Time & Space complexity

Time Complexity → Amount of time taken by an algorithm to run as a function of length of input.

eg →
```
cin>>n;
for (int i=0; i<n; i++){
    cout<<"Hello";
}
```
→ CPU is working in this loop again and again (n times).

↓

Means CPU performs this cout operation n times.

If we increase n then operations by CPU will be increased.

~~to~~ So the time taken (no. of operations) is directly propotional to n.

$$\boxed{f \propto N}$$

We does not take this time in terms of actual time but we take it as CPU operations.

So the time complexity of this loop is O(N).

Why to study Time and Space complexity?

1. A Good ~~cs~~ computer engineer always thinks about the complexity of the code written by him.
2. Resources are limited. (Here resources means CPU and memory)
3. measures algos to make efficient prog.
4. Interviewer asked after every solution/program we give.

Algo A	Algo B	
→ does same work	→ same work	So in this case algo B is better.
→ CPU → High processing	→ CPU → Low processing	

Space Complexity → Amount of space taken by an algo to run as a function of length of input.

```
int a = 1;
int b[5];
```

} These variables doesn't considered in S.C as if we increase n it there will be no effect in these.

```
int n;
cin >>n;
int *b = new int [n];

for (int i=0; i<n ; i++){
    cout << b[i];
}
```

Here if n = 2 ⇒ b[2] if n = 2000 ⇒ b[2000]

So the space complexity ⇒ O(n)

Unit to represent complexity ->

1. Big O → shows Upper bound of an algo. (max time)
2. Theta θ → Average case
3. Omega Ω → algo's lower bound.

Eg → Linear search

1	2	3	4	5	6

search 1 → Omega $\Omega(1)$

search 6 → Big O $O(n)$ ✓ better to use this.

search 3 → Theta $\theta(n/2)$

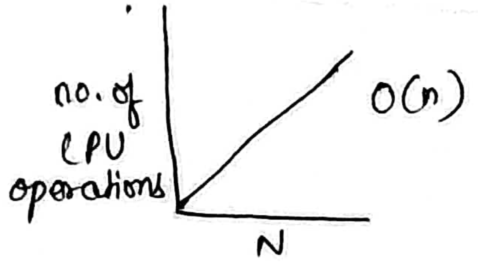
Big O: Complexities –

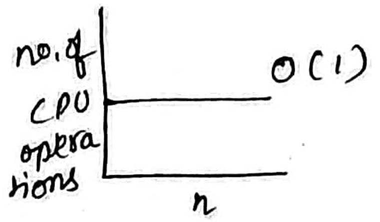
1. Constant time – O(1) → `int a = 5;`
2. Linear time – O(n)
3. Logarithmic time – O(log n)
4. Quadratic time – O(n²)
5. Cubic time – O(n³)

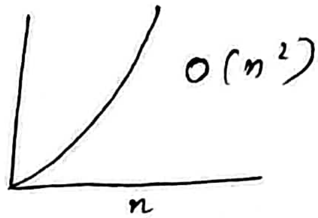
```
for(int i=0; i<n; i++){
    cout << i;
}
```

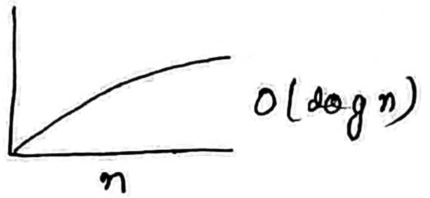
```
for (int i=0; i<n; i++){
    for (int j=0; j<n; i++){
        ~~
    }
}
```

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









Questions –

1. $f(n) = 2n^2 + 3n$ (ignore constants)

   $T.C = O(n^2)$

2. $f(n) = 4n^4 + 3n^3$

   $T.C = O(n^4)$

3. $f(n) = n^2 + \log n = O(n^2)$

4. $200 \Rightarrow O(200) = O(1)$

5. $f(n) = n/4 \Rightarrow O(n/4) = O(n)$

increasing in this order.

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

least complex

Q. $O(\log(n))$

1	2	3	4	5	6	7

sorted array.

We have to find an element in this array so we have two ways → 1. Linear search

$T.C = O(n)$

2. Binary Search → $T.C = O(\log(n))$

1	2	3	4	5	6	7

, target = 6

In this we do not search from the start but we search from mid element.

so size of array →

$n \rightarrow n/2 \rightarrow n/4 \rightarrow n/8 \rightarrow \cdots \quad \frac{N}{2^k} = 1$

$\frac{N}{2^k} = 1 \qquad N = 2^k$

$\log n = \log_2 2^k \Rightarrow \boxed{\log n = k}$

→ Some examples -

- for (int i = 0 to < n){
      cout << i;
  } ⟶ O(n)
  for (i = 0 to < m){
      cout << i; ⟶ O(m)
  }

  T.C = O(n) + O(m)
  = O(n+m)

- for (int i → 0 to < n){
      for (j → 0 to < n) ⟶ O(n²)
      — —
      }
  }
  for (int i → 0 to < n){
      cout << . — ⟶ O(n)
  }

  $T.C = O(n^2) + O(n)$
  $= O(n^2)$

- for (int i = 0; i < n; i++){
      for (int j = n; j > i ; i++){
          cout << "Hi"; ⟶ Upper bound (when i = 0)
      }   O(n)
  }

  i = 0
  j = n-i to 1
  ↳ n times

  $\boxed{T.C = O(n^2)}$

◎ Space complexity -

① int a = 5; ← not dependent upon n

$\boxed{S.C = O(1)}$

② int array [5];

$\boxed{S.C = O(1)}$

③ int *a = new int [n]

$\boxed{S.C = O(n)}$

③ int *b = new int [n * n]

$\boxed{S.C = O(n^2)}$