→ Time & Space complexity ←

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

Function of length of input = Time complexity is directly proportional to the input value. ($f \propto N$)

If we increase or decrease the input value, the time complexity will increase or decrease respectively.

$f \propto N$

Here, 'f' refers to the function of time complexity or no. of operations performed by the CPU & that is directly proportional to the input value.

Time complexity = CPU Time utilisation

→ **Why to study ?**

i) To reduce CPU operations.

ii) Resources are limited.

iii) To find optimal & efficient solution.

Date: .....
Page: .....

→ Suppose there is,
Algo A → Prints 1 TO N & It takes CPU high processing.
Algo B → " " & " " " low "
So, Algo B will be an optimal solution for print 1 → N.

→ Space complexity - Amount of space taken by an algorithm to run as a function of length of input.

If we increase or decrease the input value, the space complexity will increase or decrease respectively.

Suppose, `int a = 1;` `int n;` // variable
(Array) `int b[5];` `cin >> n;`
`for (int i = 0; i < n; i++) { ------ }`

So, when we increase or decrease the value of 'n' the variable 'a' & array 'b' remains the same or we can say that they will be take time & space complexity equal as it is in the start.

So, its time & space complexity is O(1) i.e. constant because on changing the value of 'n' there is no change in variable & array.

Eg:-
```
int n; cin >> n;
int *b = new int[n]; // Dynamically allocated
for (int i = 0; i < n; i++) { // Print array 'b'
    cout << b[i];
}
```

Suppose,
n = 2, → b[0], b[1]
n = 4 → b[0], b[1], b[2], b[3]

So, here space complexity is changing while change the value of 'n'.

Date: .....
Page: .....

Space complexity would be linear here (i.e. O(n))

→ Unit To Represent complexity

1) Big-Oh (O) → upper Bound (worst-to-worst case)
2) Big-Omega (Ω) → Lower Bound (Best case)
3) Theta (Θ) → Average Bound (Average case)

Mostly, complexity is represented by Big-Oh.

→ Big-Oh (O) : complexities.

i) Constant Time = O(1)
ii) Linear " = O(n)
iii) Logarithmic " = O(logN)
iv) Quadratic " = O(N²)
v) Cubic " = O(N³)

→ Least To most complexities:

$1 < \log n < \sqrt{n} < n < n\log n < n^2 < n^3 < \cdots < 2^n < 3^n < n! < n^n$

↳ Least complex ... most complex

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

→ Lower Bound = $1 < \log n < \sqrt{n} < n$

→ Average " = n

→ Upper " = $n < n\log n < n^2 < n^3 < \cdots < 2^n < 3^n < \cdots < n^n$

→ Big-Oh

The function $f(n) = O(g(n))$ if & only, if there exist positive constant c & n_0.

such that, $f(n) \leq c * g(n)$ for all $n \geq n_0$ (c = constant)

Date: .....
Page: .....

Eg:- $f(n) = 2n+3$

$= \underbrace{2n+3}_{f(n)} \leq \underbrace{10}_{c}\underbrace{n}_{g(n)}, \quad n \geq 1$

$\therefore f(n) = O(n)$

Eg:- $f(n) = 2n+3$

$= 2n+3 \leq 2n^2+3n^2$

$= \underbrace{2n+3}_{f(n)} \leq \underbrace{5}_{c}\underbrace{n^2}_{g(n)}$

$\therefore f(n) = O(n^2)$

→ **Omega.**

The function, $f(n) = \Omega(g(n))$, if & only if there exist positive constant, c & n_0

such that, $f(n) \geq c * g(n)$ for all $n \geq n_0$

Eg:- $f(n) = 2n+3$

$= \underbrace{2n+3}_{f(n)} \geq \underbrace{1}_{c}\underbrace{*n}_{g(n)}$ for all $n \geq 1$

$\therefore f(n) = \Omega(n)$

Eg:- $f(n) = 2n+3$

$= \underbrace{2n+3}_{f(n)} \geq \underbrace{1}_{c} * \underbrace{\log n}_{g(n)}, \quad n \geq 1.$

$\therefore f(n) = \Omega(\log n).$

→ **Theta.**

The function, $f(n) = \Theta(g(n))$, if & only if there exist positive constant, c_1, c_2 & n_0

Such that, $c_1 * g(n) \leq f(n) \leq c_2 * g(n)$

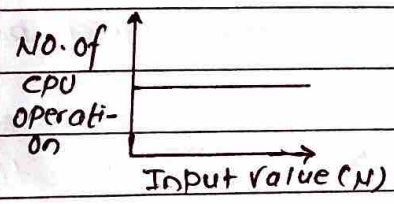
Eg:- $f(n) = 2n+3$

$= \underbrace{1}_{c_1}\underbrace{*n}_{g(n)} \leq \underbrace{2n+3}_{f(n)} \leq \underbrace{5}_{c_2}\underbrace{*n}_{g(n)}$

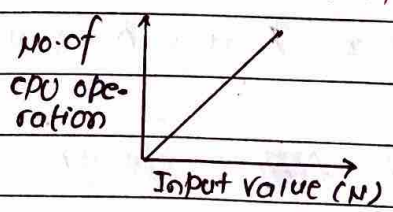
$\therefore f(n) = \Theta(n).$

→ We can't use any other things except 'n' both side, Because theta notation is average notation. so we only use 'n'.

→ constant Time Graph



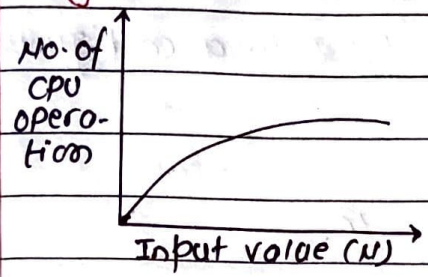
→ Linear Time Graph



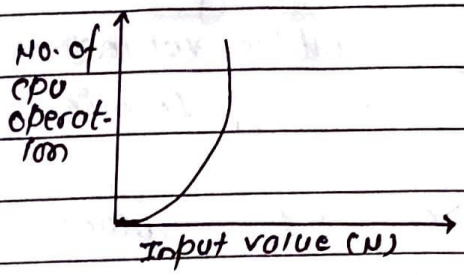
Date: .....

Page: .....

→ Logarithmic Time Graph



→ Quadratic Time Graph



NOTE :- For Nested for loops, we multiply.

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

complexity will be " $n \times n = n^2 \Rightarrow O(n^2)$

For two or more different loops, we add

Eg:-
```
for (int i = 0; i<n; i++) {....}
for (int j = 0; j<n; j++) {....}
```

complexity will be $n+n = 2n \Rightarrow O(n)$

↳ constant, ignore it.

→ Examples

1) $f(n) = 2n^2 + 3n = O(2n^2) \Rightarrow O(n^2)$

↳ constant always ignore

2) $f(n) = 4n^4 + 3n^3 = O(4n^4) \Rightarrow O(n^4)$

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

4) $f(n) = 200 \Rightarrow O(1)$

5) $f(n) = \frac{N}{4} \Rightarrow O(N)$

Big-oh always looks for upper Bound!