Time Complexity :→ Time Complexity of an algorithm quantifies the amount of time taken by a program to run as a function of length of the input.

Eg

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

Here, in this code, if value of n = 5; then loop will run 5 times.

Hence, we can say that Time Complexity of the given program is linearly proportional to n.

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

Here, as there are two for loops. Therefore,

i = 1 → j = 1 to n
i = 2 → j = 1 to n
⋮
i = n → j to n.

Hence, time complexity ∝ n^2.

Space Complexity :→

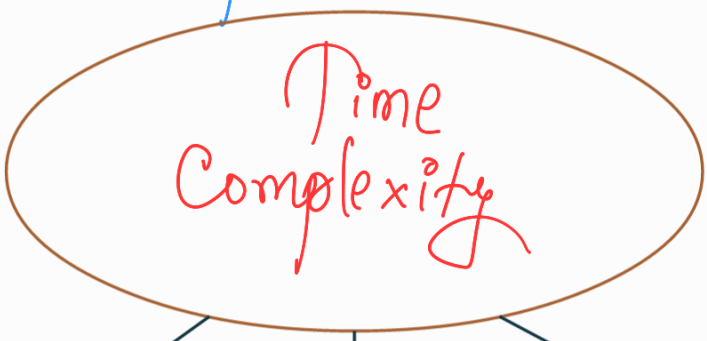
Space Complexity of an algorithm quantifies the amount of time taken by a program to run as a function of length of the input. It is directly proportional to the largest memory your program acquires at any instance during run time.

Eg

```
int n;          → 4 bytes
cin >> n;
int a = 0;      → 4 bytes
for (int i = 1; i < n; i++)   → 4 bytes
{
    a = a + 1;
}
```

Here, Space Complexity of the above code = 12 bytes (irrespective the no. of times the loop is running)

Types and Representation of Time Complexities



- Worst Case - [O (big Oh) Notation]
- Best Case [Ω (big omega notation)]
- Average Case [Θ (big theta) notation]

```
int n, m;
cin>>n>>m;
for(int i=1; i<=n; i++) → n
  {
    for(int j=1; i<=m; i++) → m
      {
        a = a + rand();
      }
  }
for(int k=1; k<=n; k++) → n
  {
    a = a + rand();
  }
```

Time Complexity : $O(n+n+m)$

```
int n;
cin>>n;
int a=0, i=n; → n
while(i>=1)
  {
    a = a+1;
    i /= 2;
  }
```

Explaination :→

$n \rightarrow \frac{n}{2} \rightarrow \frac{n}{4} \ldots$

$\therefore \frac{n}{2^k} \geq 1$

$\Rightarrow n \geq 2^k \Rightarrow \log n \geq \log 2^k$

$\Rightarrow \log n \geq k$

Time Complexity : $O(\log n)$

☺ Dreams ON ☺

HAPPY LEARNING

More Examples on Time Complexity :-

```
int n;
cin >> n;
int count = 0;
for (int i = 1; i <= n; i++)
{
    for (int j = 1; j <= n; j += i)
    {
        count++;
    }
}
```

No. of times, this statement will execute??

Time Complexity (Analysis)

i = 1, then j = 1, 2, 3 ... n times → n terms

i = 2, then j = 1, 3, 5 ... n times → n/2 terms

i = 3 then, j = 1, 4, 7 ... n times → $\frac{n}{3}$ terms

⋮

i = n, then, j = ... n times → $\frac{n}{n}$ = 1 term

Now, on analysing the term, we can now say that :-

$$\text{Total Terms} = \left(n + \frac{n}{2} + \frac{n}{3} \cdots + \frac{n}{n}\right)$$

$$= n\left(1 + \frac{1}{2} + \frac{1}{3} \cdots \frac{1}{n}\right).$$

Therefore, $n\left(1 + \frac{1}{2} + \frac{1}{3} \cdots \frac{1}{n}\right) \leq n \log n$

We know from integration that

$$\int \frac{1}{x}\,dx = \log n$$

$$\frac{1}{1} + \frac{1}{1 \cdot 2} + \frac{1}{2} \cdots = \log n$$

Hence, time complexity is :-

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

Comparision of Function

(1)

Time Complexity →	n	n^2	n^3
$n=1$,	1	1	1
$n=2$,	2	4	8
$n=3$,	3	9	27

So, from the chart beside, we can conclude that,

$O(n)$ is the most efficient and the fastest algorithm, as compared to others.

$\therefore$ $O(n) \rightarrow$ fast

$O(n^3) \rightarrow$ slow

Order of time complexity →

$$O(n) < O(n^2) < O(n^3)$$

(2)

Time Complexity	n	$\log n$
$n=1$	1	$\log(1)=0$
$n=2$	2	$\log_2(2)=1$
⋮		
$n=1024$	1024	$\log_2(1024) = \log_2(2^{10})$
		$= 10 \log_2(2)$
		$= 10$

Hence, we can see from the above chart that the time complexity $O(n)$ isn't efficient if compared with $\log(n)$.

Therefore, order for time complexity order is:-

$\log(n) \rightarrow$ fast and efficient

$O(n) \rightarrow$ slow (will take much time).

$$\log(n) < O(n)$$

(3)

Time Complexity	Sqrt(n)	log(n)
n=1	1	0
n=2	$\sqrt{2}$	log(2)
n=3	$\sqrt{3}$	log(3)
⋮	⋮	⋮

From mathematics, we can simplify it like this :-

$\sqrt{n}$ $\log(n)$

taking log on both sides,

$\log(n^{1/2})$ $\log(\log(n))$

$= \frac{1}{2}\log(n)$ $\log(\log(n))$

So, from the mathematical analysis, we can say that –

$O(\log n) < O(\sqrt{n})$

Here, it means :

$O(\log n) \rightarrow$ fast and efficient

$O(\sqrt{n}) \rightarrow$ slow.

(4) $f(n) = n$, $n <= 1000$

n^2 $n > 1000$

↳ $O(n^2)$ [Overall Time Complexity]

*NOTE :- We consider the time complexity for higher data or high value of n (in this case)

(5) $g1(n) = n$ $n <= 100$

$= n^4$ $n > 100$

} → Time complexity → $O(n^4)$

$g2(n) = n^3$ $n <= 100$

$= n^2$ $n > 100$

} → Time Complexity → $O(n^2)$

Tell the time complexity of $g1(n) + g2(n)$

$= g1(n) + g2(n)$ {overall time complexity}

$= O(n^4) + O(n^2)$

$= O(n^4)$ (Always take maximum wala case).

END OF TIME AND SPACE COMPLEXITY

☺ HAPPY LEARNING ! DREAMS ONN ☺