## WEEK-2

## Algorithm Analysis

* Analysis of efficiency of an algorithm can be performed by usinf space complexity and time complexity of the algorithm.

1. **Space complexity:**

* Space complexity refers to the total amount of memory space used by an algorithm/program, including the space of input values for execution. Calculate the space occupied by variables in an algorithm.
* The best algorithm/program should have a low level of space complexity. The less space required, the faster it executes.

1. **Time complexity:**

* Time complexity **represents the number of times a statement is executed**.
* The time complexity of an algorithm is NOT the actual time required to execute a particular code, since that depends on other factors like programming language, operating software, processing power, etc.

**Worst Case, Average Case and Best Case**

1. **Best case**

* Fastest time to complete, with optimal inputs chosen.
* For example, the best case for a linear search algorithm is key will present at the beginning of array.

1. **Worst case**

* Slowest time to complete..
* For example, In linear search element present at the end of array

**3. Average case**

* Here key element present at middle of given array

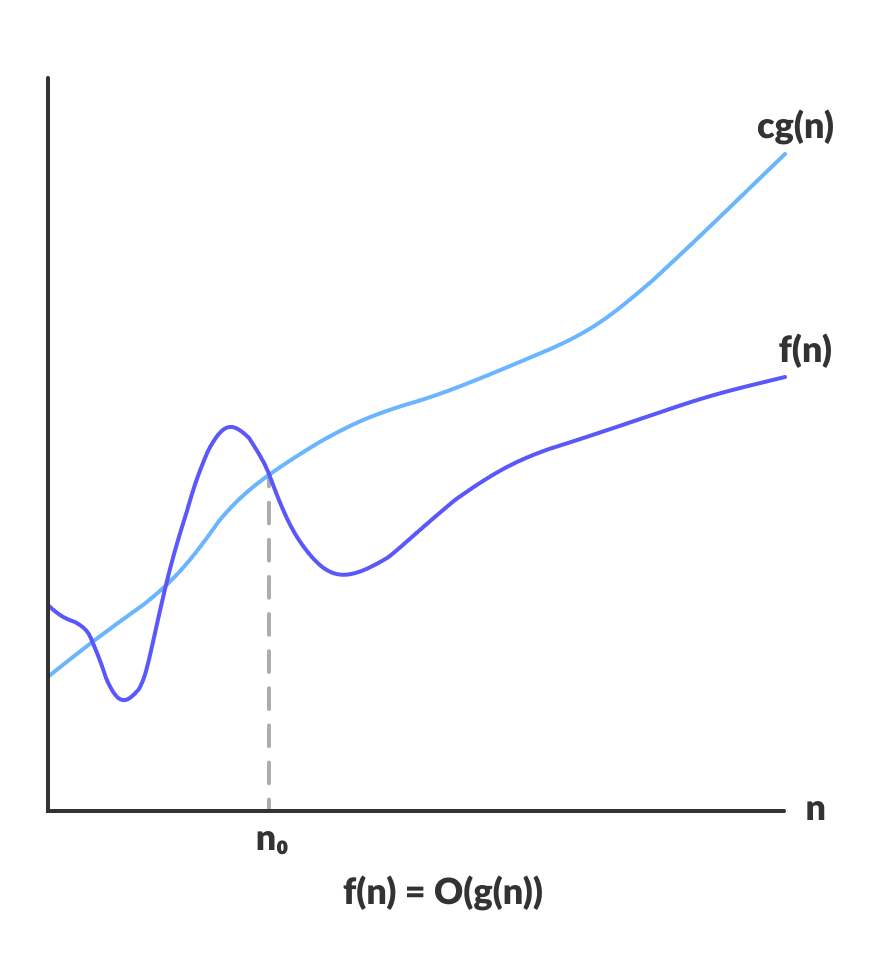
**Asymptotic Notations**

* Asymptotic notations are the mathematical notations used to describe the running time of an algorithm when the input tends towards a particular value or a limiting value.
* For example: In bubble sort, when the input array is already sorted, the time taken by the algorithm is linear i.e. the best case.
* But, when the input array is in reverse condition, the algorithm takes the maximum time (quadratic) to sort the elements i.e. the worst case.
* When the input array is neither sorted nor in reverse order, then it takes average time. These durations are denoted using asymptotic notations.

