

## What is Algorithm Analysis ?

Simple Definition –

- Algorithm analysis is a **study** to provides theoretical **estimation** for the required **resources** of an algorithm to solve a specific computational problem. i.e. calculating **efficiency**.

Generally, the **efficiency** an algorithm is related to the input length (number of steps), known as **time complexity**, or volume of memory, known as **space complexity**.

## Why do we need Algorithm Analysis ?

- Knowing efficiency of an algorithm is very vital on **mission critical** tasks.*
- Generally there are **multiple** approaches/method/algorithms to solve one problem statement. Algorithm analysis is performed to figure out which is the **better/optimum** approaches/method/algorithms out of the options.*

## What does a **BETTER** Algorithm mean ?

- Faster ? (Less execution time) – **Time Complexity**
- Less Memory ? – **Space Complexity**
- Easy to read ?
- Less Line of Code ?
- Less Hw/Sw needs ?

\*Note: Algorithm Analysis **does not give you accurate/exact values**(time, space etc), however it gives **estimates** which can be used to study the **behavior** of the algorithm.

## What is **Asymptotic** Algorithm Analysis ?

- Definition: In mathematical analysis, **asymptotic** analysis of algorithm is a method of defining the **mathematical boundaries** of its **run-time** performance.
- Using the asymptotic analysis, we can easily estimate about the average case, best case and worst case scenario of an algorithm.

Simple words: It is used to mathematically calculate the running time of any operation inside an algorithm.

Asymptotic Algorithm analysis is to estimate the **time complexity** function for arbitrarily large input.

Time Complexity : is a computational way to show **how(behavior)** runtime of a program increases as the size of its input increases.

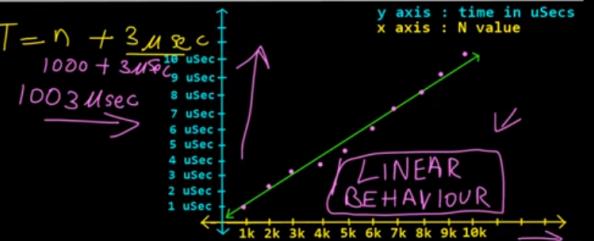
### Problem Statement : Write an Algorithm/program to find the SUM of N numbers (0 - N)

Algorithm 1

```

function sumOfNumbers(N)
{
    1.sum = 0 → 1 uSec
    → 2.for(i = 0 to N) → 1 uSec
    {
        → 3.sum = sum + i → n uSec
    }
    → 4.print(sum) → 1 uSec
}

```

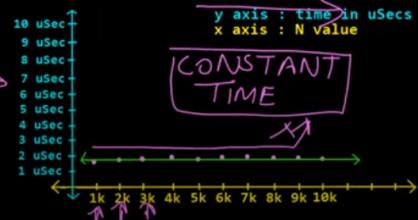


Algorithm 2

```

N=1000 → 2 uSec
function sumOfNumbers(N)
{
    1.sum = (N*(N+1))/2
    2.print(sum)
}

```



### Big O Notation

>> The notation  $O(n)$  is the formal/mathematical way to express the upper bound (worst case) of an algorithm's running time.

>> It measures the worst case time complexity or the longest amount of time an algorithm can possibly take to complete.

Following is a list of some common asymptotic notations –

>> constant time -  $O(1)$     >> linear -  $O(n)$   
 >> logarithmic -  $O(\log n)$     >> quadratic -  $O(n^2)$

>> cubic -  $O(n^3)$

Algorithm 1  $O(n)$

```

function sumOfNumbers(N)
{
    1.sum = 0 →
    → 2.for(i = 0 to N) →
    {
        3.sum = sum + i
    }
    → 4.print(sum) →
}

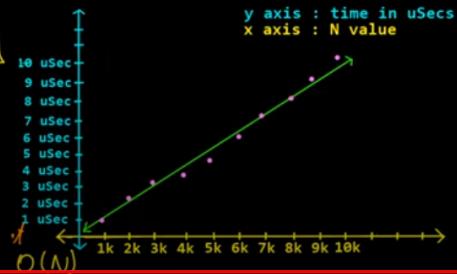
```

$$T = \boxed{n^2} + \boxed{2n}$$

$$= O(n^2)$$

>> How to find out the Big O notation ?

