CS171 Midterm Practice Exam

Name:______________

You are to honor the Emory Honor Code. This is a closed-book and closed-notes exam. You have 75 minutes to complete this exam. Read each problem carefully, and review your answers. Good luck!

<table>
<thead>
<tr>
<th>Problem</th>
<th>Points</th>
<th>Earned</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>2.</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>3.</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>4.</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>5.</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>100</td>
<td></td>
</tr>
</tbody>
</table>

Below are some potentially helpful formulas.

\[ 1 + 2 + 3 + \ldots + (n - 1) + n = \frac{n \times (n + 1)}{2} \]

\[ 1 + x + x^2 + \ldots + x^{n-1} + x^n = \frac{x^{n+1} - 1}{x - 1} \]
Problem 1. (10 points) Give brief answers to the following questions.

1. What does a variable of object types store in Java?
   The reference to an object

2. How to check an object variable obj contains a null reference?
   if (obj == null)

3. Which are the 2 type of fundamental data structures used in computer programs?
   arrays and linked list

4. What is the main advantage of using linked list over arrays to store data?
   dynamic sizing, fast insert and delete

5. What is the main advantage of using arrays over linked list to store data?
   supports index based access
Problem 2. (20 points) Analyze the runtime of the following loops given an input size $N$ using direct loop analysis or recurrence relations. Derive the runtime cost function, and then a Big-O notation for the cost function. Assume that each loop statement takes 1 unit of time, and the update of loop variable can be ignored (does not count into running time).

1. for (int $i = 0; i < N; i++$) {
   if ($i \% 2 == 0$) {
      for (int $j = 0; j < N; j++$)
         System.out.println("1 iteration executed!");
   }
   else {
      for (int $j = 0; j < 10; j++$)
         System.out.println("1 iteration executed!");
   }
}

suppose $N$ is even
$i=0$, $j=0$, 1, 2, ..., $N-1$, #steps = $N$
$i=1$, $j=0$, 1, 2, ..., 9, #steps = 10
$i=2$, $j=0$, 1, 2, ..., $N-1$, #steps = $N$
$i=3$, $j=0$, 1, 2, ..., 9, #steps = 10
...
$i=N-1$, $j=0$, 1, 2, ..., 9, #steps = 10

Total #steps = $N \times N/2 + 10 \times N/2$

The runtime is $O(N^2)$.

2. for (int $i = 0; i < N; i++$) {
   for (int $j = i; j < 2 \times N; j++$) {
      System.out.println("1 iteration executed!");
   }
}

$i=0$, $j=0$, 1, ..., $2N-1$, #steps = $2N$
$i=1$, $j=1$, 2, ..., $2N-1$, #steps = $2N-1$
...
$i=N-2$, $j=N-2$, $N-1$, ..., $2N-1$, #steps = $N+2$
$i=N-1$, $j=N-1$, $N$, ..., $2N-1$, #steps = $N+1$

Total #steps = $(N+1) + (N+2) + \ldots + (2N-1) + 2N = N^2 + (1 + 2 + \ldots + N)$

$T(N) = 3N^2/2 + N/2 = O(N^2)$
Problem 3. (20 points) Solve the following recurrence relations and derive the Big O notation.

1. \( T(N) = T(N/3) + N/3 \)
   \( T(1) = 1 \)

   Suppose \( N = 3^k \)
   \[ T(N) = T(3^k) = T(3^{k-1}) + 3^{k-1} \]
   \[ = T(3^{k-2}) + 3^{k-2} + 3^{k-1} \]
   ...
   \[ = T(3^0) + 3^0 + ... + 3^{k-2} + 3^{k-1} \]
   \[ = T(1) + 1 + 3 + ...3^{k-2} + 3^{k-1} \]
   \[ = 1 + \frac{3^{k-1}}{3-1} \]
   \[ = 1 + \frac{N}{2} \]
   \[ = N/2 + 1/2 \]
   \[ = O(N) \]

2. \( T(N) = T(N-1) + 3N - 1 \)
   \( T(0) = 0 \)

   \[ T(N) = T(N - 1) + 3N - 1 \]
   \[ = T(N - 2) + 3(N - 1) - 1 + 3N - 1 \]
   \[ = T(N - 3) + 3(N - 2) - 1 + 3(N - 1) - 1 + 3N - 1 \]
   ...
   \[ = T(0) + 3 \ast 1 - 1 + ... + 3(N - 2) - 1 + 3(N - 1) - 1 + 3N - 1 \]
   \[ = 0 + 3 \ast (1 + ...N) - N \]
   \[ = 3 \ast (1 + N)N/2 - N \]
   \[ = 3N^2/2 + N/2 \]
   \[ = O(N^2) \]
Problem 4. (20 points) Below is an example Java method which uses a stack of integers with typical push and pop operations.

```java
public static int foo(int x, int y) {
    if (x <= 0 || y <= 0)
        return 0;
    stack.push(y);
    return foo(x - 1, y-1) + stack.pop();
}
```

1. Assuming the stack is initially empty, draw a snapshot of the stack after every push and pop statement for each recursive call for `foo(3,4)`. Label each stack snapshot with the recursive call, and the push or pop statement. For example, the following shows the first snapshot of the stack after the push statement for the call `foo(3,4).

```
4
foo(3,4) - push

3
4
---
foo(2,3) - push

2
3
4
---
foo(1,2) - push

3
4
---
foo(1,2) - pop

4
---
foo(2,3) - pop

---
foo(3,4) - pop
```

2. What does `foo(3,4)` evaluate to?

```
foo(3,4) = foo(2,3) + pop
foo(2,3) = foo(1,2) + pop
```
foo(1,2) = foo(0,1) + pop
foo(0,1) = 0
foo(1,2) = 0 + 2 = 2
foo(2,3) = 2 + 3 = 5
foo(3,4) = 5 + 4 = 9
Problem 5. (30 pts) A *doubly circular linked list* is a list where each list element points to its successor and its predecessor in a *circular manner*. The head variable points to the *first* element in the list. An *empty* list is represented by a *null* value in head. The prev value of the first list element points to the *last* element and the next value of the last element in the list points to the *first* element. The structure of a list element and an (incomplete) list class are given by the following class definitions.

```
public class ListElem
{
    public int value;
    ListElem next;  // "next" points to the successor
    ListElem prev;  // "prev" points to the predecessor
}

public class List
{
    public ListElem head;  // head must point to first element in list

    public List()  // Constructor
    {
        head = null;  // Empty list
    }

    //... other methods ...
}
```
Hint: be careful and take care of the special *empty list* case for all the following questions.

1. Write a Java method for the `List` class that *inserts* an `ListElem` object `x` at the *head* of the doubly circular list:

   ```java
   public void insertAtHead(ListElem x)
   ```

2. Write a Java method for the `List` class that *inserts* an `ListElem` object `x` at the *tail* of the doubly circular list:

   ```java
   public void insertAtTail(ListElem x)
   ```

3. Write a Java method for the `List` class that *deletes* the `ListElem` object at the *tail* of the doubly circular list:

   ```java
   public void deleteAtTail()
   ```