the beginning of the linked list. O(1).
      public void add(String element) {
            newNode = new LLStringNode();
            newNode.setData(element);
            newNode.setNext(head);
            head = newNode;
      }
```

(End of page 5)

```
// Return the k-th element on the list where k is the index
// (the first element has index 0 and so on). If the linked
// list has less than (k+1) elements, return null. O(n).
public String elementAt(int k) {
      LLStringNode current = head;
      for (int count = 0; count < k; count++)
      {
            if (current == null) return null;
            current = current.getNext();
      }
      return current.getData();
}
```

(End of page 6)