<table>
<thead>
<tr>
<th>Problem</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1- General questions (14pts)</td>
<td></td>
</tr>
<tr>
<td>2- Big-O (18pts)</td>
<td></td>
</tr>
<tr>
<td>3- Sorting (25pts)</td>
<td></td>
</tr>
<tr>
<td>4- Queue (8pts)</td>
<td></td>
</tr>
<tr>
<td>5- Stack (8pts)</td>
<td></td>
</tr>
<tr>
<td>6- Recursion (10pts)</td>
<td></td>
</tr>
<tr>
<td>7- Linked-List (17pts)</td>
<td></td>
</tr>
<tr>
<td>TOTAL (100pts)</td>
<td></td>
</tr>
</tbody>
</table>
1- General questions [14pts] (brief answers)

a- If memory was not an issue, which sorting algorithms would be the best to use for addressing any situations (among the algorithms that we have seen so far)?

b- If we assume that the data are already sorted, what would be the big-O complexity of insertion sort and quicksort (using pivot on the right)?

c- What is the main advantage of using a double-ended feature for a linked-list (be precise)?

d- Cite two main advantages of using a doubly-linked list?

e- Fill up the following table using big-O notation for various insertion algorithms- Important, we consider only the number of comparisons and shift/copies for sorting a single item - (we note that enhanced insertion sort uses a binary search instead of a linear search)

<table>
<thead>
<tr>
<th></th>
<th>Comparisons</th>
<th>Shift/Copies</th>
</tr>
</thead>
<tbody>
<tr>
<td>Insertion sort using array</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Enhanced Insertion sort using array</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Insertion sort using Linked-list</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Enhanced Insertion sort using Linked list</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

f- How would you use a linked-list to sort an array?
2- Big-O [18pts]

a- Find the Big-O complexity of the following code (write it below each code)

```java
for (int i=1; i<N; i=i*2) {
    //do something
}
```

```java
for (int i=0; i<N*N; i++) {
    //do something
}
```

```java
for (int i=0; i<N; i++) {
    for (int j=0; j<i; j++) {
        for (int k=0; k<10; k++) {
            //do something
        }
    }
}
```

```java
for (int i=N; i>0; i--) {
    //do something
}
```

```java
for (int i=N; i>0; i=i/2) {
    for (int j=1; j<N; j=j*2) {
        //do something
    }
}
```

b- Analysis

```java
public void surprise() {
    Random rnd = new Random();
    int out, index;
    double temp;
    for (out=N-1; out>0; out--)
        {
            index = rnd.nextInt(out+1);
            temp = array[index];
            array[index] = array[out];
            array[out] = temp;
        }
}
```

What is the big-O of method `surprise`?

What does it do?

What would happen if we replace `index=rnd.nextInt(out+1);` by `index=out;`
3- Sorting[25pts]

[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 |

How many comparisons are used in sorting this sample array A?

What is the Big-O running time complexity of insertion sort?
Let us suppose your prefer instead using ShellSort to sort the array $A$ with an increment of 3 to start with. What would be the resulting array just after an increment of 3 is used?

What can you say about this set of data compared to the original one?

If you now decide to use ShellSort with increment of 1 (i.e. insertion sort) on this new array, how many comparisons would you get?

What is the Big-O running time complexity of Shellsort (in average) ? Comment also on the optimal big-O for insertion sort.

Quicksort (as seen in class, pivot at the right end). Apply it to array $A$.

Initial array

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

pivot=

pivot=

pivot=

pivot=

pivot=

pivot=

pivot=

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
4- Queue [8pts]
We consider first a priority queue implemented using an array. We know that removing an item from the front of queue (dequeue) is O(1) while inserting a new item is O(N) (enqueue). Why enqueue is so slow?

Can you think on a way to improve a bit the insertion time (which is still O(N))?

We now consider the following code:

```java
class QueueApp {
    public static void main(String[] args) {
        Queue[2] aqueue = new Queue[2];
        for (int i=0;i<aqueue.length;i++)
            aqueue[i]=new Queue(10);
        for (int i=1;i<=10;i++)
            if (i%2==0)
                aqueue[0].enqueue(i);
            else
                aqueue[1].enqueue(i);
        for (int i=1;i<=10;i++)
            if (!aqueue[0].isEmpty())
                System.out.println(aqueue[0].dequeue());
            else
                System.out.println(aqueue[1].dequeue());
    }
}
```

What is the output of this code?
5- Stack [8pts]

We consider the following code that is making use of the class Stack.java (containing the methods: pop, push, size and isEmpty,).

class StackCalcApp
{    
    public static void main(String[] args)
    {
        StackCalc mystack = new StackCalc(100); //100 max capacity
        EasyIn easy = new EasyIn();
        int n=easy.readInt();
        for (int i=n;i>0;i--) mystack.push(i);
        boolean opadd=true;
        while (mystack.size()>1)
        {
                if (opadd) {
                        mystack.push((int) mystack.pop() + (int) mystack.pop());
                        opadd=false;
                }
                else       {
                        mystack.push((int) mystack.pop() * (int) mystack.pop());
                        opadd=true;
                }
        }
        System.out.println("Result is "+mystack.peek());
    }
} 

What does this code do (be precise)?

What would be the result after each run if the user choose successively n=1,2,3,4,5, or 6?
We consider the Tribonacci series: 0, 1, 1, 2, 4, 7, 13, 24, 44, 81, 149, ...
And the recurrence relation (n>2): \( T_n = T_{n-1} + T_{n-2} + T_{n-3} \) with \( T_0 = 1; T_1 = 1; T_2 = 2 \)

Complete the recursive trib function below

```c
int trib(int n)
{
    //trib function implementation
}
```

For the case of \( n=5 \), plot the (divide and conquer) structure of the recurrence including the various calls to trib and the intermediate summation. As a reminder here is what we obtained for Fibonacci.
We propose to use a doubly double-ended linked list to store a series of natural positive (integer) number: 1, 2, 3, 4, …, n
The link class contains only one item integer named **value. Example for n=4**

![Linked-List Diagram]

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

**a-** Write the constructor for the DoublyLinkedList class that accepts “n,” the length of the list, as an argument and initializes the linked-list.

```java
public DoublyLinkedList(int n){
   // Constructor implementation
}
```

**b-** We propose to implement a method that will compute the sum (1+2+3+…+n) and return the result. Requirement: the list must be traversed forward.

```java
public int sumForward()
{
   // Method implementation
}
```
c- We propose to implement a method that will compute the product \((1*2*3*...*n)\) which is \(\text{fact}(n)\) and return the result. Requirement: the list must be traversed backward.

```java
public int factBackward()
{
}
```

d- We propose to implement the method “factBackward” above using a recursion. The new method `factBackward` will call a recursive method named `recfactBackward`. Write the two methods below.