Steps to find Big – O

1) Find no. of iterations ✓

2) Neglect the lower order terms

3) Neglect the constant co-efficient

Quiz 1: $F(N) = 3N^2 + 6N + 10^3 \Rightarrow O(N^2)$

Quiz 2: $F(N) = 3N^2 + 6N^3 + 10^3 \Rightarrow O(N^3)$

Quiz 3: $F(N) = 3N + 6N\log N + 10^3 N^0 \Rightarrow O(N\log N)$

How to compare two algorithms.

Task: Given 10^5 integers, sort them in increasing order

Asish [SortAlgo]	Arjun [FlexAlgo]
15sec (windows 95)	10sec (Macbook Pro M1)
7sec (Macbook ProM1) (C++)	10sec (Macbook Pro M1) (Python)
7sec C++ (Hot Mountains)	5sec C++ (Antartica)

Execution time depends on lots of factors

```
for(i=1; i≤N; i++){
        print(i)
}
```

#iterations = N

No. of iterations/operations is always fixed and doesn't depend on any external factors

Sort Algo $\qquad\qquad$ FlexAlgo

$100 \log N$ $\qquad\qquad$ $\frac{N}{10}$

N: [0, 3550] : Flex Algo has lesser iterations

N: [3550, ∞] : Sort Algo has lesser iterations

$$\Downarrow$$

$$\boxed{\text{Sort Algo}}$$

Asympotic Analyis of Algorthms

→ Observing performane of algorithms at higher value of N (Input Size)

1) Big – O
2) Omega
3) Theta

Steps to find Big-O → [To compare algorithms]

1) Find no. of iterations
2) Neglect the lower order terms
3) Neglect the constant co-efficient

1) Find no. of iterations

It is always fixed and doesn't depend on external factors

2) Neglect the lower order terms

$$f(N) = \underline{N^2} + \underline{10N}$$

$N = 10$ $= 10^2 + 10 \cdot 10 = 200$

N^2	$10N$	Contribution of lower order term
100	100	$\frac{100}{200} \times 100 = 50\%$

$N = 100$

N^2	$10N$	Contribution of lower order term
10^4	10^3	$\frac{10^3}{10^3+10^4} \times 100$

$$= \frac{10^3}{10^3(1+10)} \times 100$$

$$= \frac{100}{11} \approx 9.9\%$$

$N = 10^4$

N^2	$10N$	Contribution of lower order term
10^8	10^5	$\frac{10^5}{10^5+10^8} \times 100$

$$= \frac{10^5}{10^5(1+1000)} \times 100 = \frac{100}{1000} \approx 0.1\%$$

As N ↑, contribution of lower order term decreases. Hence, neglect the lower order terms.

3) Neglect the constant co-efficient

Sort Algo	Flex Algo	For larger N
N	10 logN	Flex Algo
N	100 logN	Flex Algo
N	1000 logN	Flex Algo
$\frac{N^2}{10}$	100N	Flex Algo

$N = 2^{32}$

sort Algo ⇒ 2^{32} items $= 10^9$

flex Algo ⇒ $10 \cdot \log_2 2^{32} = 10 \times 32 = 320$

$N = 10^6$

Sort Algo : $\frac{10^{12}}{10} = 10^{11}$

Flex Alg : $100 \times 10^6 = 10^8$

Issues with Big-O

Issue1:

	Sort Algo	Flex Algo	
	100N	N^2	
	$O(N)$	$O(N^2)$	
$N=10$	1000	100	Flex Algo is better
$N=50$	5000	2500	Flex Algo is better
$N=100$	10^4	10^4	Both are same
$N>100$	100×101	101×101	Sort Algo is better

$O(N)$ is better than $O(N^2)$ only after a certain threshold

Issue2:

Sort Algo	Flex Algo
$10N^2 + 5N$	$5N^2 + 100N$
$O(N^2)$	$O(N^2)$

Worst case scenario

```
boolean func (int arr[], int N, int K){
        for (i=0; i<N; i++){
              if(arr[i] == K){
                 return True;
              }
        }
        return False;
}
```

#iterations?

Best case

A = 5 4 6 1 7 K = 5 [1 iteration]

A = 5 4 6 1 7 K = 7 [N iterations]

A = 5 4 6 1 7 K = 100 [N iterations]

Worst Case

#iterations in worst case => N

8 mins

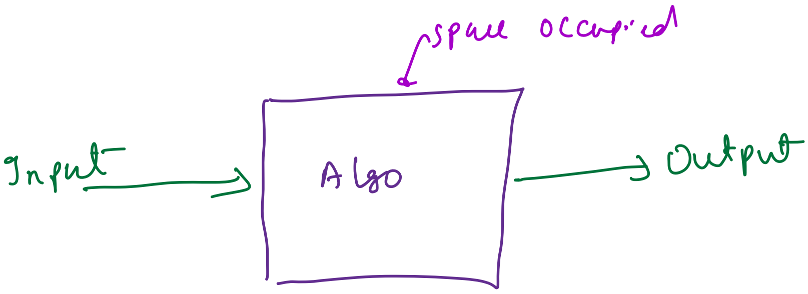
8 : 2

Time Complexity

1) Find no. of iterations [Function of N]
2) Neglect lower order terms
3) Neglect constant co-efficient

Space Complexity

Extra space used apart from input and output



```
void function (int N) {
    int [10] arr;      ⇒   10x4 = 40 bytes
    float f;           ->   4 bytes
    int [N] arr2;      ->   4N bytes
}
```

Space Complexity

1) Find the memory/space occupied in bytes
2) Neglect lower order terms
3) Neglect constant co-efficient

Total = $4N + 44$ bytes $\Rightarrow \boxed{O(N)}$

$N = 10, N = 100, N = 10^{100}$

```
void function(int N) {
    int[10] arr;     =>   40
    float f;         =>   4
    int x;           =>   4
}
```

Space = 48 bytes $= 48 \cdot N^0$

$= O(1)$ space complexity

```
void function(int N) {
    int[10] arr;        =>    40 bytes
    float f;            =>    4 bytes
    int[N] arr2;        =>    4N bytes
    int[N][N] arr3;     =>    4N^2 bytes
}
```

$$\text{Space} = 4N^2 + 4N + 44$$

$$\Downarrow$$

$$O(N^2)$$

Eg:

```
void    function (int   N) {

        int [] arr1  =   new  int [n];  =>   4N
        int [] arr2  =   new  int [n];  =>   4N

}
```

Space = 8N => O(N)

Ex:

```
int   sum  (int  arr[],  int  N) {
      int      ans = 0;
      for (int   i=0;  i<N;  i++) {
              ans  +=  arr[i];
      }
      return  ans;
}
```

i => 4bytes => $\boxed{O(1)}$

Time Limit exceed error

Limit: 1-2 sec

1sec ⇒ 10^8 iterations X

	$O(N)$	$O(N^2)$	$O(N^3)$
$N=10$	✓	✓	✓
$N=10^3$	✓	✓	✗
$N=10^6$	✓	✗	✗
$N=10^9$	✗	✗	✗

$(10^3)^2 = 10^6$

$(10^3)^3 = 10^9$ ⇒ TLE

$O(N^3)$ is $N = 10^3$

$N^3 = ((10^3)^3 = 10^9$

$N \leq 10^6$

$N \leq 10^3$ } ⇒ $O(N^2)$

$(10^6)^2 = 10^{12}$