There are mainly three asymptotic notations:

1. Big-O notation
2. Omega notation
3. Theta notation
4. **Big-O Notation (O-notation)**

Big-O notation represents the upper bound of the running time of an algorithm. Thus, it gives the worst-case complexity of an algorithm.



* Big-O gives the upper bound of a function

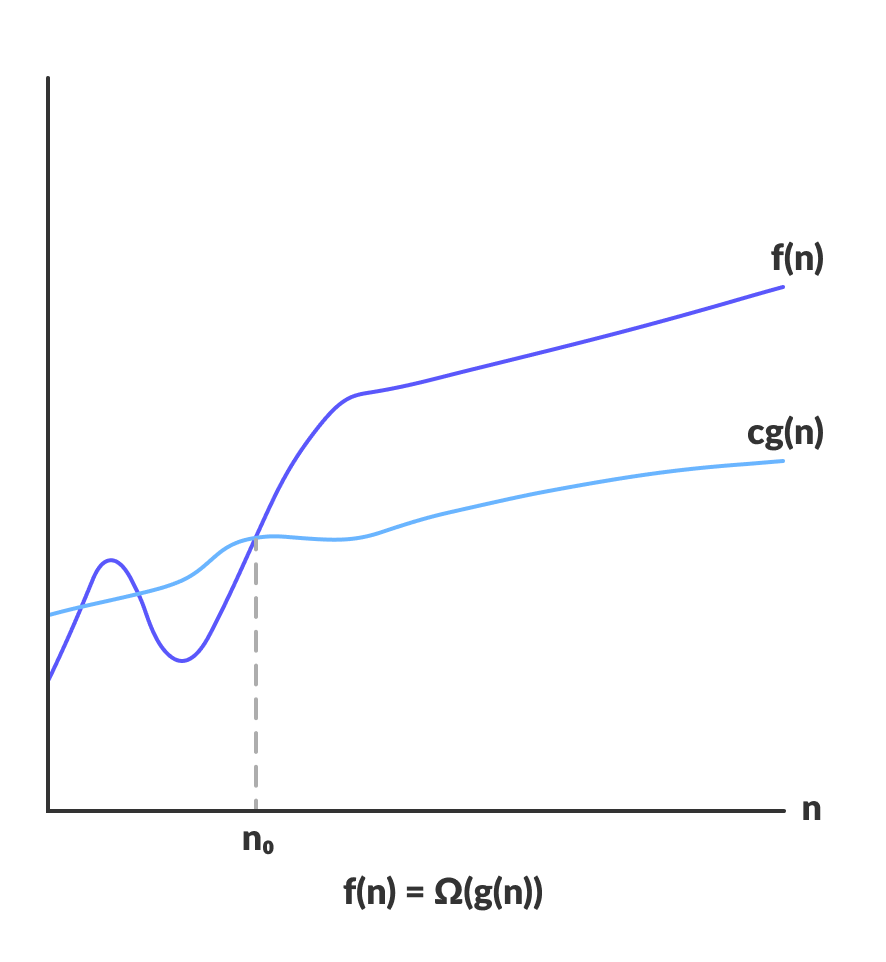
**O(g(n)) = { f(n): there exist positive constants c and n0**

**such that 0 ≤ f(n) ≤ cg(n) for all n ≥ n0 }**

* The above expression can be described as a function f(n) belongs to the set O(g(n)) if there exists a positive constant c such that it lies between 0 and cg(n), for sufficiently large n.
* For any value of n, the running time of an algorithm does not cross the time provided by O(g(n)).
* Since it gives the worst-case running time of an algorithm, it is widely used to analyze an algorithm as we are always interested in the worst-case scenario.

**Omega Notation (Ω-notation)**

* Omega notation represents the lower bound of the running time of an algorithm. Thus, it provides the best case complexity of an algorithm.



