Some common recurrence relations

<table>
<thead>
<tr>
<th>T(n) = T(n/2) + O(1)</th>
<th>O(log n)</th>
</tr>
</thead>
<tbody>
<tr>
<td>T(n) = T(n/2) + O(n)</td>
<td>O(n)</td>
</tr>
<tr>
<td>T(n) = 2T(n/2) + O(1)</td>
<td>O(n)</td>
</tr>
<tr>
<td>T(n) = 2T(n/2) + O(n)</td>
<td>O(n log n)</td>
</tr>
<tr>
<td>T(n) = T(n-1) + O(1)</td>
<td>O(n)</td>
</tr>
<tr>
<td>T(n) = T(n-1) + O(n)</td>
<td>O(n²)</td>
</tr>
</tbody>
</table>

NOTE: You do not need to solve any recurrence relation. If you get a recurrence relation that is not equivalent to one above, just put the recurrence relation and not the big-Oh.

PROBLEM 1: (Stack/Queue (6 points))

Consider the following code.

```java
Stack<String> stk = new Stack<String>();
stk.push("La");
stk.push("So");
stk.push("Do");
stk.push("Me");
stk.pop();
stk.pop();
stk.push("Re");
System.out.println(stk.peek());

// a stack is printed from bottom (leftmost) to top (rightmost)
System.out.println(stk);

Queue<String> que = new LinkedList<String>();
que.add("La");
que.add("So");
que.add("Do");
que.add("Me");
que.remove();
que.remove();
que.add("Re");
System.out.println(que.peek());

// a queue is printed from the front (leftmost) to the back (rightmost)
System.out.println(que);
```

List the output from this code below here.
PROBLEM 2 : (Analyze (11 points))

Consider the following for the first three parts.
Suppose we want to create a data structure called NDS that has the following operations:

- find(int x) - returns true if x is in NDS
- findMax(int x) - returns the value of maximum element (does not remove it from NDS)
- findMedian() - returns the median (assume there are an odd number of items in NDS) (does not remove it from NDS).

Note: The median element is the middle element, if the elements were in sorted order. For example, the median of 3, 9, 11, 5, 7) is 7.

Consider the following implementations for NDS. Assume operations are implemented efficiently. Give the worst case big-Oh analysis of each operation.

A. (3 pts) The implementation is an array sorted in increasing order. Suppose n items are in NDS. What are the worst case times of the following operations?

- find(int x):
- findMax(int x):
- findMedian(int x):

B. (3 pts) The implementation is a doubly linked list in increasing sorted order with a pointer to the start of the linked list. Suppose n items are in NDS. What are the worst case times of the following operations?

- find(int x):
- findMax(int x):
- findMedian(int x):

C. (3 pts) The implementation is a min-heap. Suppose n items are in NDS. What are the worst case times of the following operations?

- find(int x):
- findMax(int x):
- findMedian(int x):
D. (2 pts) For this part consider a different data structure, an ArrayList of binary search trees, where the information field in a node is a point (X,Y).

The ArrayList is organized in the following way. Assume the ArrayList is of size M and each entry in the ArrayList is a binary search tree with N nodes that all have the same X values in their (X,Y) information. The ArrayList is in increasing sorted order by X values. For example, the binary search tree associated with the 0 slot in the ArrayList would all be points with the minimum X value.

To determine if a point (X, Y) is in the data structure, the ArrayList is first searched to find which slot in the array has that X value. If the X-value is valid, then its corresponding binary search tree is searched for the Y value. If the Y value is found, true is returned.

What is the AVERAGE case big-Oh runtime complexity in terms of N and M?

PROBLEM 3:  (Linking the lists (17 points))

Consider the following node for a linked list.

```java
public static class Node {
    int info;
    Node next;

    public Node( int data, Node link)
    {
        info = data;
        next = link;
    }
}
```

PART A. (5 pts) Write the method `doubleSmallOnes` that has two parameters, `list`, a reference to a linked list and an integer `value`. This method returns the linked list modified such that for each number in the linked list that is less than `value`, it has been doubled.

