Quiz 4

A TreeSet stores each item in the Set in a binary tree. Below is a skeleton of a similar class, MyTreeSet. Some methods has been filled in. MyTreeSet performs some of the functions of TreeSet.

```java
public class MyTreeSet{
    private class TreeNode{
        public TreeNode leftChild;
        public TreeNode rightChild;
        public String data;
        public TreeNode(String str, TreeNode l, TreeNode r){
            data = str;
            leftChild = l;
            rightChild = r;
        }
    }
    private TreeNode root;
    public MyTreeSet(){
        root = null;
    }
    public void add(String str);
    private boolean helperContains(TreeNode tn, String str);
    public boolean contains(String str){
        helperContains(root, str);
    }
}
```

1) Implement `helperContains()` recursively such that a call to `contains()` correctly reports if the input String appears in the internal tree. Notice the code for `contains()` has already been filled in (4 points).

```java
private boolean helperContains(TreeNode tn, String str){
    if (tn == null)
        return false;
    if (str.equals(tn.data))
        return true;
    if (str.compareTo(tn.data) > 0)
        return helperContains(tn.rightChild, str);
    return helperContains(tn.leftChild, str);
}
```
Below is a class which uses a hash table, meant to store positive integers.

```java
public class MyHashTable{
    private int[] hashedArray;

    public MyHashTable(int n){
        hashedArray = new int[n];
        for (int i = 0; i < n; i++){
            hashedArray[i] = -1;
        }
    }

    private int hashOne(int n){
        return (3*n) % hashedArray.length;
    }

    private int hashTwo(int n){
        return (5*n) % hashedArray.length;
    }

    private boolean slotEmpty(int n){
        return (hashedArray[n] == -1);
    }

    public void doubleHash(int n);}
```

2) Implement `doubleHash()` using `hashOne()` and `hashTwo()` so `n` is placed in the hash table using double hashing. (5 points)

```java
public void doubleHash(int n){
    int one = hashOne(n);
    if (slotEmpty(one)){
        hashedArray[one] = n;
        return;
    }
    int two = hashTwo(n);
    while(slotEmpty((one + two) % hashedArray.length)){
        two *= hashTwo(n);
    }
    hashedArray[(one + two) % hashedArray.length] = n;
}
```

3) What problem occurs if `hashedArray` has length 5? (1 point)