Time Complexity (it is always an approximation)

Rule :- all constant values with arthematic operators are going to ignored.

Ex :-

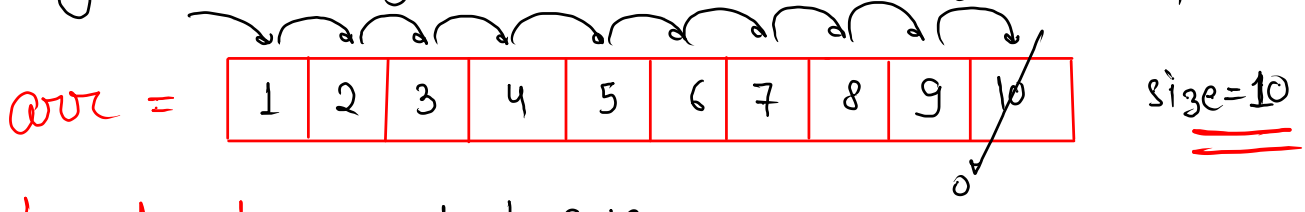
no. of operations		T.C
$n+7$	$\simeq$	$O(n)$
$4*(n^3+3)$	$\simeq$	$O(n^3)$
$4*((n/2)+3)$	$\simeq$	$O(n)$
$n/7$	$\simeq$	$O(n)$
(n^3+n^2+n)	$\simeq$	$O(n^3)$

⇒ Notations for T.C

O(N)

- worst case :- when code take most time (capital O)
- average case :- when code take average time
- best case :- when code take least time

Ex:- given array, check if a target is present or not.



target = 1 ⟶ best case

target = 10 ⟶ worst case

Operation = $n * m$

T.C = $O(n * m)$

Ex:-

$n =$ ____ , $m =$ ____

```
for( int i=0; i < n ; i++) {
    for( int j=0; j < m ; j++) {
        Syso( "Hello" );
    }
}
```

Ex:-

```
int n = ____

for(int i=0; i<n; i++){
    Syso("abc");
}

for(int i=0; i<n; i++){
    Syso("efg");
}
```

operations = 2 * n

T.C ≅ O(n)

Ex:-

```
int n = 5
int m = 10

for(int i=0; i<n; i++){
    Syso("abc");
}

for(int i=0; i<m; i++){
    Syso("efg");
}
```

Operations = n+m

T.C ≅ O(n+m)

Ex:-

n = ____

```
for( int i=0; i < n ; i++){
    for( int j=0; j < n ; j++){
        Syso("Hello");
    }
}
```

Operations = $n^2 + n$

$T.C = O(n^2+n)$

$\simeq O(n^2)$

```
for( int i=0; i<n; i++){
    Syso("abc");
}
```

Ex:-

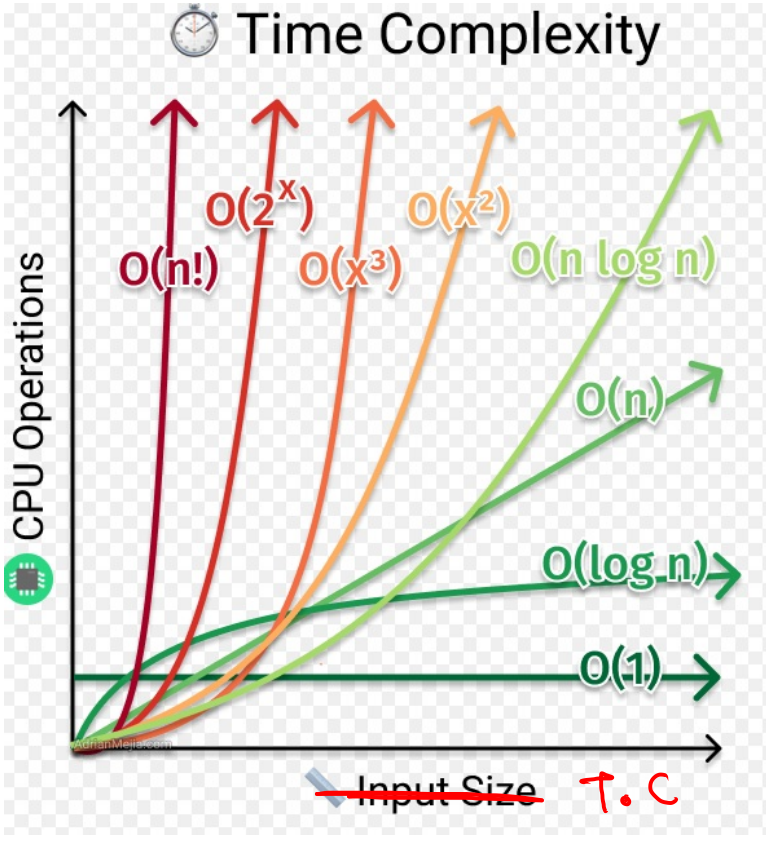
n = ____, m = ____, p = ____

```
for( int i=0; i < n ; i++) {
    for( int j=0; j < m ; j++) {
        for( int k=0; k< p ; k++) {
            Syso( "Hello" );
        }
    }
}
```

