CS 315 Sample Final Exam Questions

1. Build an expression tree from the following RPN expression:

   12 18 7 9 + - 5 % *

2. For the expression tree in question 1

   a. Which traversal algorithm would you use to evaluate the expression on the tree: preorder, inorder, postorder?

   b. Which data structure would you use to help with the evaluation?

3. Evaluate the tree from question 1.

4. Given the following array, named array, use the algorithm, buildBST, that comes after the array to help you build a balanced BST. Assume that the initial call to buildBST is

   buildBST(array, tree, 0, array.length-1)

   and tree is a pointer to an empty BST.

0	1	2	3	4	5	6
95	112	250	267	300	321	400

```
buildBST(array, tree, left, right)
{
   if (right < left) return;
   mid = (left + right) / 2;
   tree.add(array[mid]);
   buildBST(array, tree, left, mid-1);
   buildBST(array, tree, mid+1, right);
}
```

5. Use the tree formed in question 4 to do the following:

    a. A postorder traversal.

    b. Remove the root.

6. When deleting a node that has two children from a BST explain why it doesn't matter if you go right once and then all the way left or go left once and then all the way right. Use the BST from question 4 to demonstrate this.

The following description of an inner class that is used by a BST class to define a node is:

```
private class TreeNode<T extends Comparable<T>>
{
    T data;
    TreeNode<T> left, right;

    public TreeNode(T data)
    {
        this.data = data;
        left = right = null;
    }

    public int compareTo(TreeNode<T> obj)
    {
        if (obj == null) return 1;
        return data.compareTo(obj.data);
    }
}
```

Use this to help you with questions 7 and 8.

7. Write a recursive method called addNodesOnBST that returns the sum of the values of the nodes on a BST that has integers. The header for the method is:

```
int addNodesOnBST(TreeNode<Integer> root)
```

8. Write a recursive method called countPrimeNodesOnBST that returns the number of the nodes on a BST that have prime integers in the nodes. To help you determine if a value on a node is prime you can use a

method called isPrime that takes an integer as a parameter and returns true if the parameter is a prime number and false otherwise. The header for the countPrimeNodesOnBST method is:

```
int countPrimeNodesOnBST(TreeNode<Integer> root)
```

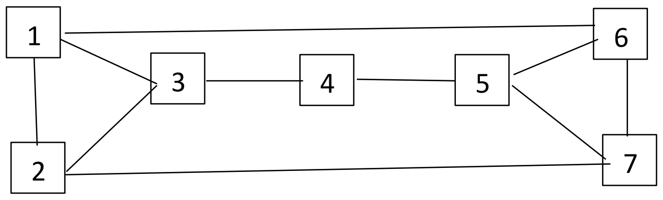
9. Using the values below show the steps to do the following:

   400   321   300   267   250   112   95

   a. Build a **min** heap graphically.
   b. Assign the values on the heap to an array so that the structure of the heap is maintained.
   c. Remove the top of the heap graphically.
   d. Using the original heap as represented in the array, show how you would have to manipulate the array if you removed the top of the heap.

10. For the following graph do:

    a. Create the adjacency list.
    b. Create the adjacency matrix.
    c. A breadth-first traversal starting at 4 and draw the resulting spanning tree. Select adjacent nodes in numerical order.
    d. A depth-first traversal starting at 4 and draw the resulting spanning tree. Select adjacent nodes in numerical order.



11. The class NonDirectedConnectedGraph2<T> has two variables named vertices and adjacencyLists defined as:

```
private ArrayList<T> vertices;
      private ArrayList<ArrayList<T>> adjacencyLists;
```

Explain how the following block of code, taken from the NonDirectedConnectedGraph2 class, uses those two variables.

```
public void addVertex(T vertex)
{
    if (vertex == null) return;
    if (vertices.contains(vertex)) return;
    vertices.add(vertex);
    adjacencyLists.add(new ArrayList());
}
```

12. Explain why each of the following statements is true:

   a. Accessing the element at the top of a heap is O(1).
   b. Creating a spanning tree of a connected graph from a given node is O(n).
   c. Creating all of the spanning trees from every vertex of a connected graph is O(n^2).
   d. Finding out if a vertex is connected to another vertex if the graph is represented by an adjacency matrix is O(1).