



$O(g(n)) = \{ f(n): \text{there exist positive constants } c \text{ and } n_0 \text{ such that } 0 \leq f(n) \leq cg(n) \text{ for all } n \geq n_0 \}$

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.

If we use Θ notation to represent the time complexity of Insertion sort, we have to use two statements for best and worst cases:

- The worst-case time complexity of Insertion Sort is $\Theta(n^2)$.
- The best case time complexity of Insertion Sort is $\Theta(n)$.

The Big-O notation is useful when we only have an upper bound on the time complexity of an algorithm. Many times we easily find an upper bound by simply looking at the algorithm.

Examples :

$\{ 100, \log(2000), 10^4 \}$ belongs to $O(1)$

$U \{ (n/4), (2n+3), (n/100 + \log(n)) \}$ belongs to $O(n)$

$U \{ (n^2+n), (2n^2), (n^2+\log(n)) \}$ belongs to $O(n^2)$