* Omega gives the lower bound of a function

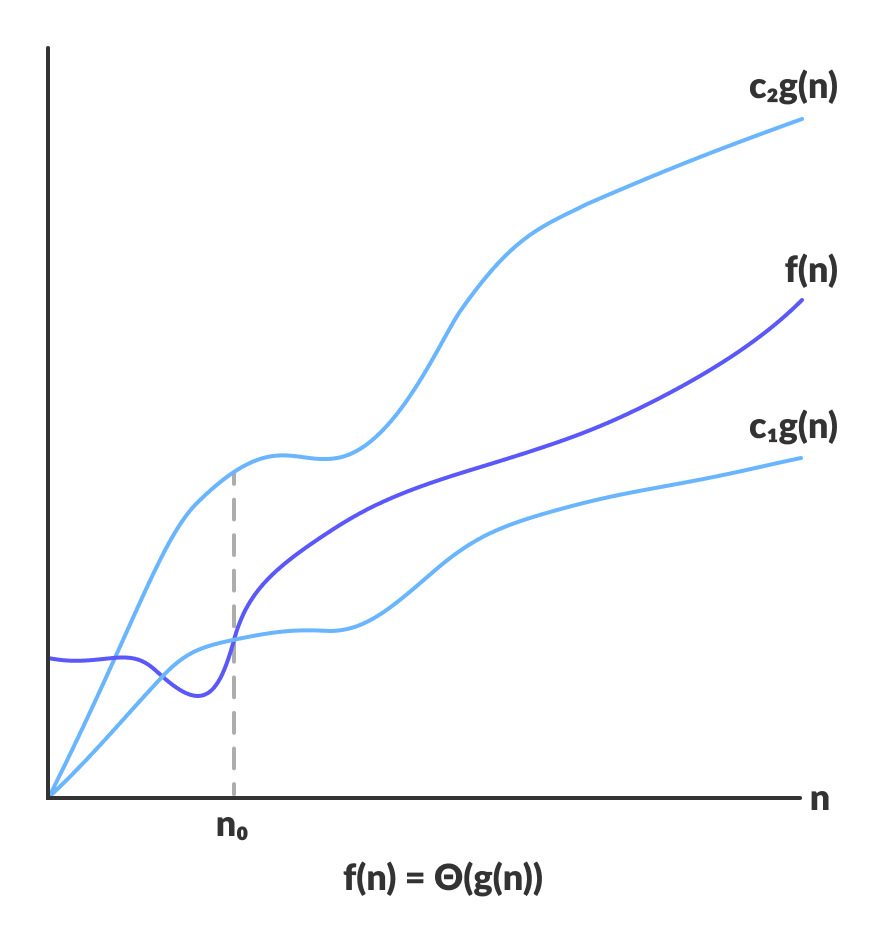
**Ω(g(n)) = { f(n): there exist positive constants c and n0**

**such that 0 ≤ cg(n) ≤ f(n) for all n ≥ n0 }**

* The above expression can be described as a function f(n) belongs to the set Ω(g(n)) if there exists a positive constant c such that it lies above cg(n), for sufficiently large n.
* For any value of n, the minimum time required by the algorithm is given by Omega Ω(g(n)).

**Theta Notation (Θ-notation)**

* Theta notation encloses the function from above and below.
* Since it represents the upper and the lower bound of the running time of an algorithm, it is used for analysing the average-case complexity of an algorithm.



* Theta bounds the function within constants factors
* For a function g(n), Θ(g(n)) is given by the relation:

**Θ(g(n)) = { f(n): there exist positive constants c1, c2 and** n0

such that 0 ≤ c1g(n) ≤ f(n) ≤ c2g(n) for all n ≥ n0 }

* The above expression can be described as a function f(n) belongs to the set Θ(g(n)) if there exist positive constants c1 and c2 such that it can be sandwiched between c1g(n) and c2g(n), for sufficiently large n.
* If a function f(n) lies anywhere in between c1g(n) and c2g(n) for all n ≥ n0, then f(n) is said to be asymptotically tight bound.

**Common Asymptotic Notations**

Following is a list of some common asymptotic notations

|  |  |
| --- | --- |
| constant | Ο(1) |
| logarithmic | Ο(log n) |
| linear | Ο(n) |
| n log n | Ο(n log n) |
| quadratic | Ο(n2) |
| cubic | Ο(n3) |
| polynomial | nΟ(1) |
| exponential | 2Ο(n) |