ITI1120 – Lab8
Analysis of algorithms in Searching and Sorting processes

Objectives

The big O notation
- Running Time = Number of Operations

Searching an element
- in a list (Linear Search)
- in an array
- in a sorted list (Binary Search)

Count the number of an element occurrences
- in a list
- in an array

Sorting algorithms
- Simple algorithm (Bubble Sort)
- Sorting by insertion (Insertion Sort) (modifies the initial list)

O(n) Exercises

```
def foo1(n):
  i=1
  while i<n:
    i=i*2
    count = count + 1

def foo2(n):
  i = n
  while i>=1:
     i =i//2

def foo3(n):
  for i in range(n):
     print("*")

def foo4(n):
  for i in range(-n,n,3):
     print("*")
```

```
def foo5(n):
    for i in range(n):
        for j in range(i+1,n):
            Print("*")

def foo6(n):
    for i in range(n):
        for j in range(n):
            for k in range(n):
                Print("*")

def foo7(n):
    for i in range(n):
        j = n
        while j>=1:
            j = j//2
```

Generating very large lists

```
import random

n = 100  # nb of elements, try with 1000, 100000

# list in ascending order from 1 to n
list1 = [ v for v in range(1, n+1) ]
print(list1)

# list in descending order from n to 1
list2 = [ v for v in range(n, 0, -1) ]
print(list2)

# list with random elements
list3 = []
for index in range(n):
  v = random.randrange(1,n+1) # between 0 and n
  list3.append(v)
print(list3)
```

Searching for an element in a list (linear search)

- Create a function that takes a list and an integer v, and returns True if v is in the list (False otherwise). The function should be efficient and stop searching as soon as possible.
- The main program must generate a very large list with random elements, call the function to search a value and display the result.

  Add in the function a variable Nsteps to count the number of steps used by the algorithm (number of times the loop is executed) and display a message with this information.

Example

```
>>> l3 = [7, 23, 86, 71, 17, 5, 63, 91, 69, 92, 76, 49, 22,
45, 39, 52, 84, 51, 72, 28, 81, 52, 7, 50, 96, 64, 18, 71,
44, 46, 96, 89, 31, 9, 31, 83, 80, 20, 17, 12, 97, 51, 90,
53, 25, 24, 6, 86, 41, 2, 69, 26, 63, 9, 32, 91, 13, 33, 32,
96, 63, 10, 63, 63, 79, 96, 82, 9, 70, 16, 84, 100, 55, 86,
22, 8, 2, 90, 91, 99, 15, 38, 30, 7, 44, 8, 1, 8, 81, 86, 44,
44, 64, 2, 78, 96, 28, 80, 42, 44]
>>> find(l3, 78)
Number of steps: 95
True
```

Searching an element in an array

- Create a function that takes an array and an integer v, and returns True if v is in the array or 2D list, False otherwise.
- The function should be efficient and stop searching the 2 loops as soon as possible. the function should add a variable Nsteps to count the number of steps used by the algorithm when the loops execute and then display that information.
- The main program takes an array, call the function, and display the result.

Example:

```
>>> M = [[1,2,10],[7,5,6], [8,8,9,10]]
>>> find_m(M,5)
Number of steps 5
True
```

Count the number of an element occurrences in a list

- Create a function that takes an array and an integer v, and returns the number of v occurrences in the array. Add a variable Nsteps to count how many times the loops have executed and display that information in a message.
- The main program must take an array, call the function, and display the result.

```
>>> l3 = [52, 14, 14, 8, 85, 69, 1, 77, 94, 96, 51, 65, 35, 32, 87, 92, 74,
47, 27, 88, 11, 11, 26, 14, 100, 37, 62, 3, 63, 5, 20, 53, 28, 10, 43, 16, 94,
6, 82, 49, 74, 55, 89, 97, 12, 38, 72, 94, 3, 77, 42, 26, 25, 16, 89, 10, 8,
63, 93, 77, 68, 56, 74, 46, 54, 50, 80, 33, 69, 95, 2, 79, 73, 6, 31, 41,
38, 81, 88, 12, 39, 77, 49, 30, 18, 22, 40, 40, 12, 51, 69, 32, 76, 77, 90,
60, 41, 12, 30, 65]

>>> count(l3,6)
Nomber of steps 100
```

Binary search in a sorted list

- If the list is already sorted in an ascending order, can you find an efficient algorithm to look for the value v, without having to go through the whole list?
- Add a variable to count the number of steps executed by your function.

```
>>> binary_search([1, 2, 3, 4, 4, 5, 7, 9, 10,
  13], 10)
Number of steps 3
True
>>> binary_search([1, 2, 3, 4, 4, 5, 7, 9, 10,
  13], 6)
Number of steps 4
False
```

Sorted a list

- Think of a simple algorithm to sort a list in ascending order.
- Implement it in a Python function and add a counter that counts the number of steps taken.
- The function must modify the initial list.
- Is your function efficient? How many steps have been executed?

- Note: There is a predefined function sort(). What algorithm does it use?
- If the list's name is l1, we can sort it with:

```
>>> l1.sort()
```

Sort by insertion

- An efficient algorithm can be done by insertion (Insertion Sort): Insert a value in the good position in the list already sorted on the left. The first element is in the good position. Check the second element, and insert the left part of the list before him (thus the left part of the list is sorted). Repete with the other elements, until the list is fully sorted.
- Modify the initial list. Calculate the number of steps.

Example

```
>>> L = [3, 4, -1, 7, 2, 5, 16, -2, 17, 7, 82, -1]
>>> sort_by_insertion(L)
Number of steps 38
>>> L
[-2, -1, -1, 2, 3, 4, 5, 7, 7, 16, 17, 82]
```