Lecture 11: Time & Space Complexity

What is Time Complexity?

- It is the amount of time taken by the algorithm to run as a function of length of input

Why do we need it?

- Comparisons of Algorithm
- In order to write efficient code.

Representation of Time Complexity

- Big O notation (upper bound) (Worst case complexity)
- Theta Θ notation (for avg. case complexity)
- Omega Ω notation (lower bound) (best case complexity)

→ Constant time → O(1) - for (i=0; i<$\overset{\text{constant}}{10}$; i++)

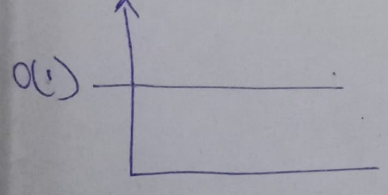
→ Linear time → O(n) - for (i=0; i<$\overset{\text{variable}}{n}$; i++

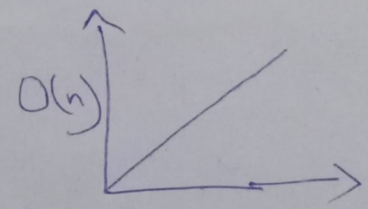
→ Logarithmic time → O(logn) - Binary Search

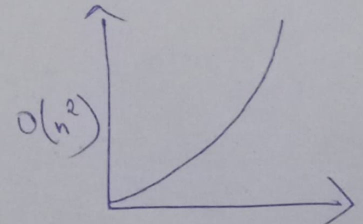
→ Quadratic time → $O(n^2)$

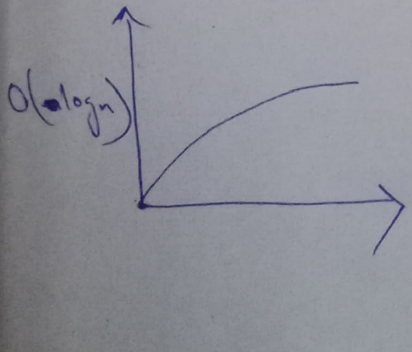
→ Cubic time → $O(n^3)$

Graphs









Best to Worst Time Complexity Table

$O(1)$

$O(\log N)$

$O(N)$

$O(N \log N)$

$O(N^2)$

$O(N^3)$

$O(2^n)$

$O(N!)$

Question:

1> $f(n) \rightarrow 2n^2 + 3n$ - $O(n^2)$

2> $f(n) \rightarrow 4n^4 + 3n^3$ - $O(n^4)$

3> $f(n) \rightarrow N^2 + \log N$ - $O(N^2)$

4> $f(n) \rightarrow 120.01$ - $O(1)$

5> $f(n) \rightarrow 3n^3 + 2n^2 + 5$ - $O(N^3)$

6> $f(n) \rightarrow \frac{n^3}{500}$ - $O(n^3)$

7> $f(n) \rightarrow 5n^2 + \log n$ - $O(n^2)$

8> $f(n) \rightarrow \frac{n+4}{4}$ - $O(n)$

9> $f(n) \rightarrow \frac{n}{4}$ - $O(n)$

* How to avoid stuck in TLE:-

Theres rule which states that in todays time most of the modern machines are capable of executing 10^8 operations per second

* Time Complexity table according to Constrains

		Time Complexity (At max)
Constrains	$< [10 \dots 11]$	$O(n!), O(n^6)$
$1 < n < 10^6$	$< [15 \dots 18]$	$O(2^n * n^2)$
$1 < n < 1000$	< 100	$O(n^4)$
	< 400	$O(n^3)$
	< 2000	$O(n^2 * \log n)$
	$< 10^4$	$O(n^2)$
	$< 10^6$	$O(n \log n)$
	$< 10^8$	$O(n), O(\log n)$

* Space Complexity

- amount of memory consumed by the program to run as a function of the length of the input.

```
int a=0, b=0
for (i=0; i<N; i++)
    a=a+rand();
for (j=0; j<M; j++)
   b=b+rand();
}
```

Space complexity is $O(1)$ no matter how many variables declared.

But, suppose we have the following code

```
func()
{
  int arr[5] = {1, 2, 3, 4, 5};
  =
}
```

S.C → O(1) since size of array is fixed.

```
int n;
cin>>n
vector<int> v(n);
```

} S.C- O(n) Since size is a variable

```
for (0-n)
{
 vector<int> v(n);
  for (0-n)
   {
   }
}
```

O(n) - since size is a variable