For example if the elements in the linked list are from left to right the first list below, then after the call `doubleSmallOnes(list, 5)` the second list below shows the modified list with all numbers that were less than 5 being doubled.

list → 4 → -2 → 8 → 9 → 3
list → 8 → -4 → 8 → 9 → 6

```java
public Node doubleSmallOnes(Node list, int value)
{
}
```

PART B. (6 pts) Write the method `addSumNodeAtEnd` that has one parameter, `list`, a reference to a linked list. This method returns the linked list modified with a new node at the end of the linked list that is the sum of all the nodes in the linked list.
For example if the elements in the linked list are from left to right the first list below, then
the second list is the list returned after the call addSumNodeAtEnd(list) applied to the first
list. The third list is a result of calling addSumNodeAtEnd(list) applied to the second list.
list → 4 → -2 → 8
list → 4 → -2 → 8 → 10
list → 4 → -2 → 8 → 10 → 20

public Node addSumNodeAtEnd (Node list)

PART C. Consider the method mystery shown below.

public Node mystery(Node list)
{
    if (list == null)
        return null;

    if (list.info < 0)
        return (mystery(list.next));
    else
        list.next = mystery(list.next);
    return list;
}

PART C1. (2 pts) If list is the linked list shown below, what is the result of the call
mystery(list)?
list → 4 → -2 → 8
PART C2. (2 pts) Explain in words what the method mystery does?
PART C3. (2 pts) What is the worst case big-Oh runtime complexity of mystery in terms
of n where n is the number of elements in the linked list? Justify your answer by giving a
recurrence relation. You do not need to solve the recurrence relation.

PROBLEM 4:  (How Many Different Trees (21 points))

These questions are on different types of trees.
PART A. (3 pts) BINARY SEARCH TREE: Insert the following elements into an empty
binary search tree in the order they appear.
Insert 21, 32, 10, 15, 18, 12 in this order into the empty tree.
PART B. (3 pts) BINARY SEARCH TREE: For the tree above list the first three values
in the order they would be printed using a preorder traversal.
PART C. (3 pts) BINARY SEARCH TREE: For the tree above list the first three values
in the order they would be printed using a postorder traversal.
PART D. (3 pts) HEAP: Insert the following elements, in this order, into an empty min-heap. Show each resulting min-heap and number them starting with 1. The numbers are: 21, 32, 10, 15, 18, 12

PART E. (3 pts) HEAP: Draw the resulting heap after deleting the min from the final heap you drew in Part D.

PART F. (3 pts) HUFFMAN TRIE: Consider the following table of character counts.

<table>
<thead>
<tr>
<th>b</th>
<th>d</th>
<th>e</th>
<th>i</th>
<th>k</th>
<th>l</th>
<th>s</th>
<th>u</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
</tbody>
</table>

Build the corresponding huffman trie using a priority queue of tree nodes. Show the resulting trie.

PART G. (3 pts) HUFFMAN TRIE: Consider the huffman trie built in part F. Using that trie, give the binary code for “duke”. (Remember right link means 1 and left link means 0).

PROBLEM 5: *Climbing trees (15 points)*

For this problem use the following TreeNode

```java
public static class TreeNode {
    int info;
    TreeNode left;
    TreeNode right;

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

For this problem, we will use a binary tree (this is not a binary search tree), in which each node has an integer field called *info*. Consider the following binary tree with the info values shown for each node.

PART A. (5 pts) Write the recursive method `weight` that has one TreeNode parameter called `root`. This method returns the sum of all the values in the tree whose root node is `root`.

For example, suppose root is the root node of the tree at the start of this problem. The call `weight(root)` would return 100, which is the sum of all the integer values in the tree. The call `weight(root.left)` would return 19, the sum of the integer values in the subtree that has 10 as its value.

Complete the method `weight` below. The method must be written recursively.

```java
// returns the sum of the integer values in the tree that starts at root
```
public int weight(TreeNode root)
{

PART B. (3 pts) Consider the method \textit{mysteryTree} that calls the method weight from Part A.

    public int mysteryTree(TreeNode root)
{
        if (root == null)
            return 0;

        int sumChildren = weight(root.left) + weight(root.right);  // weight is from Part A

        if (root.info > sumChildren)
            return 1 + mysteryTree(root.left) + mysteryTree(root.right);

        return mysteryTree(root.left) + mysteryTree(root.right);
    }

Explain what the method mysteryTree does.

PART C. (2 pts) What is the \textbf{average case} big-Oh runtime complexity of the method mysteryTree from Part B in terms of $n$ where $n$ is the number of elements in the tree? Justify your answer by giving a recurrence relation. You do not need to solve the recurrence relation.

PART D. (5 pts) Write the recursive method \textit{NumNoRC} that has one TreeNode parameter called \textit{root}. This method returns the number of nodes in the tree that have no right child.
For example, suppose root is the root node of the tree at the start of this problem. The call $\text{NumNoRC}(\text{root})$ would return $6$ because there are five leaf nodes, none of them have a right child, and one node labeled $2$ that has a left child and no right child.

Complete $\text{NumNoRC}$ below. The method must be written recursively.

