Time Complexity -

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

Ex-
```
cin>>n;
for(int i=0;i<n;i++){
cout<<i;
}
```

User has given the input n and now CPU will perform the operation inside the for loop. CPU will run the for loop for n times. So we can say that time is the function of n times. as n operations are taken place. Here the time is not the actual. It is basically the number of operation performed by CPU.

Now, we can say that Time Complexity is O(n).

Why to study Time and Space Complexity ?

1) Good computer engineer always think about the complexity of code written by him.
2) Computer resources are limited.
3) Measure algorithm to make the efficient program.
4) Improves software design by choosing appropriate data structures and algorithms for specific task.
5) Asked by interviewer after every solution you give.

Algorithm A	Algorithm B
Takes high processing time of CPU.	Takes low processing time of CPU.

Hence, Algorithm B is better.

Space Complexity -

Space Complexity is the amount of space taken by an algorithm to run as a function of the length of input.

Ex - (i) int a = 1;
int b = 5;
int arr[5]; } O(1) → Space

(ii) int n;
cin >> n;
int * arr = new int[n]; } O(n) → Space

O(n) space, as space by array is dependent on n and hence space complexity is O(n).

Units to represent Complexity -

There are three units to represent time and space complexity.

1) Big (O) : Upper Bound
2) Theta (Θ) : Average Case
3) Omega (Ω) : Lower Bound.

FREEMIND

Date

Page

1) Big (O) Notation:

Big (O) Notation is upper bound. That is the worst case time complexity.

Ex- During search, the element is found at last index (n). Hence time complexity is O(n).

2) Theta (Θ) Notation:

Theta (Θ) Notation is average case. That is its represent average time complexity.

Ex- During search, the element is found in the mid of an array. Hence time complexity is Θ(n).

3) Omega (Ω) Notation:

Omega (Ω) Notation is lower bound. That is the base case time complexity.

Ex- During search, the element is found at first index (0). Here time complexity is Ω(1).

Note - We have to focus on the worst case complexity so as to make our algorithm the best.

Big (O) Complexities -

Big (O) Notation is a mathematical concept used to describe the performance or complexity of an algorithm. It provides a way to analyze how the runtime or space requirements grow as the size of the input increases.

The different Big (O) time complexities are as follows.

Date
Page

1) Constant Time:

O(1) complexity is the constant time complexity.

Ex- int a=5; → O(1) Complexity.

2) Linear Time:

O(n) complexity is the linear time complexity.

Ex-
```
for(int i=0; i<n; i++){
    // Code.
}
```

3) Logarithmic Time:

O(log n) is the logarithmic time complexity.

Ex- Binary Search

4) Cubic Time:

O(n³) is the cubic time complexity.

Ex-
```
for(int i=0; i<n; i++){
    for(int j=0; j<n; j++){
        for(int k=0; k<n; k++){
            // Code.
        }
    }
}
```

Questions-

Find the time complexity of given code.

FREEMIND

Date

Page

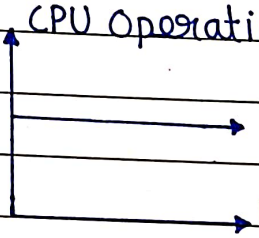
```
for(int i=0; i<n; i++){
    //Code
}
```
→ n times.

```
for(int j=0; j<n; j++){
    //Code
}
```
→ n times.

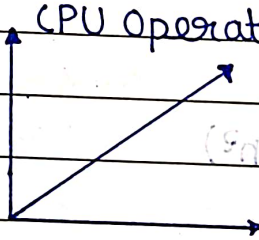
Solution- $O(n+n) \Rightarrow O(2n)$. Time Complexity is $O(n)$.

Graphs-

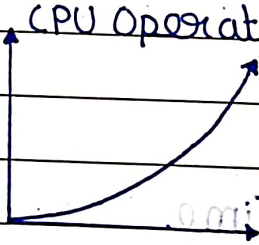
1) $O(1)$



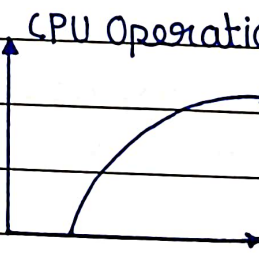
2) $O(n)$



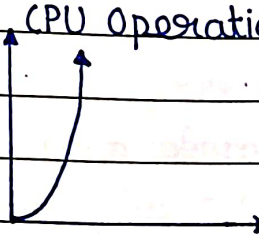
3) $O(n^2)$



4) $O(\log n)$



5) $O(n!)$



Date
Page

Questions -

Write a Big.(O) representation of the following code.

(i) $f(n) = 2n^2 + 3n$.

Solution -

$O(2n^2+3n) \Rightarrow O(2n^2) \Rightarrow O(n^2)$

(ii) $f(n) = 4n^4 + 3n^3$

Solution -

$O(4n^4+3n^3) \Rightarrow O(4n^4) \Rightarrow O(n^4)$

(iii) $f(n) = N^2 + \log N$

Solution -

$O(N^2+\log N) \Rightarrow O(N^2) \Rightarrow O(n^2)$

(iv) $f(n) = 200$

Solution -

$O(200) \Rightarrow O(1) \rightarrow$ Constant Time.

(v) $f(n) = n/4$.

Solution -

$O(n/4) \Rightarrow O(n)$

(vi) $f(n) = n^2 + 5n + 7$

Solution -

$O(n^2+5n+7) \Rightarrow O(n^2)$

FREEMIND

Date

Page

Comparison of Time Complexities -

The comparison of different time complexities from least to highest are as follows.

$O(1)$	Least
$O(\log n)$	
$O(\sqrt{n})$	
$O(n)$	
$O(n \log n)$	
$O(n^2)$	
$O(n^3)$	
$O(2^n)$	
$O(n!)$	
$O(n^n)$ ↓	Highest

Discussion about $O(\log n)$ Complexity -

Suppose that we are given an array and it has stored elements. We have to search for an element. This can be done by Linear search or Binary search.

In linear search, we search element by traversing through array. Thus its time complexity is $O(n)$.

In binary search, we search element by dividing the array into half part. Thus its time complexity is $O(\log n)$.

For linear search, time complexity is $O(n)$ in the worst case as it checks each element one by one. For binary search, time complexity is $O(\log n)$ in the worst case as it halves the search space at each step, but requires a sorted list.