Some common recurrence relations

\[
\begin{align*}
T(n) &= T(n/2) + O(1) \quad O(\log n) \\
T(n) &= T(n/2) + O(n) \quad O(n) \\
T(n) &= 2T(n/2) + O(1) \quad O(n) \\
T(n) &= 2T(n/2) + O(n) \quad O(n \log n) \\
T(n) &= T(n-1) + O(1) \quad O(n) \\
T(n) &= T(n-1) + O(n) \quad O(n^2) \\
T(n) &= 2T(n-1) + O(1) \quad O(2^n)
\end{align*}
\]

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:  (Analyze (15 points))

Suppose we want to create a data structure called DataStore that has the following operations:

- insert(int x) - inserts x into DataStore
- delete(int x) - if x is in DataStore, deletes x from DataStore
- deleteMin() - removes and returns the minimum value in DataStore

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

A. (3 pts) The implementation is a singly linked list not in any order with a pointer to the start of the linked list. Suppose n items are in DataStore. What are the worst case times of the following operations?

- insert(int x):
- delete(int x):
- deleteMin():

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

- insert(int x):
- delete(int x):
C. (3 pts) The implementation is a min-heap. Suppose n items are in DataStore. What are
the worst case times of the following operations?

- insert(int x):
- delete(int x):
- deleteMin():

D. (3 pts) The implementation is an array sorted in decreasing order (with the largest
element in slot 0). Assume there is plenty of space for the array to grow. Suppose n items
are in DataStore. What are the worst case times of the following operations?

- insert(int x):
- delete(int x):
- deleteMin():

E. (3 pts) The implementation is a regular binary search tree, it does not balance itself.
Suppose n items are in DataStore. What are the worst case times of the following opera-
tions?

- insert(int x):
- delete(int x):
- deleteMin():

PROBLEM 2:  (Stacking the Deck (19 points))

Consider the following class for a new type of stack called a PriorityStack. This class will
have the usual stack operations of pop(), push() and peek(), but also have some additional
operations. The new operations are peekMin() which returns the minumum value in the
stack and popMin() which removes the minimum value from the stack. The implementation
of the stack will be with a singly linked list.
Shown below is part of the class with some of the methods not implemented. Not all the
methods are shown.
public class PriorityStack {

    private Node start; // points to the head of the linked list

    public static class Node {
        int value;
        Node next;

        Node(int val, Node ptr)
        {
            value = val;
            next = ptr;
        }
    }

    public PriorityStack()
    {
        start = null;
    }

    // pushes new value onto the stack
    public void push (int val)
    {
        start = new Node(val, start);
    }

    // returns top of stack, returns -1 if stack is empty
    public int pop ()
    {
        if (start == null)
        {
            return -1;
            int returnval = start.value;
            start = start.next;
            return returnval;
        }
    }

    // return the top of the stack, if the stack is empty return -1
    public int peek()
    {
        if (start == null)
        {
            return -1;
        }
        return start.value;
    }

    // return the minimum value from the stack

// return -1 if stack is empty
public int peekMin() { // code not shown
}

// return and remove the minimum value from the stack
// return -1 if stack is empty
public int popMin() { // code not shown
}

// returns the number of nodes in the stack
public int size() { // code not shown
}

// print the stack elements from left to right with the top element first
public void print() { // code not shown
}

PART A. (6 pts) Given the following code, show the result of the stack after this code executes. Note that print lists the elements in the stack from left to right on one line starting with the top element.

```java
PriorityStack pstack = new PriorityStack();
pstack.push(5);
pstack.push(8);
pstack.push(45);
pstack.push(7);
pstack.push(32);
pstack.push(11);
pstack.pop();
pstack.pop();
System.out.println("Top is " + pstack.peek());
pstack.push(20);
pstack.print();
System.out.println("min in stack is " + pstack.peekMin());
pstack.popMin();
pstack.print();
```

List the Output:

PART B. (4 pts) Write the method `size` that returns the number of elements in the stack.

```java
// returns the number of nodes in the stack
public int size() {
```
PART C. (6 pts) Write the method `popMin()` that removes and returns the minimum element in the stack. You may call `peekMin()` or any of the methods shown in the class. You do not need to write any methods in the class you call.

```java
// return and remove the minimum value from the stack
// return -1 if stack is empty
public int popMin() {
    // Implementation...
}
```

PART D. (3 pts) Consider the following `print2` method and recursive `rprint` method that we would like to add to the `PriorityStack` class.

```java
public void print2() {
    rprint(start);
    System.out.println();
}

public void rprint(Node s) {
    if (s != null) {
        rprint(s.next);
        System.out.print(s.value + " ");
    }
}
```

If the stack is from top to bottom: 8 4 9 7
show the result of calling `print2()`;

PROBLEM 3 :  (More Analysis (12 points))

