Name: ________________________________

Data Structures
Spring 2018
Midterm 2

This is a 50-minute, closed-notes exam. **Please put away all notes, phones, laptops, etc.** There are seven pages and four problems. Good luck!

Problem	**Score**
1	/25
2	/25
3	/30
4	/15
Total	/100

You get 5 points for writing your name at the top of this page.

You do not need to show any work beyond your final answer, but you are welcome to do so if you feel it will help me understand your thought process.

1. Hashing. Suppose you have a hash table of size $m = 7$ where the keys are 2-digit integers and the hash function is $h(k) = (2k_1 + k_2)\%m$, where k_1 is the first digit of the key and k_2 is the second digit of the key. For example, if the key is $k = 34$, then $k_1 = 3$, $k_2 = 4$, and $h(k) = (2 \cdot 3 + 4)\%7 = 3$. Draw the hash table that results when the following keys are added (in this order) to an initially empty table: 39, 56, 94, 84, 73, and using each of the following collision resolution techniques.

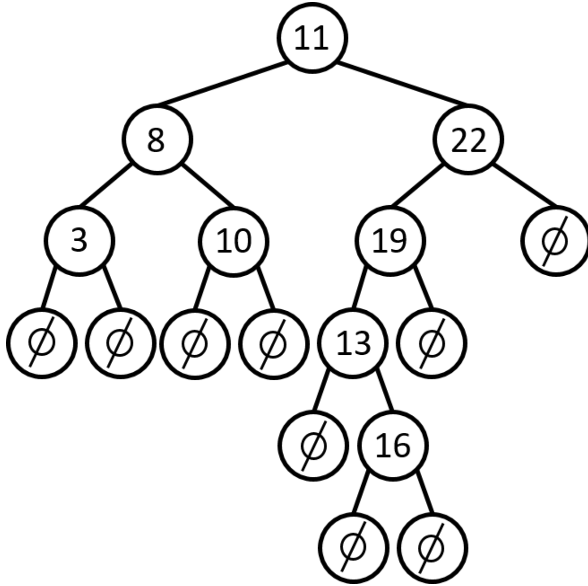
(a) Chaining

(b) Linear probing

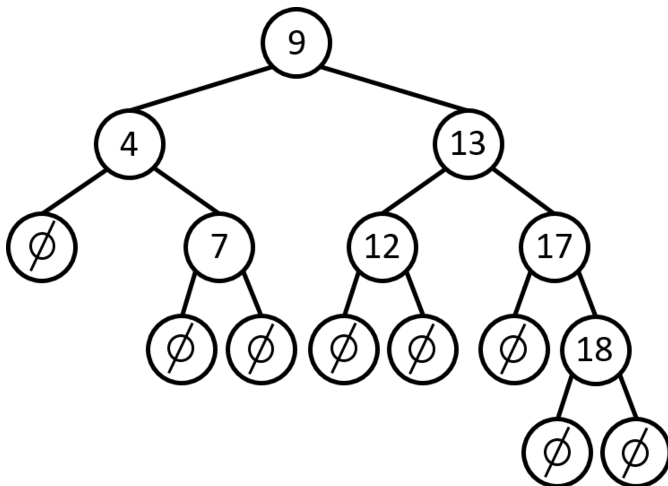
(c) Double hashing, where the second hash function is $h_2(k) = (k_1 + k_2 - 1)\%m$

2. Red-Black Trees.

(a) Can the following binary search tree be colored to make it a red-black tree? If so, shade in the black nodes to make this a valid red-black tree. If not, explain what is wrong with the tree structurally.



(b) Can the following binary search tree be colored to make it a red-black tree? If so, shade in the black nodes to make this a valid red-black tree. If not, explain what is wrong with the tree structurally.



(c) Draw the red-black tree that results after each insertion when inserting the following sequence of keys into an initially empty tree: 11, 5, 2, 9. Your solution should include four drawings of trees, one after inserting each element. Clearly indicate the color of each node at every step.

(d) Is it possible to have a red node with one internal child and one leaf child? If so, draw a red-black tree that includes this structure. If not, explain why not.

3. I've implemented my Binary Search Tree using two classes, as follows:

```
public class BST {
    Node root;

    public void add(String key) {
        ...
    }

    public boolean lookup(String key) {
        ...
    }

    public String remove(String key) {
        ...
    }

    ...
}
```

```
public class Node {

    String data;
    Node left;
    Node right;
    Node parent;

    ...

}
```

Where `...` indicates that I have (or may have) omitted some code (I don't want to give away too much of your homework assignment!).

Suppose we want to *join* two BSTs. Specifically, suppose we start with two BSTs, `T1` and `T2`, where we know that all of the keys in `T1` are smaller than all of the keys in `T2`. We also know that `T1` contains n nodes, `T2` contains m nodes, and $n < m$.

(a) Add a method called `join` to the `BST` class that, when given a `BST` as an input parameter, returns a new `BST` that is the result of joining `T1` and `T2`. That is, your method should have the header:

```
public BST join(BST T2)
```

For example, if `T1` contains the keys 1,3,6 and `T2` contains the keys 8,9,13,14, then the call `T1.join(T2)` should return a single tree containing keys 1,3,6,8,9,13,14.

Your goal should be to write a `join` method that is as efficient as possible. That is, it should have the smallest big-O runtime in n and m that you can come up with. You will get partial credit if your solution is correct, but does not have the lowest possible big-O runtime. You will also get partial credit if you give a clear explanation of what your method would do, but do not provide the actual code.

There is space on the next page for you to write your solution.

```
public BST join(BST T2) {

}
```

(b) In terms of n and m, what is the average-case big-O runtime of your `join` method?

4. Fill in the table of runtimes with the best possible big-O bound for each operation:

	Unbalanced BST		Red-black tree	Hash table using chaining*	
	Worst case	Avg. case	Worst case	Worst case	Avg. case
`add`					
`lookup`					
`remove`					

* You can assume that you are using a "good" hash function and that the size of the table is $O(n)$.