Test 2: CompSci 100e

Name (print): ________________________________

Community Standard acknowledgment (signature): ________________________________

<table>
<thead>
<tr>
<th>Problem</th>
<th>value</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>Community</td>
<td>pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 1</td>
<td>pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 2</td>
<td>pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 3</td>
<td>pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 4</td>
<td>pts.</td>
<td></td>
</tr>
<tr>
<td>TOTAL:</td>
<td>75 pts.</td>
<td></td>
</tr>
</tbody>
</table>

You may consult your eight (8) sheets, the Sedgewick and Wayne textbook, and no other resources. You may not use any computers, calculators, cell phones, or other human beings. You may refer to any program text supplied in lectures or assignments.

Do not discuss this test with anyone until the test is handed back.

The last page is blank and can be used as scratch paper. Make a note on the appropriate problem if you use the extra page. All pages including your two sheets of notes must be turned in.

In writing code you do not need to worry about specifying the proper import statements. Assume that all libraries and packages we’ve discussed are imported in any code you write.

```
public static class TreeNode {
    int info;
    TreeNode left;
    TreeNode right;
    TreeNode(int val, TreeNode lptr, TreeNode rptr) {
        info = val;
        left = lptr;
        right = rptr;
    }
}
```

<table>
<thead>
<tr>
<th>label</th>
<th>recurrence</th>
<th>solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>T(n) = T(n/2) + O(1)</td>
<td>O(log n)</td>
</tr>
<tr>
<td>B</td>
<td>T(n) = T(n/2) + O(n)</td>
<td>O(n)</td>
</tr>
<tr>
<td>C</td>
<td>T(n) = 2T(n/2) + O(1)</td>
<td>O(n)</td>
</tr>
<tr>
<td>D</td>
<td>T(n) = 2T(n/2) + O(n)</td>
<td>O(n log n)</td>
</tr>
<tr>
<td>E</td>
<td>T(n) = T(n-1) + O(1)</td>
<td>O(n)</td>
</tr>
<tr>
<td>F</td>
<td>T(n) = T(n-1) + O(n)</td>
<td>O(n^2)</td>
</tr>
<tr>
<td>G</td>
<td>T(n) = 2T(n-1) + O(1)</td>
<td>O(2^n)</td>
</tr>
</tbody>
</table>
PROBLEM 1: (Short Ones)

A. [4pts] The values 1, 2, . . . , n are added to an initially empty queue, and then n/2 values are removed from the queue. What is the big-O runtime of this sequence of queue operations where queues are implemented using doubly-linked lists with pointers to the first and list nodes; justify your answer briefly.

B. [4pts] Suppose the same sequence of queue operations are performed, but queues are implemented with singly-linked lists and pointers to the first and last nodes. What is the big-O runtime; justify your answer.

C. [4pts] Draw what root and next points to after executing the following code. Indicate if there is some kind of error.

```java
TreeNode root = new TreeNode(8, new TreeNode(3, new TreeNode(2, null, null),
    new TreeNode(7, null, null)),
    new TreeNode(10, null, null));
TreeNode next = root.right.right;
```
PROBLEM 2:  (Interval)
From Sedgewick: Consider the following data type, for intervals on the line:

```java
public class Interval implements Comparable<Interval>
{
    public int left;
    public int right;

    public Interval(int l, int r) {
        left = l; right = r;
    }

    public int compareTo(Interval b) {
        return left - b.left;
    }

    /**
     * Returns true if and only if this Interval overlaps with b
     */
    public boolean overlap(Interval b) {
        // TODO: complete in part A
    }
}
```

For a particular application, clusters of intervals are of importance. To find clusters, replace any pair of intervals that intersects (by even an endpoint) by the union of the two intervals, continuing until all intervals do not intersect. For example, the following set of intervals has 3 clusters:

```
11, 12, 22, 33, 44
```

Note that you are guaranteed to have Intervals with non-negative numbers. Given an array of intervals, how
many clusters are there? The brute-force algorithm that calculates the overlap of all entries is quadratic. It can also be done in \( O(n \log n) \) time with method \texttt{count} above.

A. Complete \textit{overlaps} above.

B. Complete \textit{countIntervals} above.
PROBLEM 3: (Molecular Dynamics (14 points))

Attached to the end of this test, you are given code to simulate the motion of $N$ colliding particles according to the laws of elastic collision using time-driven and event-driven simulations.

With time-driven simulation, time is discretized into slices of size $dt$. We update the position of each particle after every $dt$ units of time and check for overlaps.

With event-driven simulation, we focus on those times at which interesting events occur. In the hard disc model, all particles travel in straight line trajectories at constant speeds between collisions. Thus, our main challenge is to determine the ordered sequence of particle collisions. We address this challenge by maintaining a priority queue of future events, ordered by time. At any given time, the priority queue contains all future collisions that would occur, assuming each particle moves in a straight line trajectory forever. As particles collide and change direction, some of the events scheduled on the priority queue become "stale" or "invalidated", and no longer correspond to physical collisions. We adopt a lazy strategy, leaving such invalidated collisions on the priority queue, waiting to identify and discard them.

