Time and Space Complexity

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

```
cin >> n;
for (int i=0; i<n; 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 as n operations

are taken place. Here the time is not the actual time. It is basically the no. of operations performed by CPU.

Now we can say that Time complexity is O(n). We will discuss that what is O. → Big Oh of n

Why to study time & space complexity ?

1) Good computer engineer always think about the complexity of code written by him.
2) Resources are limited.
3) Measure algorithm to make efficient programs.
4) Asked by interviewer after every solution you give.

✓ Algorithm A → Takes high processing time of CPU

Algorithm B Takes low processing time of CPU.

Hence Algorithm B is better.

Space complexity

Amount of space taken by an algorithm to run as a function of length of input.

```
int a = 1;
int b = 5;
int arr[5];
```
} O(1) space

```
int n;
cin >> n;
int * arr = new int[n];
```
} O(n) space as space by array is dependent

on n and hence space complexity is O(n).

Units to represent complexity

1) Big O : This tells the upper bound complexity of an algorithm.

Ex → Linear search

1	2	3	4	5

Element to search = 1

So at first (0th index), we get the element & hence this is the lower bound i.e least time taken by an algorithm.

Element to search = 5

So at the last (4th index), we get the element & hence this is the upper bound i.e maximum time taken by our algorithm

Time complexity = O(n) → In worst case, n operations are performed.

2) Omega Ω : This is the lower bound i.e the best case time complexity. Ex → 1 was found at index = 0 & hence time complexity = Ω(1).

3) Theta Θ : This tells us that how our algorithm performs in the average case. For ex → Searching for 2, 3, 4 in the array comes under the average case.

Here average case complexity is $\Theta(n)$.

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

Big O Complexities

1) Constant time : $O(1)$ complexity is the constant time complexity.

```
int a = 5;  → O(1) complexity.
```

2) Linear time : $O(n)$ complexity is the linear time complexity.

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

3) Logarithmic time : $O(\log n)$ is the logarithmic time complexity. Ex → Binary search.

4) Quadratic time : $O(n^2)$ is the quadratic time complexity.

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

    }
}
```

5) Cubic time : $O(n^3)$ is the cubic time comblexity.

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

    }
}
}
```

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

`...`

`}`

`for (int j=0; j<n ; j++) {` } n times

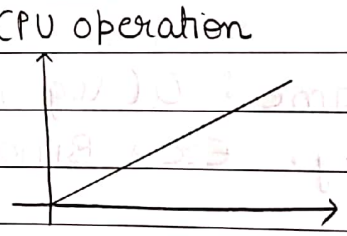
`...`

`}`

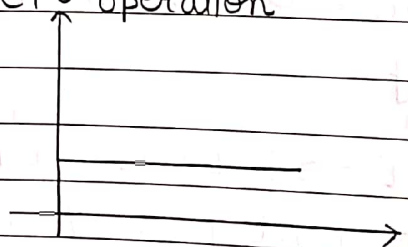
$O(n+n) = O(2n) = O(n)$ ← 2 is ignored

Graphs

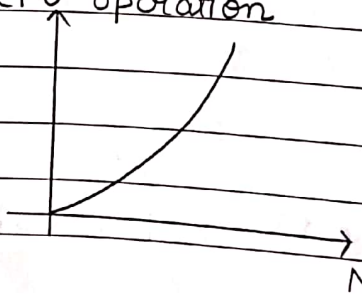
(1) $O(n)$



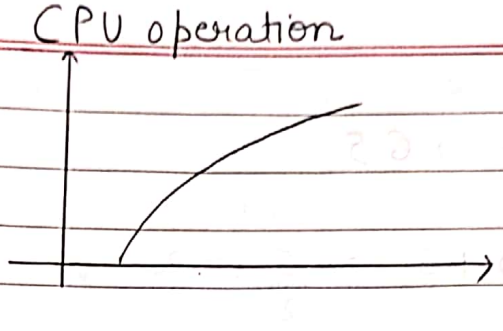
(2) $O(1)$



(3) $O(n^2)$



(4) $O(\log n)$



Ex → $f(n) = 2n^2 + 3n$

Write Big O representation

Neglect lower terms

$O(2n^2) \rightarrow O(n^2)$ as 2 will be ignored.

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

$O(4n^4) = O(n^4)$ as 4 will be ignored

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

$O(n^2)$

(iv) $f(n) = 200$

$O(1)$ → constant time

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

$O\left(\frac{n}{4}\right) \rightarrow O(n)$ as 4 will be neglected.

Comparasion of time complexities

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

↳ Least ↳ Highest

Discussion about $O(\log n)$ complexity.

Suppose that we are given an array & it has sorted elements. We have to search for an element. This can be done by binary search

1 2 3 4 5 6

Element → 5

$mid = \frac{0+5}{2} = \frac{5}{2} = 2$

6 > 3, then search in right array. This indicates that our search space has reduced.

4 5 6

$mid \rightarrow \frac{3+5}{2} = 4$

arr[4] = 5 is equal to 5. Hence found the element.

No. of operations reduces & also search space reduces like

$N \rightarrow \frac{N}{2} \rightarrow \frac{N}{4} \rightarrow \cdots \frac{N}{2^k}$

Hence this gives O(log n) complexity.

Some examples.

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

Time complexity is O(m+n) as not nested.

2)
```
for (int i=0; i<n; i++) {
    for (int i=0; i<n; i++) { .. }
}
```

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

Time complexity = $O(n^2+n) = O(n^2)$ as we take in consideration higher order terms.

3)
```
for (int i=0 ; i<n ; i++) {
        for (int j=n ; j>i ; j--) {
                --
        }
}
```

Time complexity = $O(n \times n) = O(n^2)$

Inner loop runs at max for n times for $i=0$.

```
int n; cin >> n;
```

Note → `int * b = new int [n^2]`

Space complexity = $O(n^2)$