Time complexity

**Time Complexity** of an algorithm is the representation of the amount of **time** required by the algorithm to execute to completion. **Time** requirements can be denoted or defined as a numerical function T(n)

**Asymptotic Notations** are the expressions that are used to represent the complexity of an algorithm

**Best Case**: In which we analyse the performance of an algorithm for the input, for which the algorithm takes less time or space.

**Worst Case**: In which we analyse the performance of an algorithm for the input, for which the algorithm takes long time or space.

**Average Case**: In which we analyse the performance of an algorithm for the input, for which the algorithm takes time or space that lies between best and worst case.

**Types of Data Structure Asymptotic Notation**

1. Big-O Notation (Ο) – Big O notation specifically describes worst case scenario.  
2. Omega Notation (Ω) – Omega(Ω) notation specifically describes best case scenario.  
3. Theta Notation (θ) – This notation represents the average complexity of an algorithm

## Big-O Notation (Ο)

Big O notation specifically describes worst case scenario. I

### O(1)

Big O notation O(1) represents the complexity of an algorithm that always execute in same time or space regardless of the input data.

**O(1) example**  
The following step will always execute in same time(or space) regardless of the size of input data.

Accessing array index(int num = arr[5])

### O(n)

Big O notation O(N) represents the complexity of an algorithm, whose performance will grow linearly (in direct proportion) to the size of the input data.

**O(n) example**  
The execution time will depend on the size of array. When the size of the array increases, the execution time will also increase in the same proportion (linearly)

Traversing an array

### O(n^2)

Big O notation O(n^2) represents the complexity of an algorithm, whose performance is directly proportional to the square of the size of the input data.

**O(n^2) example**

Traversing a 2D array

1. We analyze time complexity for

-very large input size

- worst case scenario

T(n)= n3+2n2+n+1= O(n3)

T(n)= 5n4+ 5n+1 = 0(n4)

Rule 1:

int a;

a=5;

a++;

simple expression always run constant time= 0(1)

Rule 2

for( int i=0 ; i<n ;i++){

//sample statement

}

Time complexity= 0(n)

Rule -3

for( int i=0 ; i<n ;i++){

for( int i=0 ; i<n ;i++){

//sample statement

}

}

n \* n = 0(n2)

Rule 4:

Void func(){

int a;

a=5;

a++;

for( int i=0 ; i<n ;i++){

//sample statement

}

for( int i=0 ; i<n ;i++){

for( int i=0 ; i<n ;i++){

//sample statement

}

}}

Time complexity = 0(1)+ 0(n)+0(n2)

Rule 4:

Void fun(){

if(some condition){

for( int i=0 ; i<n ;i++){

//sample statement

}}

else{

for( int i=0 ; i<n ;i++){

for( int i=0 ; i<n ;i++){

//sample statement

}

}

}

}

Time complexity = 0(n2)

Rule: 5

for( int i=0 ; i<n ; i\*=2){

//sample statement

}

for( int i=0 ; i<n ; i/=2){

//sample statement

}

Log(n)