CSCI 3323
Fall 2017
Exam 2

Last Name: _____

First Name: _____

Write answers on the provided answer sheets unless otherwise directed. You may refer to printouts of any source code you have written.

1. (2 pts each) What is the complexity of each of the following operations?
   (a) hash search (average case with assumptions we discussed in class)
   (b) red black tree delete + fixup
   (c) hash max
2. (4 pts) Why would someone consider using a binary search tree instead of using a hash-table? And vice-versa?
3. (4 pts) Compare and contrast chaining and open addressing.
4. (3 pts each) Briefly define each of the following terms:
   (a) primary clustering
   (b) double hashing
5. (6 pts) In class we claimed that `delete` in a hash table could be accomplished in $O(1+\alpha)$. Define α in this context and list all of the underlying assumptions required for this assertion to be true.
6. Assume the existence of the `MyStack` class defined below:

```
public class MyStack
{
  private int top;
  private int [] data;
  public MyStack() { top= 0; data= new int[1000]; }
  public boolean isEmpty() { return top==0; }
  public void push(int newElem) { data[top++]= newElem; }
  public void pop() { --top; }
  public int peek() { return data[top-1]; }  // return elem at top
}
```

   (6 pts) On this exam paper, show changes necessary to make the above class generic.
7. (5 pts) Suppose an initially empty hash table has 11 slots and that liner probing is used as the collision resolution strategy. Given the hash function below, show how the following keys would be arranged in the table if they are inserted in the order listed: 5, 12, 23, 1, 17.

```
int hash(int key) { return key%11; }
```

8. (4 pts) Draw a the red-black tree with 8 elements having the greatest possible height. Label each node with its key value and its color.

9. (2 pts) Illustrate a left-rotate operation.

10. (8 pts) Suppose a red-black tree is implemented using the following class to represent a single node in the tree.

```
class RBNode {
   int data;
   RBNode left, right, parent;
   char color;   // color='B' or color='R'
}
```

Write a method called `colorsOK` that accepts as a parameter the root node of a red-black tree. The method should examine the tree and return true if the tree does not contain any two adjacent red nodes. The method should return false otherwise.

Assume the existence of a global variable named `NIL` to serve as a sentinel in place of `null` pointers. You may also assume that the root node's parent pointer points to itself.

11. Consider the red-black tree given in figure 1.

    (a) (5 pts) Show the steps required to insert 300. Be sure to specify which cases you apply at each step and to redraw the tree after each case is applied. When drawing a red-black tree you can simply denote red nodes by putting the letter 'R' beside them. Other nodes will be considered to be black.

    (b) (5 pts) Show the steps required to insert 160. Be sure to specify which cases you apply at each step and to redraw the tree after each case is applied. Use the original tree as your starting point (not the result of problem 11a above).

12. Consider this frequency table showing how many times various letters appear in a document:

A	13	B	12
C	18	D	8
E	30	F	10
G	4	H	9

    (a) (4 pts) Construct an optimal Huffman tree for encoding a document from the table.

    (b) (2 pts) Using your tree, how would "FACE" be encoded?

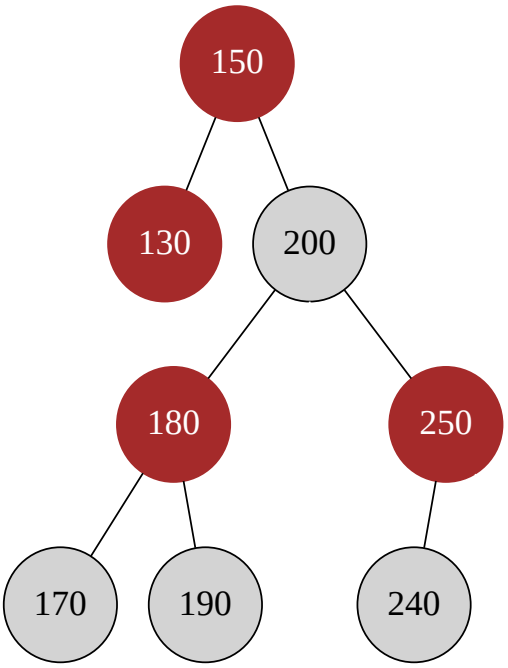

Figure 1: A Red-Black Tree

(c) (6 pts) Write a function called `showEncoding` that will accept a character and the root of a Huffman tree as parameters and will display the character encoding for the specified character in the tree assuming that a trip to the left is encoding as a 0 and a trip to the right is encoded as a 1. This is probably most easily written recursively so you may want a function that takes an additional parameter as a helper. Assume a the root is a `HuffmanNode` object as defined below:

```
class HuffmanNode {
   int         cost;
   char        letter;
HuffmanNode left,right;
```