**DESIGN AND ANALYSIS OF ALGORITHMS**

**LAB 0: Introduction**

# Algorithm

An algorithm is a sequence of unambiguous instructions for solving a problem, i.e., for obtaining a required output for any legitimate input in a finite amount of time.

# Algorithm Description

Algorithms can be written in pseudocode (or a programming language). They should be described as functions including following parts:

1. Function name
2. List of input parameters
3. Objective of the function
4. Input, output description
5. Step-by-step procedure processing input to get output
6. Output return (if needed)

Example:

|  |
| --- |
| Algorithm ***SelectionSort(A[0..n-1])***  **//The algorithm sorts a given array by selection sort**  **//Input: An array A[0..n-1] of orderable elements**  **//Output: Array A[0..n-1] sorted in ascending order**  **for i 🡨 0 to n – 2 do**  **min 🡨 i**  **for j 🡨 i + 1 to n – 1 do**  **if A[j] < A[min]**  **min 🡨 j**  **swap A[i] and A[min]**  **return A** |

# Analysis of Algorithm Efficiency as Counting of basic operation

Time efficiency is analyzed by determining the number of repetitions of the *basic operation* as a function of *input size*

*Basic operation*: the operation that contributes the most towards the running time of the algorithm

For some algorithms, efficiency depends on form of input:

Worst case (main concern for this course): Cworst(*n*) – maximum over inputs of size *n*

Best case (not important): Cbest(*n*) – minimum over inputs of size *n*

Average case (study in advanced courses): Cavg(*n*) – “average” over inputs of size *n*

Example:

|  |
| --- |
|  |

Worst case: n key comparisons, C(n) = n

# Exercises

For each problem, do the following:

1. Use Python to present the algorithm.
2. Determine the time efficiency of the algorithm [in the worst case] (as a function of input size).
3. Design and analyze a **non-recursive** algorithm to find the sum of the following series.
4. Design and analyze a **non-recursive** algorithm to find the sum of the following series.
5. Design and analyze a **non-recursive** algorithm to find the sum of the following series.
6. Design and analyze a **non-recursive** algorithm to find the factorial of a positive integer n, denoted by n!
7. Design and analyze a **non-recursive** algorithm to check whether all elements in an array are distinct.
8. Design and analyze a **non-recursive** algorithm to find the maximum element in an array, supposing that all elements in the array are unique.
9. Design and analyze an algorithm for **multiplying** () two matrices
10. Design and analyze an algorithm for **multiplying** () a matrix with a number
11. Design and analyze an algorithm for **subtracting** () two matrices
12. Design and analyze an algorithm for **adding** () two matrices