<table>
<thead>
<tr>
<th>Problem</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1- Binary Search Trees (30pts)</td>
<td></td>
</tr>
<tr>
<td>2- Heap and Heapsort (22pts)</td>
<td></td>
</tr>
<tr>
<td>3- Sorting (10pts)</td>
<td></td>
</tr>
<tr>
<td>4- Graphs (20pts)</td>
<td></td>
</tr>
<tr>
<td>5- Hash-Table (10pts)</td>
<td></td>
</tr>
<tr>
<td>6- Linked-List (8pts)</td>
<td></td>
</tr>
</tbody>
</table>

**TOTAL (100pts)**
1- Binary Search Tree [30pts]

We consider the following array:
A=

| 12 | 16 | 6 | 2 | 8 | 7 | 14 | 21 | 4 | 1 |

[4pts] Transform the 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]).

[2pts] What kind of particular tree is this?

We consider the following `method1` associated with the BST structure (we consider also that a Node includes the integer variable “key”)

```java
public int method1()
{
    if (root==null){return -1;}
    Node p1=root;
    Node p2=root;
    while (p1.right!=null){p1=p1.right;}
    while (p2.left!=null){p2=p2.left; }
    return (p1.key-p2.key);
}
```

[3pts] What does method1 do?

[2pts] What is the value returned if applied to the BST of array A above?

[2pts] Explain/Comment on its BigO running times in various situations?
We consider the following `method2` associated with the BST structure

```java
public void method2()
{
    if (root == null) {return;}

    Stack mystack = new Stack (100);
    Node current = root;

    while (current != null)
    {
        mystack.push(current);
        current = current.left;
    }

    while (!mystack.isEmpty())
    {
        current = (Node) mystack.pop();
        System.out.print(current.key + " ");

        if (current.right != null)
        {
            current = current.right;
            while (current != null)
            {
                mystack.push(current);
                current = current.left;
            }
        }
    }
}
```

[4pts] What does method2 do?

[1pts] What is the output of `method2` if applied to the BST of array A above?
[2pts] Explain/Comment on its BigO running time?

[2pts] Write a much shorter method that can do exactly the same thing without using a stack:

We consider the following `method3` and `recMethod3` associated with the BST structure

```java
public void method3() {
    recMethod3(root);
}

public void recMethod3(Node current) {
    Node temp = current.left;
    current.left = current.right;
    current.right = temp;

    if (current.left != null)
        recMethod3(current.left);

    if (current.right != null)
        recMethod3(current.right);
}
```

[4pts] What does `method3` do?

[2pts] Represent the new final tree structure after calling `method3` for the BST of array A above?
2- Heap and Heapsort [22pts]

[2pts]- Let us consider the set of numbers: 1, 8, 4. How many valid Heaps can you obtain (largest number has the highest priority)? Represent all possible valid configurations:

[2pts] Report the main differences between insertion/removal in priority queues if implemented either by a heap or directly by an ordered array?

Let us consider the following array:

A=

\[
\begin{array}{cccccccc}
12 & 16 & 6 & 2 & 8 & 7 & 14 & 21 & 4 & 1
\end{array}
\]

[4pts] Heapify the array A using successive insertion into the heap (one item at a time, first step is given)
[2pts] Provide the CBT representation of the final heaparray above.

[5pts] We now propose to heapify the array A using trickledown in place. What is the final CBT representations of the heap array.

[5pts] Heapsort in place (the first step can start with any of the heaparray above)

[1pt] What is the Big-O running time complexity of heapsort?

[1pt] What is the main difference between heapsort and mergesort?
3- Sorting [10pts]

[4pts] - Quicksort (as seen in class, pivot at the right end). Indicate the first 2 steps of the algorithm (2 partitioning steps only) applied to the array A.

Initial array

|   12 |   16 |   6 |   2 |   8 |   7 |  14 |  21 |   4 |   1 |

pivot=

pivot=

[3pts] - Quicksort - Given the partition function seen in class, complete the Java code shown below for the QuickSort function (3 instructions to complete).

```java
public static void quickSort(int[] values, int left, int right) {
    if (right-left <= 0) {
        ____________________________
    } else {
        int pivotIndex = partition(values, left, right);
        ____________________________
        ____________________________
    }
}
```

Let us suppose your prefer instead using ShellSort to sort the array A.

[2pts] What would be the resulting array just after an increment of 3 is used?

[1pt] What would be the resulting array just after a new increment of 2 is used?
4- Graphs[20pts]

We consider the following grid-graph, the graph in non-directed and weighted (with weights ranging from 1 to 4, represented by the thickness of the edges). The 12 Vertices are numbered from bottom to top and left to right (from A to L).

Here the list of edges with thickness 1: AB1,AD1,CF1,EF1,EH1,GH1,GJ1,JK1,KL1
Here the list of edges with thickness 2: BE2,HK2
Here the list of edges with thickness 3: BC3,DG3,HI3,IL3
Here the list of edges with thickness 4: DE4,FI4

[2pts] What is the total weight of the graph?

[4pts] Write its adjacency matrix (follow the alphabetical order with vertices sorted A to L)

[2pts] Starting from A, traverse the graph using BFS, and give the order of the visit (no need to consider the weights for this question). Requirement: Follow the alphabetical order.
[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.

<table>
<thead>
<tr>
<th>From</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>
<th>J</th>
<th>K</th>
<th>L</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>A1</td>
<td>∞</td>
<td>A1</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td></td>
<td>B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

[6pts]- From the complete weighted grid-graph. Find the ‘cheapest’ paths from Vertex A to any other vertices using the Dijkstra's algorithm. Continue to fill up this array. What is the cheapest path (with weight) to go from vertex A to vertex L?
Cite three main disadvantages of using hash-tables

A hash table of length 10 uses open addressing with hash function hash(k) = k%10, and linear probing. Represent the hash table below (values on the right), after we insert successively: 46, 34, 42, 23, 52, 33

[4pts] - Complete the method find below

```java
class HashTable{
    private DataItem table;
    private int size;

    public HashTable(int size){
        this.size=size;  table = new DataItem[size];
    }
    public int hash(int key){return    key%size; }

    public void insert(DataItem item)
    {
        int h = hash(item.getKey()); // key
        while (table[h]!=null && table[h].getKey()!=-1){ // until empty spot or -1
            h=(h+1)%size;//if occupied, increment by 1
        }
        table[h]=item; // enter item into table
    }

    public DataItem find(int key)
    {
    }
}
```
6- Linked-List – Problem [8pts]

We propose to use a sorted double-ended doubly linked list to store a series of integer. Each link contains a publicly visible integer called `value` to store its contents integer.

*Example with series=3,7,9,11*

![Diagram of doubly linked list](image)

The DoublyLinkedList class contains the standard methods `insertFirst`, `insertLast`, `deleteFirst`, `deleteLast`

**[4pts]** Write the constructor for the DoublyLinkedList class that accepts a sorted array “array” containing the list, and initializes the linked-list.

```
MERRY CHRISTMAS and HAPPY HOLIDAYS
```