

# Java - Introduction to Programming

## Lecture 8

### Time & Space Complexity



Time complexity of an algorithm quantifies the amount of time taken by an algorithm to run as a function of the length of the input.

Types of notations

1.  $O$ -notation: It is used to denote asymptotic upper bound. For a given function  $g(n)$ , we denote it by  $O(g(n))$ . Pronounced as "big-oh of  $g$  of  $n$ ". It is also known as worst case time complexity as it denotes the upper bound in which the algorithm terminates.
2.  $\Omega$ -notation: It is used to denote asymptotic lower bound. For a given function  $g(n)$ , we denote it by  $\Omega(g(n))$ . Pronounced as "big-omega of  $g$  of  $n$ ". It is also known as best case time complexity as it denotes the lower bound in which the algorithm terminates.
3.  $\Theta$ -notation: It is used to denote the average time of a program.

**Examples :**

```
int a = 0;
for (int i = 1; i <= n; i++)
{
    a = a + 1;
}
```

Linear Time Complexity.  $O(n)$

### Comparison of functions on the basis of time complexity

It follows the following order in case of time complexity:

$O(n^n) > O(n!) > O(n^3) > O(n^2) > O(n \cdot \log(n)) > O(n \cdot \log(\log(n))) > O(n) > O(\sqrt{n}) > O(\log(n)) > O(1)$

Note: Reverse is the order for better performance of a code with corresponding time complexity, i.e. a program with less time complexity is more efficient.

### Space Complexity

Space complexity of an algorithm quantifies the amount of time taken by a program to run as a function of length of the input. It is directly proportional to the largest memory your program acquires at any instance during run time.

For example: *int* consumes 4 bytes of memory.