Lecture 6-Algorithm Design and Analysis

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 c ategories 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 I n a data structure.
- Delete − Algorithm to delete an existing item from a data structure.

Problem Development Steps

The following steps are involved in solving computational problems.

- Problem definition
- Development of a model
- Specification of an Algorithm
- Designing an Algorithm
- Checking the correctness of an Algorithm
- Analysis of an Algorithm
- Implementation of an Algorithm
- Program testing
- Documentation

Characteristics of Algorithms

The main characteristics of algorithms are as follows −

- Algorithms must have a unique name
- Algorithms should have explicitly defined set of inputs and outputs
- Algorithms are well-ordered with unambiguous operations

- Algorithms halt in a finite amount of time. Algorithms should not run for infinity, i.e., an algorithm must end at some point

Algorithm Complexity

Sometimes, there are more than one way to solve a problem. We need to learn how to compare the performance different algorithms and choose the best one to solve a particular problem. 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 $S(P) = C + S(I)$, where C is the fixed part and S(I) is the variable part of the algorithm, which depends on instance characteristic I.

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.

For example, addition of two n-bit integers takes n steps. Consequently, the total computational time is T(n) = c*n, where c is the time taken for the addition of two bits. Here, we observe that T(n) grows linearly as the input size increases.

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. Asymptotic analysis refers to computing the running time of any operation in mathematical units of computation. For example, the running time of one operation is computed as f(n) and may be for another operation it is computed as g(n2). This means the first operation running time will increase linearly with the increase in n and the running time of the second operation will increase exponentially when n increases. Similarly, the running time of both operations will be nearly the same if n is significantly small.

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

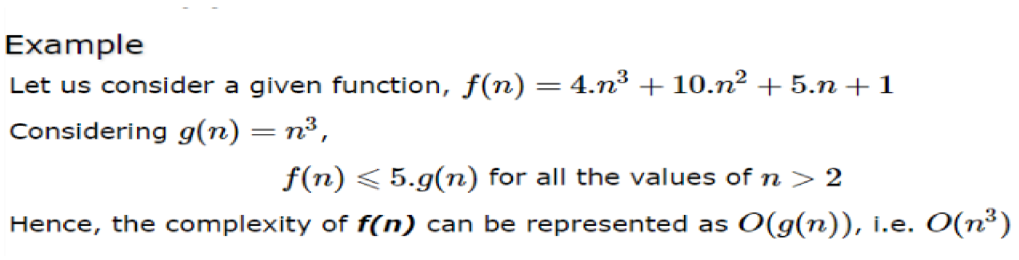
Following are the commonly used asymptotic notations to calculate the running time complexity of an algorithm.

- 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 al gorithm can possibly take to complete.

For example, for a function f(n)

O(f(n)) = { g(n) : there exists c > 0 and n0 such that g(n) ≤ c.f(n) for all n > n0. }



Example

Let us consider a given function, $f(n) = 4.n^3 + 10.n^2 + 5.n + 1$

Considering $g(n) = n^3$,

$$f(n) \leqslant 5.g(n) \text{ for all the values of } n > 2$$

Hence, the complexity of ***f(n)*** can be represented as $O(g(n))$, i.e. $O(n^3)$

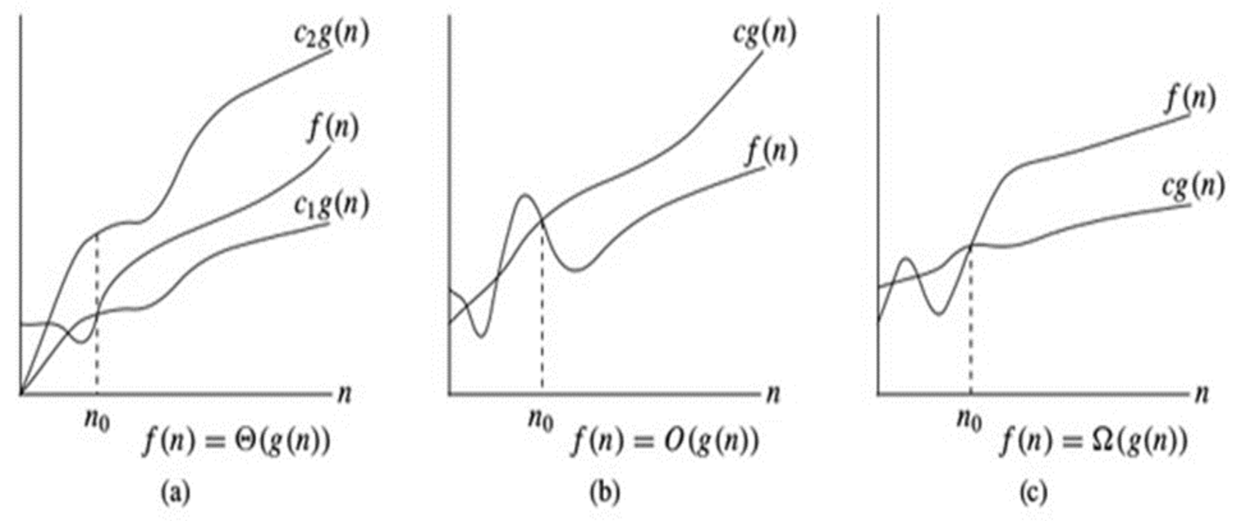
Omega Notation, Ω The notation Ω(n) is the formal way to express the lower bound of an algorithm's running time. It measures the best case time complexity or the best amount of time an algorithm can possibly take to complete.