PART A. (4 pts) The method `average` below correctly computes the average of the elements in array. Note that it calls `sum` and neither of these are recursive. What is the worst case big-Oh runtime complexity of `average` in terms of `n` where `n` is the number of elements in array?

```java
public double average(double[] array) {
    return sum(array, 0, array.length-1)/array.length;
}

public double sum(double[] array, int start, int end) {
    double total = 0.0;
    // Implementation...
}
```
for (int k=start; k <= end; k++)
    total += array[k];
return total;
}

PART B. (4 pts) The method average below computes the average of the elements in array by calling recAverage which is recursive. What is the worst case big-Oh runtime complexity in terms of n where n is the number of elements in array? Justify your answer by giving a recurrence relation. You do not need to solve the recurrence relation.

public double average(double[] array) {
    double lavg = recAverage(array, 0, array.length/2);
    int lnum = array.length/2 + 1;
    double ravg = recAverage(array, (array.length/2)+1, array.length-1);
    int rnum = array.length - 1 - array.length/2 ;
    return (lavg*lnum + ravg*rnum)/(lnum+rnum);
}

private double recAverage(double[] array, int start, int end) {
    if (start == end) // 1 element
        return array[start];
    if (start+1 == (end)) { // just 2 elements
        return (array[start]+array[end])/2.0;
    }

    int lnum = (end+start)/2-start +1;
    int rnum = end - (end+start)/2;
    return (recAverage(array, start, (end+start)/2) * lnum +
            recAverage(array, (end+start)/2 + 1, end)*rnum)/(lnum+rnum);
}

PART C. (4 pts) The method random below picks a random number from array by calling the recursive method recRandom. Assume generator is a random number generator that is O(1) time. What is the worst case big-Oh runtime complexity of random in terms of n where n is the number of elements in array? Justify your answer by giving a recurrence relation. You do not need to solve the recurrence relation.

public double random(double[] array) {
    return recRandom(array, 0, array.length-1);
}

private double recRandom(double[] array, int start, int end) {
    int num = generator.nextInt(2);
if (start == end) // 1 element
    return array[start];
if (start+1 == end)
{
    if (num == 0)
        return array[start];
    else
        return array[end];
}

double leftRand = recRandom(array, start, (end+start)/2);
double rightRand = recRandom(array, (end+start)/2+1, end);
if (num==0)
    return leftRand;
else
    return rightRand;

PROBLEM 4: (A Forest of Different Trees (19 points))

These questions are on different types of trees.

PART A. (3 pts) BINARY SEARCH TREE: Consider the binary search tree shown below. Insert 30, 70 and 10 in this order into the tree.

PART B. (3 pts) BINARY SEARCH TREE: For the tree above (don’t include the nodes you added), list the values using a postorder traversal.

PART C. (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: 10, 15, 8, 12, 20, 3

PART D. (3 pts) HEAP: Given an empty array, suppose n unique elements are inserted into an array in increasing order starting with slot 1 (slot 0 is ignored). Then the elements in the second half of the array are put into reverse order using the following code. Assume this code works correctly.
index = a.length - 1;
for (int k=a.length/2; k<(a.length/4)*3; k++)
{
    swap(a[k], a[index]);
    index--;
}

Is the resulting array with slot 1 the root (ignoring slot 0) a min-heap? Explain. **PART E.** (3 pts) HUFFMAN TRIE: Consider the following huffman trie (where right link means 1 and left link means 0).

```
  O
 / \
 i  O
 / \
 a s t
```

Using this trie, what character sequence is represented by the data

0110011101011101

**PART F.** (4 pts) HUFFMAN TRIE: Consider the huffman trie in PART E. Give frequencies for each character in the trie such that the exact same trie would be built using the priority queue algorithm we discussed in class.

```
a =  
e =  
i =  
s =  
t =  
m =  
r =  
```

**PROBLEM 5 : (Climbing the tree (16 points))**

For this problem use the following TreeNode

```
public static class TreeNode {
    int value;
    TreeNode left;
```
PART A. (4 pts) A node in a binary tree is **heavy** if it is larger than its sibling and also larger than all the nodes in the siblings subtree. Consider the following code.

```java
public boolean testNodeHeavy()
{
    return isNodeHeavy(root.right, root.left);
}

public boolean isNodeHeavy(TreeNode node, TreeNode sibling)
{
    if (sibling == null)
        return true;
    return ((node.value > sibling.value) &&
            isNodeHeavy(node, sibling.left) &&
            isNodeHeavy(node, sibling.right));
}
```

What is the **average case** big-Oh runtime complexity of the recursive method isNodeHeavy 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 B.** (6 pts) Write the recursive method rLeafCount that returns the number of leaves in a binary tree.

```java
// return the number of leaves in the tree
public int leafCount()
{
    return rLeafCount(root);
}

private int rLeafCount(TreeNode root) {
```
PART C. (6 pts) Write the recursive method printRecLevels that prints the value of every leaf in the tree on a separate line with its level number. Assume the root is at level 0. For example for the original tree in Problem 4 Part A, there are 4 leaves and the output would be:

8 level 2
25 level 3
87 level 3
94 level 3

```java
public void printLeafLevels()
{
    printRecLevels(root, 0);
}

private void printRecLevels(TreeNode node, int level) {
```