```
// returns the number of nodes that do not have a right child.
public int NumNoRC(TreeNode root)
{

PROBLEM 6:  (Where is the treasure? (12 points))

The dungeon floors of castles have been mapped out indicating walls, open areas and treasure chests and this information is stored in a 2 dimensional character array. An 'X' represents a wall, a '.' represents an open area, and a 'C' represents a chest of treasure.

For example, a possible map of the dungeon floor of a castle is shown below (the blanks are not part of the map, they are just used for spacing to make the map easier to read). This map is 20 by 20 with every square represented by one of the three characters mentioned above. One cannot walk through walls, so this map shows that the dungeon floor is divided into six regions. One can get to any cell within a region as one can walk on spaces indicated by '.', but one cannot walk between regions (cannot walk through an X). This map would be stored in a 20 by 20 array with the top left corner 'X' at location [0][0] in the array.

```

| X X X X X X X X X X X X X X X X X X X X |
| X . . . . X . . . . X . . . . . . . X |
| X . . . . X . . . . X . . . . . . . X |
| X . . . . X . . . . . . . . . . . . . X |
| X . . . . X . . . . . . . . . . . . . X |
| X . . X X X X X X X X X X X X X X X X X |
| X . . . X . . . . . . . . . X . . . C X |
| X . . . X . . . . . . . . . X . . . . X |
| X . . . X . . . . . . . . . . . . . . X |
| X . . . X . . . . . . . . . . . . . . X |
| X X X X X X X X X X X X X X X X X X X X |
| X . . . . . . . X . . . X . . . . . . X |
| X . . . . . X . . . . . . . X . . . . X |
| X . . . . . X . . . . . . . X . . . . X |
| X . . . . . . . . . . . . . X . . . . X |
| X X X X X X X X X X X X X X X X X X X X |
| X . . . . . . . X . . . X . . . . . . X |
| X . . . . . X . . . . . . . C . . X |
| X . . . . . X . . . . . . . X . . . . X |
| X X X X X X X X X X X X X X X X X X X X |
```

Consider the following class that includes a 2D array castle and a method. Not all methods are shown.
public class treasureHunt {
    private char[][] castle; // assume castle is a square
    // not all methods shown.

    // returns true if position (xPos,yPos) has a clear path to the treasure
    // assume all castles are square in shape.
    public boolean accessChest(int xPos, int yPos) { // code not shown
    }
}

Given a particular position in the 2D array, one wants to know if there is a clear path to the chest of treasure. The example below shows a call to position `hunt.accessChest(1,8)` using the map on the previous page. In that case there is a clear path to the treasure that is located at position (2,15), so this call returns true.

```java
treasureHunt hunt = new treasureHunt();
// code not shown to initialize castle values
if (hunt.accessChest(1, 8))
{
    System.out.println("From position (1,8) can get to chest");
}
else
{
    System.out.println("From position (1,8) cannot get to chest");
}
```

For another example with the same map, the call `hunt.accessChest(1,1)` would return false as there is no chest in that region.

**PART A.** (1 pt) What value should be returned if the (xpos,ypos) position is not a valid position? (In the castle example, the call `hunt.accessChest(50, 89)`)

**PART B.** (1 pt) What value should be returned if the (xpos,ypos) position is a wall? (In the castle example, the call `hunt.accessChest(0, 3)` refers to a wall).

**PART C.** (2 pt) In using recursion to solve this problem, if the (xpos, ypos) position is in an area where there is no treasure, how does one make sure that the recursion does not continue forever? Explain.

**PART D.** (8 pts) Write the recursive method `accessChest` that is given a position (xpos, ypos) and returns true if there is a clear path to some chest. You can assume that the program stores a copy of the original map so that it is ok for you to modify the values in the castle array in this method in determining if there is a clear path to a chest. Remember the 2D array is defined on page 17.

```java
// returns true if position (xPos,yPos) has a clear path to the treasure chest.
// assume all castles are square in shape.
```
public boolean accessChest(int xPos, int yPos) {
    // First make sure the position is a valid position in the array