Algoriothm

==========

Algorithm is a step-by-step procedure, which defines a set of instructions to be executed in a certain order to get the desired output. Algorithms are generally created independent of underlying languages, i.e. an algorithm can be implemented in more than one programming language.

From the data structure point of view, following are some important categories of algorithms −

Search − Algorithm to search an item in a data structure.

Sort − Algorithm to sort items in a certain order.

Insert − Algorithm to insert item in a data structure.

Update − Algorithm to update an existing item in a data

structure.

Delete – Algorithm to delete an existing item from a data structure.

Characteristics of an Algorithm

Not all procedures can be called an algorithm. An algorithm should have the following characteristics –

Unambiguous – Algorithm should be clear and unambiguous. Each of its steps (or phases), and their inputs/outputs should be clear and must lead to only one meaning.

Input – An algorithm should have 0 or more well-defined inputs.

Output – An algorithm should have 1 or more well-defined outputs, and should match the desired output.

Finiteness − Algorithms must terminate after a finite number of steps.

Feasibility − Should be feasible with the available resources.

Independent − An algorithm should have step-by-step directions, which should be independent of any programming code.

design an algorithm to add two numbers and display the result.

Step 1 − START

Step 2 − declare three integers a, b & c

Step 3 − define values of a & b

Step 4 − add values of a & b

Step 5 − store output of step 4 to c

Step 6 − print c

Step 7 − STOP

Algorithms tell the programmers how to code the program. Alternatively, the algorithm can be written as −

Step 1 − START ADD

Step 2 − get values of a & b

Step 3 − c ← a + b

Step 4 − display c

Step 5 − STOP

Algorithm Complexity

Suppose X is an algorithm and n is the size of input data, the time and space used by the algorithm X are the two main factors, which decide the efficiency of X.

Time Factor − Time is measured by counting the number of key operations such as comparisons in the sorting algorithm.

Space Factor − Space is measured by counting the maximum memory space required by the algorithm.

The complexity of an algorithm f(n) gives the running time and/or the storage space required by the algorithm in terms of n as the size of input data.

Space Complexity

Space complexity of an algorithm represents the amount of memory space required by the algorithm in its life cycle. The space required by an algorithm is equal to the sum of the following two components −

A fixed part that is a space required to store certain data and variables, that are independent of the size of the problem. For example, simple variables and constants used, program size, etc.

A variable part is a space required by variables, whose size depends on the size of the problem. For example, dynamic memory allocation, recursion stack space, etc.

Space complexity S(P) of any algorithm P is S(P) = C + SP(I), where C is the fixed part and S(I) is the variable part of the algorithm, which depends on instance characteristic I. Following is a simple example that tries to explain

Time Complexity

Time complexity of an algorithm represents the amount of time required by the algorithm to run to completion. Time requirements can be defined as a numerical function T(n), where T(n) can be measured as the number of steps, provided each step consumes constant time.

Data Structures - Asymptotic Analysis

Asymptotic analysis of an algorithm refers to defining the mathematical boundation/framing of its run-time performance. Using asymptotic analysis, we can very well conclude the best case, average case, and worst case scenario of an algorithm.

Asymptotic analysis is input bound i.e., if there's no input to the algorithm, it is concluded to work in a constant time. Other than the "input" all other factors are considered constant.

Usually, the time required by an algorithm falls under three types −

Best Case − Minimum time required for program execution.

Average Case − Average time required for program execution.

Worst Case − Maximum time required for program execution.

Asymptotic Notations

Following are the commonly used asymptotic notations to calculate the running time complexity of an algorithm.

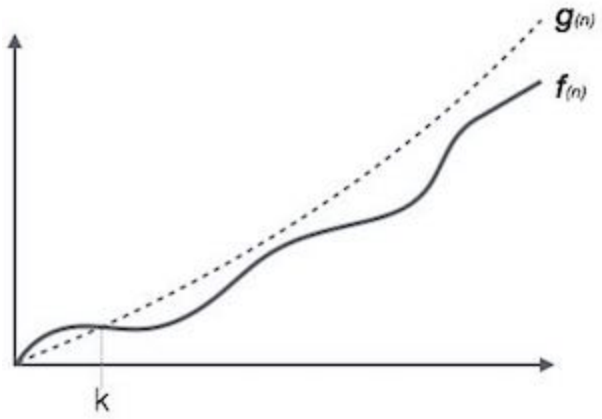
O Notation

Ω Notation

θ Notation

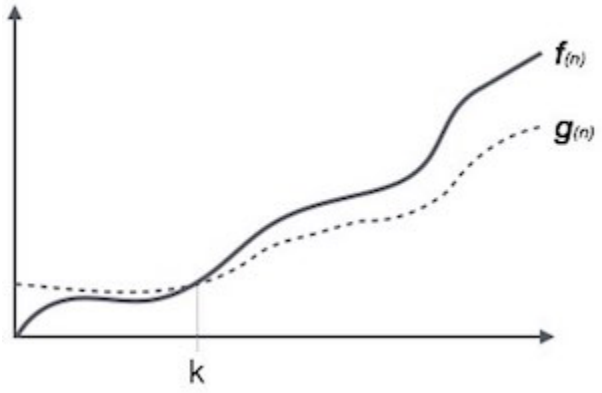
Big Oh Notation, O

The notation O(n) is the formal way to express the upper bound of an algorithm's running time. It measures the worst case time complexity or the longest amount of time an algorithm can possibly take to complete.



Omega Notation, Ω

The notation Ω(n) is the formal way to express the lower bound of an algorithm's running time. It measures the best case time complexity or the best amount of time an algorithm can possibly take to complete.



Theta Notation, θ

The notation θ(n) is the formal way to express both the lower

bound and the upper bound of an algorithm's running time. It is represented as follows −

Theta Notation

$\theta(f(n)) = \{ g(n)$ if and only if $g(n) = O(f(n))$ and $g(n) = \Omega(f(n))$ for all $n > n0. \}$

Common Asymptotic Notations

Following is a list of some common asymptotic notations −