For example, for a function f(n)

Ω(f(n)) ≥ { g(n) : there exists c > 0 and n0 such that g(n) ≤ c.f(n) for all n > n0. }

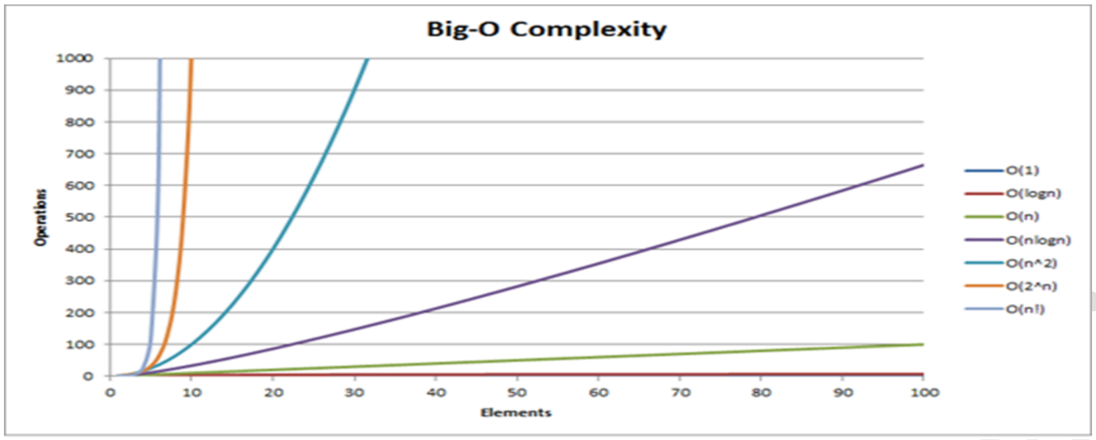
Theta Notation, θ The notation θ(n) is the formal way to express both the lower bound and the upper bound of an algorithm's running time. It is represented as follows –

θ(f(n)) = { g(n) if and only if g(n) = O(f(n)) and g(n) = Ω(f(n)) for all n > n0. }

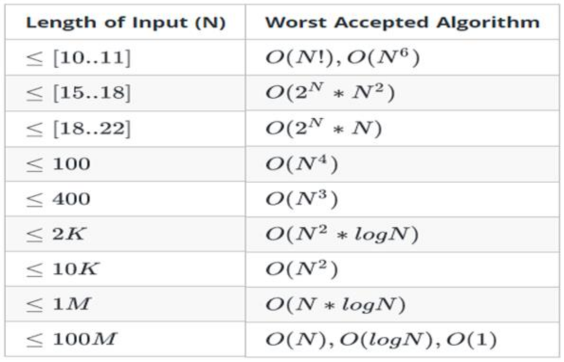


Common Asymptotic Notations

Following is a list of some common asymptotic notations:



constant − O(1)
logarithmic − O(log n)
linear − O(n)
n log n − O(n log n)
quadratic − O(n2)
cubic − O(n3)
polynomial − nO(1)
exponential − 2O(n)
constant − O(1)



Length of Input (N)	Worst Accepted Algorithm
$\leq [10..11]$	$O(N!), O(N^6)$
$\leq [15..18]$	$O(2^N * N^2)$
$\leq [18..22]$	$O(2^N * N)$
≤ 100	$O(N^4)$
≤ 400	$O(N^3)$
$\leq 2K$	$O(N^2 * logN)$
$\leq 10K$	$O(N^2)$
$\leq 1M$	$O(N * logN)$
$\leq 100M$	$O(N), O(logN), O(1)$