Problem 1:
Consider the following definition of a node for a tree in which each node can have an arbitrary number of children.

```
public class Q1Node<R> {
    final R payload;
    // Pointer to the head of a linked list of child nodes
    // This link goes down the tree
    // So if the current node is at depth N, then
    // firstChild is at depth N+1
    Q1Node<R> firstChild;
    // Pointer (in a single linked list style) to a sibling node
    // If the current node is at depth N then nextChild is also
    // at depth N
    Q1Node<R> nextChild;
    public Q1Node(R pload) {
        payload=pload;
        firstChild=null;
    }
    public void addChild(Q1Node n) {
        n.nextChild = firstChild;
        firstChild = n;
    }
}
```

Write a method that returns the total number of nodes in such a tree .

Problem 2:
Suppose you have an array-based binary heap with data already in an array as follows: (Array locations are in the first row, data items are in the second row:

| **Loc** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** | **15** | **16** | **17** | **18** | **19** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Value | 19 | 13 | 18 | 11 | 12 | 17 | 15 | 10 | 9 | 2 | 3 | 8 | 16 | 14 | 4 | 6 | 7 | 0 | 5 | 1 |

Show the ***array*** after each of the following heap operations:

poll()
poll()
insert(22)
remove(15)
remove(16)
poll()

Question 3: Sorting
Sometimes mergesort is faster than quicksort. Construct a list of length at least 10 on which mergesort is significantly faster than quicksort. Show the entire mergesort and quicksort process on your list and give a primitive operations count for each algorithm.

Question 4:
Consider the following method:

```
public int mystery(int[] arr) {
    return mystery(arr, 0, arr.length-1);
}
private int mystery(int arr[], int begin, int end)
{
    int c = arr.length * 0.5;
    if (begin < end) {
        int partitionIndex = partition(arr, begin, end);
        if (partitionIndex == c)
            return arr[c];
        if (partitionIndex > c)
            mystery(arr, begin, partitionIndex-1);
        else
            mystery(arr, partitionIndex+1, end);
    }
    return arr[c];
}
```

In the above, partition() is the partition method of quicksort. It does not appear here.

What does the mystery method return when given an array of integers? What is the worst case and expected case runtime of this method? Explain/justify each of your answers. (Examples will aid your explanations.)

Question 5:

```
public class LinkedBinaryTree<E extends Comparable<E>> {
    protected class Node
    {
        E payload;
        Node right;
        Node left;
        public Node(E e)
        {
            payload=e;
            right=null;
            left=null;
        }
        public String toString()
        {
            return payload.toString();
        }
    /** The root of the tree */
    protected Node root;

     // Other stuff as needed

    }
```

Given the tree and Node definition above, write a method returns the depth of the lowest node that has 2 children.

Question 6:
Fill in the following table with big-O time estimates. For “find min/max” give the time to find the maximum or minimum value in the data structure.

| | Unsorted array | Sorted array | Unsorted list | Sorted list | Complete Tree | Tree |
|---|---|---|---|---|---|---|
| search | | | | | | |
| insert | | | | | | |
| remove | | | | | | |
| find min/max | | | | | | |

In the table below give the expected worst case for a hashtable with the given load factor (α).

| | Hashtable load factor < 0.5 | Hashtable load factor > 0.9 |
|---|---|---|
| search | | |
| insert | | |
| remove | | |
| find min/max | | |

Question 7. Hashing
Suppose that there exists a method valueOf() that returns 1-10 for the characters A-J. Here is its signature:

int valueOf(char c)

So, valueOf('A') is 1, valueOf('B') is 2, etc.
Further suppose the polynomial accumulator function below for strings of length 3 or less that contain only A-J.

p(str) = valueOf(str.charAt(0)) + valueOf(str.charAt(1))*z + valueOf(str.charAt(2))*z*z

charAt(x) returns the characters at the given location in a string. So "ABC".charAt(1) return 'B'.
Finally suppose that the in polynomial accumulator z=3. Hence
p("ADB") = 1 + 4*3 + 2*3*3 = 31

Now, suppose that

h(x) = x%11
h2(x) = (x%7)+3

where h(x) is the hashing function and h2(x) is the function used for double hashing. x is the number calculated by the polynomial accumulator.

Finally, the method put(String key, Integer value) inserts the key-value pair into the hashtable.

You have 3 hashtables of size 11 that respectively use linear probing, quadratic probing and double hashing. Show the contents of each hashtable after completing the following operations (you need only show the final contents)

put("ABC",1)
put("ABD",2)
put("EFG",3)
put("EEE",4)
put("EFF",5)
put("GFG".6)
put("JFG",7)
put("EFF", 8)