Homework Lesson 1- LU Practical Algorithms and Data Structures

Document Prepared by **Dr. Margarita Diaz Cortes**

1. Analyze the following codes by estimating their complexities (best and worst scenarios)

In [ ]:

```
def func_1(items):
    result = items[3] * items[3]
    return result

func_1([1,2,3,4])
```

In [ ]:

```
def func_2(items):
    for item in items:
        print(item)

func_2([2,5,6,7,8])
```

In [ ]:

```
def func_3(items):
    for item in items:
        print(item)

    for item in items:
        print(item+1)

func_3([1,2,3,4])
```

In [ ]:

```
def func_4(items):
    for item in items:
        for item2 in items:
            print(item, ' ' ,item)

func_4([4, 5, 6, 8])
```

In [ ]:

```
def mergeSort(arr):
    if len(arr) > 1:

         # Finding the mid of the array
        mid = len(arr)//2

        # Dividing the array elements
        L = arr[:mid]

        # into 2 halves
        R = arr[mid:]

        # Sorting the first half
        mergeSort(L)

        # Sorting the second half
        mergeSort(R)

        i = j = k = 0

        # Copy data to temp arrays L[] and R[]
        while i < len(L) and j < len(R):
            if L[i] < R[j]:
                arr[k] = L[i]
                i += 1
            else:
                arr[k] = R[j]
                j += 1
            k += 1

        # Checking if any element was left
        while i < len(L):
            arr[k] = L[i]
            i += 1
            k += 1

        while j < len(R):
            arr[k] = R[j]
            j += 1
            k += 1

arr = [12, 11, 13, 5, 6, 7]

mergeSort(arr)
```

In [ ]:

```
def heap_permutation(data, n):
    if n == 1:
        print(data)
        return

    for i in range(n):
        heap_permutation(data, n - 1)
        if n % 2 == 0:
            data[i], data[n-1] = data[n-1], data[i]
        else:
            data[0], data[n-1] = data[n-1], data[0]

data = [1, 2, 3]
heap_permutation(data, len(data))
```

2. Propose an algorithm, that finds the element different from 1 (number 9) in the following matrix:

$$X = \begin{bmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 9 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1
\end{bmatrix}$$

Estimate the complexity of your algorithm (Big O, Big Omega and Big Theta)

3. Propose an algorithm, that finds the template

$$A = \begin{bmatrix} 2 & 3 \\ 5 & 6 \end{bmatrix}$$

in the following matrix:

$$X = \begin{bmatrix}
1 & 5 & 7 & 9 & 6 & 2 & 1 & 7 & 7 & 9 & 3 & 4 & 2 & 3 & 5 & 7 & 9 & 4 \\
1 & 5 & 7 & 9 & 6 & 2 & 1 & 7 & 7 & 9 & 3 & 4 & 2 & 3 & 5 & 7 & 9 & 4 \\
1 & 5 & 7 & 9 & 6 & 2 & 1 & 7 & 7 & 9 & 3 & 4 & 2 & 3 & 5 & 7 & 9 & 4 \\
1 & 5 & 7 & 9 & 6 & 2 & 1 & 7 & 7 & 9 & 3 & 4 & 2 & 3 & 5 & 7 & 9 & 4 \\
1 & 5 & 7 & 9 & 6 & 2 & 1 & 7 & 7 & 9 & 3 & 4 & 2 & 3 & 5 & 7 & 9 & 4 \\
1 & 5 & 7 & 9 & 6 & 2 & 1 & 7 & 7 & 9 & 3 & 4 & 2 & 3 & 5 & 7 & 9 & 4 \\
1 & 5 & 7 & 9 & 6 & 2 & 1 & 7 & 7 & 9 & 3 & 4 & 2 & 3 & 5 & 7 & 9 & 4 \\
1 & 5 & 7 & 9 & 6 & 2 & 1 & 7 & 7 & 9 & 3 & 4 & 2 & 3 & 5 & 7 & 9 & 4 \\
1 & 5 & 7 & 9 & 6 & 2 & 1 & 7 & 7 & 9 & 3 & 4 & 2 & 3 & 5 & 7 & 9 & 4 \\
1 & 5 & 7 & 9 & 6 & 2 & 1 & 7 & 7 & 9 & 3 & 4 & 2 & 3 & 5 & 7 & 9 & 4 \\
1 & 5 & 7 & 9 & 6 & 2 & 1 & 2 & 3 & 9 & 3 & 4 & 2 & 3 & 5 & 7 & 9 & 4 \\
1 & 5 & 7 & 9 & 6 & 2 & 4 & 5 & 6 & 9 & 3 & 4 & 2 & 3 & 5 & 7 & 9 & 4 \\
1 & 5 & 7 & 9 & 6 & 2 & 7 & 8 & 9 & 9 & 3 & 4 & 2 & 3 & 5 & 7 & 9 & 4
\end{bmatrix}$$

Estimate the complexity of your algorithm (Big O, Big Omega and Big Theta)

In [ ]: