NAME:  
ID:  

<table>
<thead>
<tr>
<th>Problem</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1- Trees (18pts)</td>
<td></td>
</tr>
<tr>
<td>2- Sorting (30pts)</td>
<td></td>
</tr>
<tr>
<td>3- Graphs (24pts)</td>
<td></td>
</tr>
<tr>
<td>4- Hash Table (8pts)</td>
<td></td>
</tr>
<tr>
<td>5- Linked-List (10pts)</td>
<td></td>
</tr>
<tr>
<td>6- Tree-problem (10pts)</td>
<td></td>
</tr>
<tr>
<td><strong>TOTAL (100pts)</strong></td>
<td></td>
</tr>
</tbody>
</table>
1- Trees[18pts]

[4pts]- Let us consider a 3 nodes binary tree- Represent all possible valid configurations:

[5pts]- We want to transform a list into a binary search tree (BST). Represent the final tree structure after inserting all the elements of array A (starting from A[0] to A[9]).
A=

| 5 | 3 | 8 | 0 | 6 | 2 | 9 | 1 | 4 | 7 |

[4pt]- Considering a BST structure. What is this method doing? Comments on its BigO running time?
```java
public Node method()
{
    Node current,last;
    current=root;
    while(current!=null)
    {
        last=current;
        current=current.right;
    }
    return last;
}
```

[5pts]- Write the recursive method postOrder that should traverse the nodes (and display the value of 'current.key'). What is the BigO running time of the method? Indicate also the result for array A (tree in question b). Hint: one application of the postOrder traversal is the postfix/RPN notation.
```java
public void postOrder(Node current)
{
    if (current!=null){

    }
}
```
2- Sorting[30pts]

[10pts=3+4+3]- Insertion sort. Complete the code. Apply it to the array of Integer A.

```java
public void insertionSort()
{
    int in, out;
    int temp;  // temp variable
    for(out=1; out<N; out++) // outer loop – select key
    {
        temp=array[out];  // save in memory select key item
        System.out.println("pivot="+temp);
        in=out;  // start shifting at out
        while(in>0 && array[in-1]>=temp)//To COMPLETE
        {

        }
    }
}
// array.display(); // display array at each iteration
} // end outer loop
} // end insertionSort()

Initial array A

| 5 | 3 | 8 | 0 | 6 | 2 | 9 | 1 | 4 | 7 |

pivot=  

pivot=  

pivot=  

pivot=  

pivot=  

pivot=  

pivot=  

pivot=  

pivot=  

pivot=  

pivot=  

pivot=  

pivot=  

pivot=  

pivot=  

What is the Big-O running time complexity of insertion sort? What is happening if the data are almost sorted to start with?
**[5pts]**- Quicksort (as seen in class, pivot at the right end). Apply it to array A.

**Initial array**

<p>| | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>3</td>
<td>8</td>
<td>0</td>
<td>6</td>
<td>2</td>
<td>9</td>
<td>1</td>
<td>4</td>
<td>7</td>
</tr>
</tbody>
</table>

pivot=

pivot=

pivot=

pivot=

pivot=

pivot=

<p>| | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
</tbody>
</table>

**[15pts=5+5+5]**- Heapsort

* we propose to first heapify the array A using successive insertion into the heap (one item at a time, first step is given)

<p>| | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<p>| | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<p>| | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<p>| | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<p>| | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<p>| | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<p>| | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<p>| | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
We now propose to heapify the array A using trickledown in place. You will provide two CBT representations: (i) the original array and (ii) the final heap array.

* heapsort in place (the first step can start with any one of the heaparray above)
[2pts] What kind of graph is this?

[4pts] Write its adjacency matrix (we consider for the ordering of nodes: ABCDEFGHI)

[6pts] For this question *only*, we suppose that all the weights are the same. Starting from Vertex E, give the order of the visited vertices using both the DFS and BFS algorithms. You will also provide the status of the Stack just before the first pop, and the status of the queue just before the first dequeue.
[6pts]- Find the Minimum Spanning Tree using the Prim's algorithm. Indicate a “step by step procedure” (Vertex visited, display of the priority Queue, dequeue action). You will start with Vertex E. Represent/Plot the final MST. Give the minimum total weight.

[6pts]- Find the shortest paths from Vertex A to any other vertices using the Dijkstra's algorithm. Fill up this array. What is the shortest path to go from vertex A to vertex I?

<table>
<thead>
<tr>
<th>From A to</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
<th>I</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
4- Hash Table[8pts]

class HashTable() {
    private LinkedList[] table = null;
    private N;

    public HashTable(int N) {
        this.N = N;
        table = new LinkedList[N];

        for (int i = 1; i < N; i++)
            table[i] = new LinkedList();
    }

    private int hash(int x) {
        return x % N;
    }

    private void insert(int x) { // To complete (one-line)
        // To complete (one-line)
    }
}

[4pts]- What type of hash-table this code is implementing ? How does it handle collisions?

[4pts]- Complete the insert method (on-line of code). We consider that the class LinkedList contains a method 'insert'.

5- Linked-List [10pts]

Suppose you have implemented the simple linked-list class (it is not double-ended or doubly). The 'data field' in the Link contains the integer current.data. The linked-list contains the methods insertFirst, delete, find, display., etc. Here is the example of the display method:

public void display() {
    Link current = first; //start probe at the beginning
    while (current != null) { // until the end of the list
        System.out.println(current.data);
        current = current.next; // move to next Link
    }
}
We propose to write two new methods that can return the middle element of the list. We note that we do not know the length of the list a priori.

[5pts] - the first method 'findMiddle1' will first need to calculate the length of the list (one traverse) and then traverse again the list but only half the length. Complete this method:

```java
public int findMiddle1() {
    //find middle element (two passes)
    // return the integer data value
}
```

[5pts] - Write the method 'find2Middle' that finds the middle element using only one traverse (on pass). Hint: You can use 2 probes.
6- Trees (again) – Problem [10pts]

Given a binary search tree, it is sometime desirable to visit the elements “level-by-level”. For example, if we wanted to print a BST so we could see its actual layout in memory, we would want to print each level of the tree from left to right and top to bottom (starting from the root). We call this technique ‘level-order traversal’.

[2pts]- Indicate what should be the result (visited nodes in level-order) for array A in question 1.b

[8pts]- Implement a level-order method to be added to the BST class. You may assume that the Queue class is available and holds elements of types Node (Hint: Think of what order elements from the tree must be added to the queue in order to visit each level completely before proceeding to the next level). The method is not recursive.

```java
public void levelOrder()
{
}
```

HAPPY HOLIDAYS