TIME & SPACE COMPLEXITY

Time Complexity :-
Amount of Time taken up by an algorithm / code as function of input size.
NOT the actual time taken.

★ Time for linear search in worst case will remain same even if our array is sorted

* Big O Notation :-
↳ upper bound → O( )

★ we always try to find worst case complexity.

time ⇒ $an^2 + bn + c$
$n^2 + n + 1$

This is theoretical way for calculating TC.

$TC = O(n^2)$

Steps :-
① Ignore constants
② Largest term

time → $an^3 + b\log n + c$
$n^3 + \log n + 1$

$TC = O(n^3)$

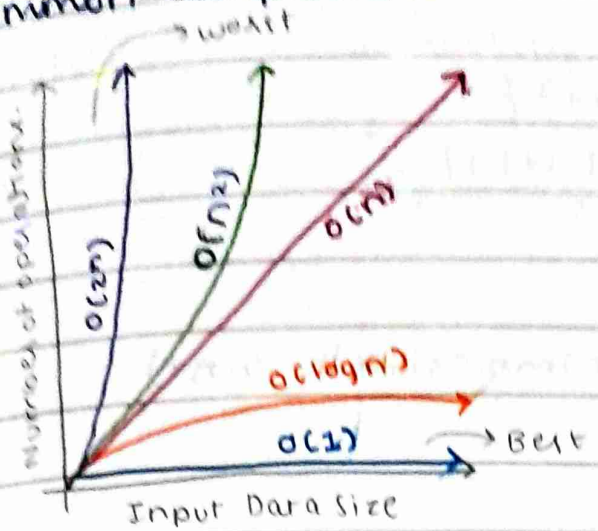
Big Omega Notation :- (Ω)
↳ lower bound → Best case TC.

$\Omega(n^2)$
$\Omega(n)$ ✗
$\Omega(1)$
n^3
n^4
n^5 ✓

Big Theta (Θ) :-
avg cases

$O(n^2)$ $\Omega(n^2)$
↳ LB = UB
$\Theta(n^2)$

Common Complexities:-



Space Complexity:-

→ memory / space → heap → objects
→ stack → functions.

input space + auxillary space.
↳ temporary / extra.

* While calculating space complexity we only consider auxillary space.

Time Complexity of loops:-

1. Simple loop.

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

Time complexity :- O(n)

2. Nested loop 1.

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

Time complexity :- $O(n^2)$

Nested loop 2.

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

Time complexity :- $O(n^2)$

Nested loop 3

```
//some k<n
for (int i=0; i<n; i=i+k) {
    for (int j=i+1; j<=k; j++) {
        ____
        ____
    }
}
```

Time complexity :- $O(n)$

	Linear search	Binary search
$N = 1000$	1000	$\log n$ ($\log 10^3$)
$N = 10^5$	10^5	≈ 10
$N = 10^9$	10^9	≈ 30

Recursive Algorithms.

→ linear $f(n) = \square + f(n-1)$

→ D & C merge = $f(n) = f(n/2) + f(n/2) + \square$

1. Total work done = (no of calls * work in each call)
2. Recurrence Equation

Space complexity = (max depth * memory in each call)

description	order of growth	typical code framework	description	example
constant	1	`a = b + c;`	*statement*	*add two numbers*
logarithmic	$\log N$	[*see page 47*]	*divide in half*	*binary search*

linear	N	`double max = a[0];`
		`for (int i = 1; i < N; i++)`
		`  if (a[i] > max) max = a[i];`
			loop	*find the maximum*

linearithmic	$N \log N$	[*see* ALGORITHM 2.4]	*divide and conquer*	*mergesort*

quadratic	N^2	`for (int i = 0; i < N; i++)`
		`  for (int j = i+1; j < N; j++)`
		`  if (a[i] + a[j] == 0)`
		`  cnt++;`
			double loop	*check all pairs*

cubic	N^3	`for (int i = 0; i < N; i++)`
		`  for (int j = i+1; j < N; j++)`
		`  for (int k = j+1; k < N; k++)`
		`  if (a[i] + a[j] + a[k] == 0)`
		`  cnt++;`
			triple loop	*check all triples*

exponential	2^N	[*see* CHAPTER 6]	*exhasutive search*	*check all subsets*