Time & Space Complexity

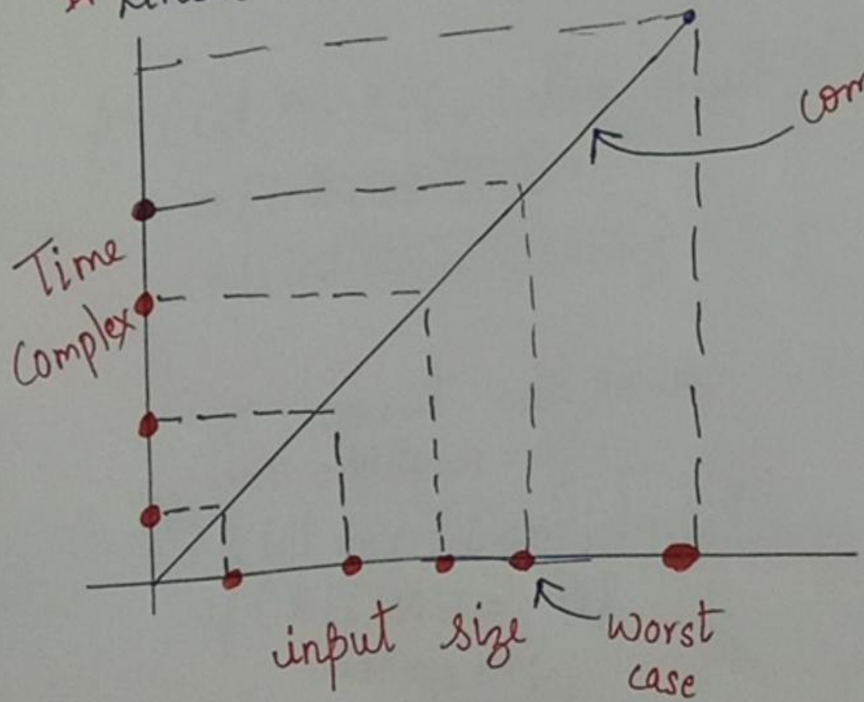
Order Complexity Analysis:

⇒ Amount of space or 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 was sorted

★ Linear Search



complexity ($y = ax + b$)

$time = an + b$

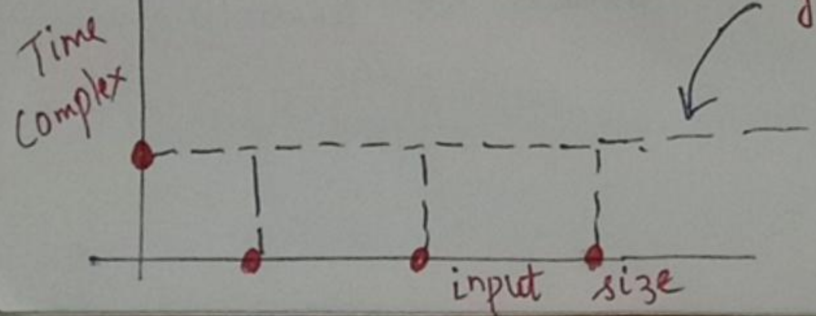
$T.C = O(n)$

$T \propto n$ (Linear)

for sorted elements:

$y =$ constant value

$T.C = O(1)$



Big O Notation :-

Big O → upper bound

$\boxed{\text{NOTE}}$ - We always try to find worst case complexity.

* How to find time ?

1. Ignore constants
2. largest term

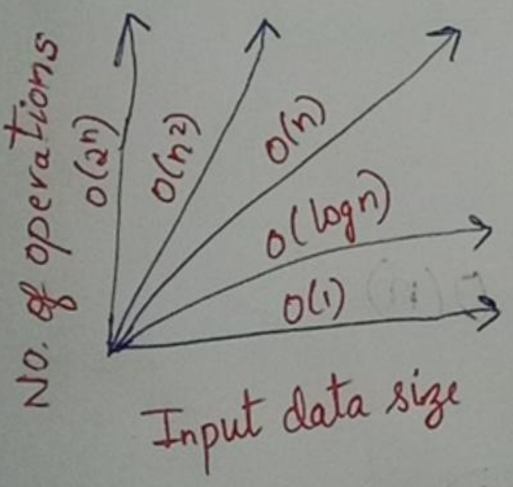
Big Omega Notation :-

Big Omega → lower bound → best case TC

Big Theta (θ) :-

Big Theta → avg bound

Common Complexities :



Space Complexity → memory/space → heap → objects

memory/space → stack → funⁿ calls

[input space + auxiliary space]

In terms of space Quick Sort > merge sort
$O(1)$ $O(n)$

Theoretical Analysis:

- Loop based examples
- Sorting / Searching
- Recursive Problems

Simple Loop $\rightarrow O(n)$

Nested loop:

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

$\} \rightarrow O(n^2)$

Nested Loop 2:

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

$\} \rightarrow O(n^2)$

Nested Loop 3:

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

$\} \; O(n)$

Bubble Sort $\rightarrow$ Worst $\rightarrow O(n^2)$

Bubble Sort $\rightarrow$ Best $\rightarrow O(n)$

optimized bubble Sort

```
public static void modifiedBubbleSort(int arr[]){
  for(int i=0; i< arr.length -1; i++){
    boolean swapped = false;
    for(int j=0; j< arr.length-1 -turn; j++){
      if(arr[j] > arr[j+1]){
        //swap
        int temp= arr[j];
        arr[j]= arr[j+1];
        arr[j+1]= temp;
        swapped = true;
      }
    }
    if (swapped == false){
      break;
    }
  }
}
```

[Binary Search → O(log n)]

Recursive Algorithms → Linear
→ Divide & Conquer

① Total work done = (no. of calls * work in each call)
↳ Linear

② Recurrence Equation

⇒ $\boxed{\text{Space Complexity} = (\text{max depth} * \text{memory in each call})}$

Recursion:

☆ Factorial ⇒ T.C & S.C → O(n)

☆ Sum of n ⇒ T.C & S.C → O(n)

☆ Fibonacci ⇒ T.C = 2^n , SC = O(n)

☆ MergeSort ⇒ {T.C & S.C → O(n)} → mergeSort funn

↳ T.C → nlogn , S.C → O(n)

☆ Power funn | ⇒ T.C → O(logn)
(optimised)