Operations :- n*m*p

T.C = O(n*m*p)

	$O(n)$	$O(n^2)$	$O(n^3)$
$n = 1$	1	1	1
$n = 2$	2	4	8
$n = 3$	3	9	27
$n = 4$	4	16	64
⋮	⋮	⋮	⋮
$n = 10^5$	10^5	10^{10}	10^{15}
	(0.5 - 1 sec)		(month)



worst to best

- O(n!)
- O(2^n)
- O(n^3)
- ✓✓ O(n^2)
- O(n log(n))
- ✓✓ O(n)
- O(log(n))
- ✓✓ O(1)

better

Space Complexity

↳ amount of space consumed

↳ no. of variables

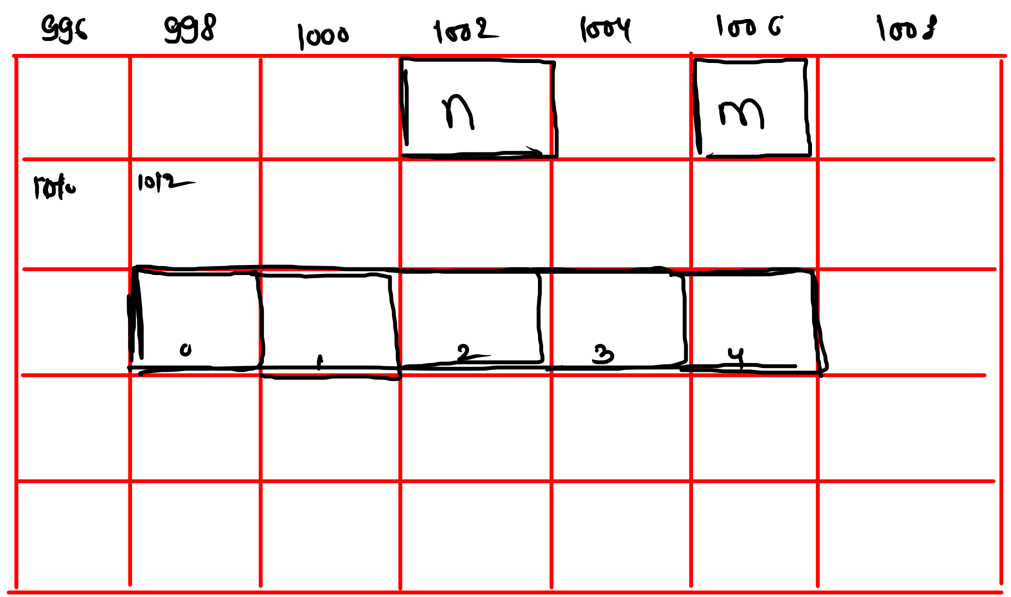
```
main(){
        Scanner ____ ____ ____ ____ ____
        int n= _____        //1        O(2)
        int m= _____        //1        ≅ O(1)

        Syso(n+m);

}
```

```
main() {
    Scanner ___
    int n= ___
    int m= ___
    int [] arr = new int[n];
}
```

⇒ $n+2$

S.C ⇒ $O(n+2)$

S.C ⇒ $O(n)$



```
main() {
    Scanner scn = ------;
1 → int n = ------
1 → int m = ------
n → int [] arr1 = new int [n];
m → int [] arr2 = new int [m];

}
```

allocation = m+n+2

S.C = O(m+n)

```java
public class Solution {

    public static void main(String[] args) {
        Scanner scn = new Scanner(System.in);
        int n = scn.nextInt();
        int[] arr = new int[n];
        for (int i = 0; i < n; i++) {
            arr[i] = scn.nextInt();
        }

        System.out.println(findDuplicates(arr));
    }
    public static boolean findDuplicates(int[] arr) {
        int n = arr.length;
        for (int i = 0; i < n; i++) {
            for (int j = i + 1; j < n; j++) {
                if ( arr[i] == arr[j] ) { // myself == other
                    return true;
                }
            }
        }
        return false;
    }
}
```

Op = $n + n^2$

T.C = $O(n^2)$

S.C = $O(n)$

$O(1)$

```
main() {
    1 → int n = 5, 500000
    1 → int m =
    5 → int[] arr = new int[5];
}
```

allocation = 7

S.C = O(1)

constant