DESIGN AND ANALYSIS OF ALGORITHMS
LAB 04
Brute-force Algorithms

1. Definition:
   Following Brute-force approach means we try to solve problems without worrying about the cost (running time, space)
2. Properties:
   This type of algorithms is easy to understand and easy to implement as it is close to the design's natural thinking. Because when studying certain problems, we will see their constraints "accidentally" lead to naive, brute-force solution.
3. Some examples of the brute-force algorithm

Problem 1: The factorial of A positive integer is defined as follows:

$$n! = \begin{cases} 1 \text{ nếu } n = 1 \\ n * (n-1) * \ldots * 1 \text{ nếu } n > 1 \end{cases}$$

a. Write the code that assumes the factorial algorithm with the parameter.n
b. Determine the fundamental operation of the algorithm that has been written and calculated complexity calculated by.n

The solution to the problem 1

```
ALGORITHM: calculate_Factorial(n)
#The algorithm determines the factorial of a given number $n$
#Input A given number$n > 0, n \in N$
#Output The factorial of $n$
    1. result = 1
    2. For I←1 To N By
    3.         $result = result * i$
    4. end for
    5. Return Result
    6. End calculate_Factorial
```

The basic operation is in lines (**<u>focus on the most important (costly), most frequently executed operations: assignment (“=”) comparison(“==, >, <, >=, <=, !=”) and arithmetic operations</u>**)

1. Assignment in line 3

The total number of basic operations is:

$$M(n) = n \in \Theta(n)$$

So the complexity of the algorithm is $\Theta(n)$

4. Exercises

1. Exponential power of a positive integer $a^n (a, n \geq 0)$ is defined as follows:

$$a^n = \begin{cases} 1 \, if \;\; n = 0 \\ a * a * \ldots * a \, if \; n > 0 \end{cases}$$

   a. Write pseudocode to calculate the exponential power, where n is the input size.
   b. Determine the basic operations of the algorithm that has been written and calculate complexity as a function of n.
   c. Implement the algorithm in Python programming language.

2. Design algorithms, evaluate their complexity, then implement them in Python for the following problems:
   a. Looking for a value K in the list of given integers A
   b. Sorting the elements of the list in non-descending order.
   c. Multiplying two square matrices with size n
   d. Calculating the combination determined by the following formula:

$$C_n^k = \frac{n!}{k!\,(n-k)!}$$

(example) $C_6^2 = \frac{6!}{2!(6-2)!} = \frac{6!}{2!4!} = 15$

3. Determine the computational complexity of the following algorithm:

```
ALGORITHMS Secret_2 ($A[0..n-1]$)
1. For $i \leftarrow 0$ $to$ $n-2$ $do$
2.     For$j \leftarrow i+1$ to $n-1$ $do$
3.         If $A[i] == A[j]$ Return
False
```

4. N-th Fibonacci numbers are calculated by the system:

$$\begin{cases} F_n = F_{n-1} + F_{n-2}, \text{nếu } n \geq 2 \\ F_0 = 0; F_1 = 1, \end{cases}$$

The above formula creates the Fibonacci sequence as

follows:$\{0, 1,1,2,3,5,8,13,21,24, \dots\}$

a. Construct an iterative algorithm to calculate the n-th Fibonacci, where n is the input size.

b. Analyze and determine the computational complexity of the suggested algorithm.

c. Implement the algorithm in Python programming language.

5. **Multiply two square matrices with size n defined as follows**:

$$\begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & \cdots & c_{nn} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \times \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{nn} \end{bmatrix}$$

$$\begin{bmatrix} .59 & .32 & .41 \\ .31 & .36 & .25 \\ .45 & .31 & .42 \end{bmatrix} = \begin{bmatrix} .70 & .20 & .10 \\ .30 & .60 & .10 \\ .50 & .10 & .40 \end{bmatrix} \times \begin{bmatrix} .80 & .30 & .50 \\ .10 & .40 & .10 \\ .10 & .30 & .40 \end{bmatrix}$$

a. Writes the algorithm to multiply the square matrix and return the resulting matrix.

b. Analyze and define Big-O of the written algorithm.

c. Implement the algorithm in Python programming language.

6. **Nearest pair (closest pair)**

In the Space with dimension d, the Euclidean distance between two points $A(x_1, x_2, \dots, x_d)$ and $B(y_1, y_2, \dots, y_d)$ is determined by the formula:

$$D(A, B) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \cdots + (x_d - y_d)^2}$$

Suppose there are n points in the space. It is required to find **closest pair**.

a. Design an algorithm (in form of pseudocode) for the problem. The algorithm returns a pair of points and distances between them.
b. Analyze and determine the computational complexity of the written algorithm.
c. Implement the algorithm using Python programming language.

Range(n, -1, -1)