Each Event contains the time at which it will occur (assuming no supervening actions). There are 5 different kinds of Events

- **WallCollision**: abstract class representing a collision with a wall
  - **HorizontalCollision**: collision with a horizontal wall
  - **VerticalCollision**: collision with a vertical wall
- **BinaryCollision**: collision between two particles
- **RedrawEvent**: event where all particles are drawn in response to a collision or enough time passing

A. [5pts] How would the results of the simulation be different if you used a Stack rather than a PriorityQueue? Justify your answer briefly.

B. [5pts] If the collisions are not elastic, then some energy would be lost in a collision. What code would you need to change in order to have inelastic collisions?
C. [4pts] For each iteration through the `eventSimulate` loop in `CollisionSystem` simulation, where is the majority of time spent? That is, what line dominates the overall time?
PROBLEM 4:  \((\text{BST redux} (16 \text{ points}))\)

For this problem binary trees store integer values instead of strings and no duplicate values appear in the tree.

```java
public class IntNode{
    int info;
    IntNode left,right;
    public IntNode(int value){
        info = value;
    }
}
```

By definition a \textit{binary search tree} is a binary tree if each of the following properties hold for every node in the tree.

1. All values in the left subtree of a node are less than the node’s value.
2. All values in the right subtree of a node are greater than the node’s value
3. Both the left and right subtree of a node are binary search trees.

A. [2pts] Draw a binary tree that satisfies the first two properties, but not the third, so that it’s not a search tree. Nodes contain integer values.

B. [2pts] Draw a binary tree that satisfies the second two properties, but not the first, so that it’s not a search tree. Nodes contain integer values.
C. [6pts] The methods `getMin` and `getMax` below return the smallest and largest values in a tree, respectively. They are used in writing `isSearchTree` that returns a boolean value indicating whether a tree is a search tree using the definition of a search tree given above.

```java
public int getMax(IntNode root){
    if (root == null) return Integer.MIN_VALUE;
    return Math.max(root.info,
                    Math.max(getMax(root.left), getMax(root.right)));
}

public int getMin(IntNode root){
    if (root == null) return Integer.MAX_VALUE;
    return Math.min(root.info,
                    Math.min(getMin(root.left), getMin(root.right)));
}

public boolean isSearchTree(IntNode root){
    if (root == null) return true;
    return
        isSearchTree(root.left) &&
        isSearchTree(root.right) &&
        root.info > getMax(root.left) &&
        root.info < getMin(root.right);
}
```

What is the runtime of `isSearchTree` in both the average and the worst case for a tree of \( n \) nodes? You must provide two recurrence relations for `isSearchTree`: one for the average case in which trees are balanced, and one in the worst case in which trees are completely unbalanced. Your answer should include justifications for the runtime for other methods that are called (i.e. `getMin` and `getMax`).

(continued with Part D on next page)
D. [2pts] A student suggests different implementations of \texttt{getMax} and \texttt{getMin} because in their use in the code above they are only called when it is known that the tree is a search tree. The alternate implementations are below.

```java
public String getMax(TreeNode root){
    while (root.right != null){
        root = root.right;
    }
    return root.info;
}

public String getMin(TreeNode root){
    while (root.left != null){
        root = root.left;
    }
    return root.info;
}
```

Explain what the recurrences would be in the average and worst case for \texttt{isSearchTree} using these alternative implementations of \texttt{getMin} and \texttt{getMax}. You do not need to solve the recurrences, you just need to give them.

E. [4pts] Another student says that the alternative implementations can be used even when the body of \texttt{isSearchTree} is below, where the recursive calls are swapped with the calls to \texttt{getMin} and \texttt{getMax}. In a few words indicate whether you think this code will work correctly with the alternative implementations and why you think this.

```java
public boolean isSearchTree(IntNode root){
    if (root == null) return true;
    return
        root.info > getMax(root.left) &&
        root.info < getMin(root.right) &&
        isSearchTree(root.left) &&
        isSearchTree(root.right) &&
    }
```
**PROBLEM 5 : (Grid)**

Attached to the end of this test is the writeup and a working solution to the *GridGame* APT. There are between two and five lines missing from the solution – these lines complete the code using appropriate back-tracking.

Add the lines below:

```java
public int winCount(){
    int wins = 0;
    ArrayList<int[]> list = findLegalMoves();
    if (list.size() == 0) return 0;

    for(int[] a : list){
        myGrid[a[0]][a[1]] = 'X';

        // TODO: fill in code here
    }

    return wins;
}
```
(nothing on this page)