- 1. Find the fastest growing variable term
- 2. Eliminate the co-efficients/constant terms



### Space Complexity

# What is Space complexity?

- **Definition:** The space complexity of an algorithm or a computer program is the amount of memory space required to solve an instance of the computational problem as a function of the size of the input.
- **Simple words :** It is the memory required by an algorithm to execute a program and produce output.
- Similar to time complexity, Space complexity is often expressed asymptotically in big O notation, such as  $O(n)$ ,  $O(n\log(n))$ ,  $O(n^2)$  etc., where  $n$  is the input size in units of bits needed to represent the input.

$$\text{Space Complexity} = \text{Input Size} + \text{Auxillary Space}$$

Algorithm 1 - Addition of 2 numbers  
→ function add( $n_1, n_2$ )  
    { 2 DO  
         $\sum = n_1 + n_2$  / 10, 90  
        → return sum  
    }

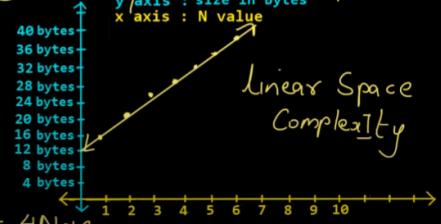
$n_1 \rightarrow 4 \text{ bytes } O(1)$   
 $n_2 \rightarrow 4 \text{ bytes}$   
Sum  $\rightarrow 4 \text{ bytes}$   
Aux Sp  $\rightarrow 4 \text{ bytes}$   
Total  $\rightarrow 16 \text{ bytes} = C$



Algorithm 2 - Sum of all elements in array  
function sumOfNumbers(arr[], N)

{  
    1.sum = 0  
    2.for(i = 0 to N)  
    {  
        3.sum = sum + arr[i]  
    }  
    4.print(sum)  
}

$arr \rightarrow N \times 4 \text{ bytes}$   
Sum  $\rightarrow 4 \text{ bytes}$   
 $i \rightarrow 4 \text{ bytes } O(N)$   
Aux  $\rightarrow 4 \text{ bytes}$   
Total  $= 4N + 12 \text{ bytes} = 4N + C$

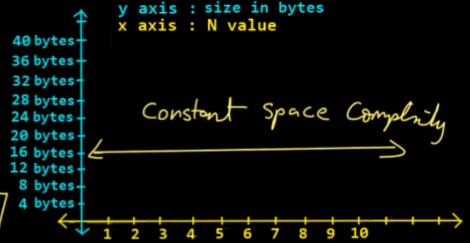


$$\rightarrow \text{Space Complexity} = \text{Input Size} + \text{Auxiliary Space}$$

**Algorithm 3 - Factorial of a number (iterative)**

```
int fact = 1; n=5
for (int i = 1; i <= n; ++i)    fact → 4 bytes
{
    fact *= i;                n → 4 byte
                                i → 4 byte. O(1)
}
return fact;                  Aux → 4 bytes
```

$$S = 12 \text{ bytes} + 4 \text{ bytes} = 16 \text{ bytes}$$


**Algorithm 4 - Factorial of a number (recursive)**

```
factorial2(n)=
{
    if(n<=1)    n→4byte   fn(5)
    {
        return 1;    Aux→4byte  1/20
    }
    else
    {
        return (n*factorial2(n-1));
    }
}
```

$= 8 \text{ bytes} \times 5 = 40 \text{ bytes}$

$$S = 4 \text{ bytes} + [n \times \text{bytes}]$$

