Time Complexity (It's always an approximation)

Rule :- all constant values with airthematic operators are always going to be ignored.

Ex :-

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

Rule:- In an expression, T.C will always pick variable of highest power.

$\Rightarrow 2n + 3n - n$

$\Rightarrow 4n \simeq O(n)$

⇒ Notations of T.C

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

Ques Given array, check if a target is present in array or not.

arr =

1	2	3	4	5	6	7	8	9	10

target = 1 ⟶ 1 operation needed (best case)

target = 10 ⟶ 10 operations needed (worst case)

Ex:-

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

T.C = O(n)

Ex:-

```
n = scn.nextInt(); //3
m = scn.nextInt(); //4

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

no. of operations = $n * m$

T.C $\simeq O(n * m)$

Ex:-

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

no. of operations = n^2

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

Ex:-

$n = 3, \quad m = 5$

```
for (int i=0; i<n; i++){
    Syso("Hello1");
}
```

// n times

```
for (int i=0; i<m; i++){
    Syso("Hello2");
}
```

// m times

no. of operations = $m+n$

$T.C \cong O(m+n)$

Ex:-

```
for(int i=0; i<n; i++){
    Syso("Hello1");
}
```

// n times

```
for(int i=0; i<n; i++){
    Syso("Hello2");
}
```

// n times

no. of operations = $n + n = 2 * n$

$T.C \cong O(n)$

Ex:-

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

no. of operations = $n * m * p$

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

Ex:-

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

no. of operations = n^3

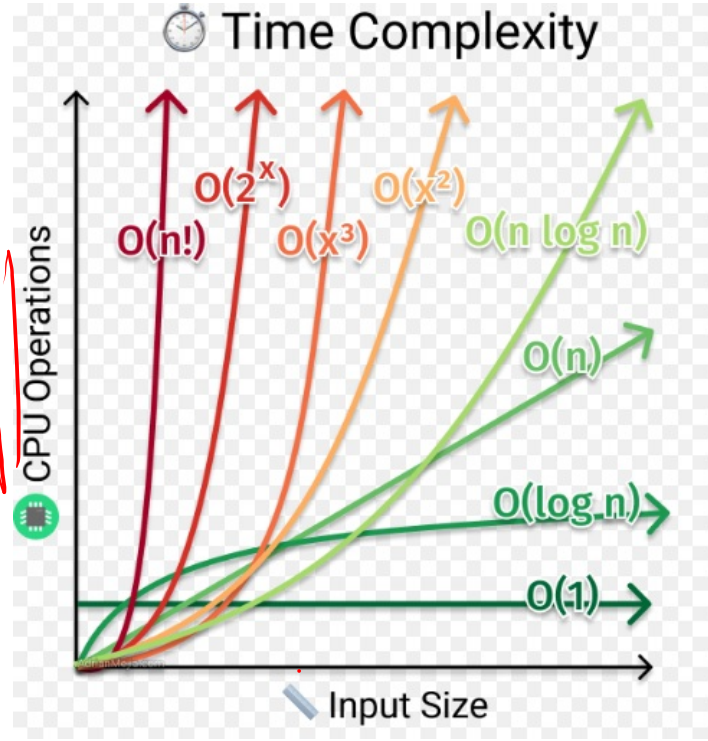
T.C = $O(n^3)$

i/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)		10 - 12 months (1 year)

best to worst

- $O(1)$
- $O(\log(n))$
- $O(n)$
- $O(n\log(n))$
- $O(n^2)$
- $O(n^3)$
- $O(2^n)$
- $O(n!)$

best to worst



Ex :-

```
for( int  i=0; i<n ; i++) {
    for( int j=0; j<3 ; j++){
        Syso("Hi");
    }
}
```

no. of operations :- $n * 3$

T.C = $O(n)$

⇒ Space Complexity

→ amount of space consumed

→ no. of variables

Ex:-

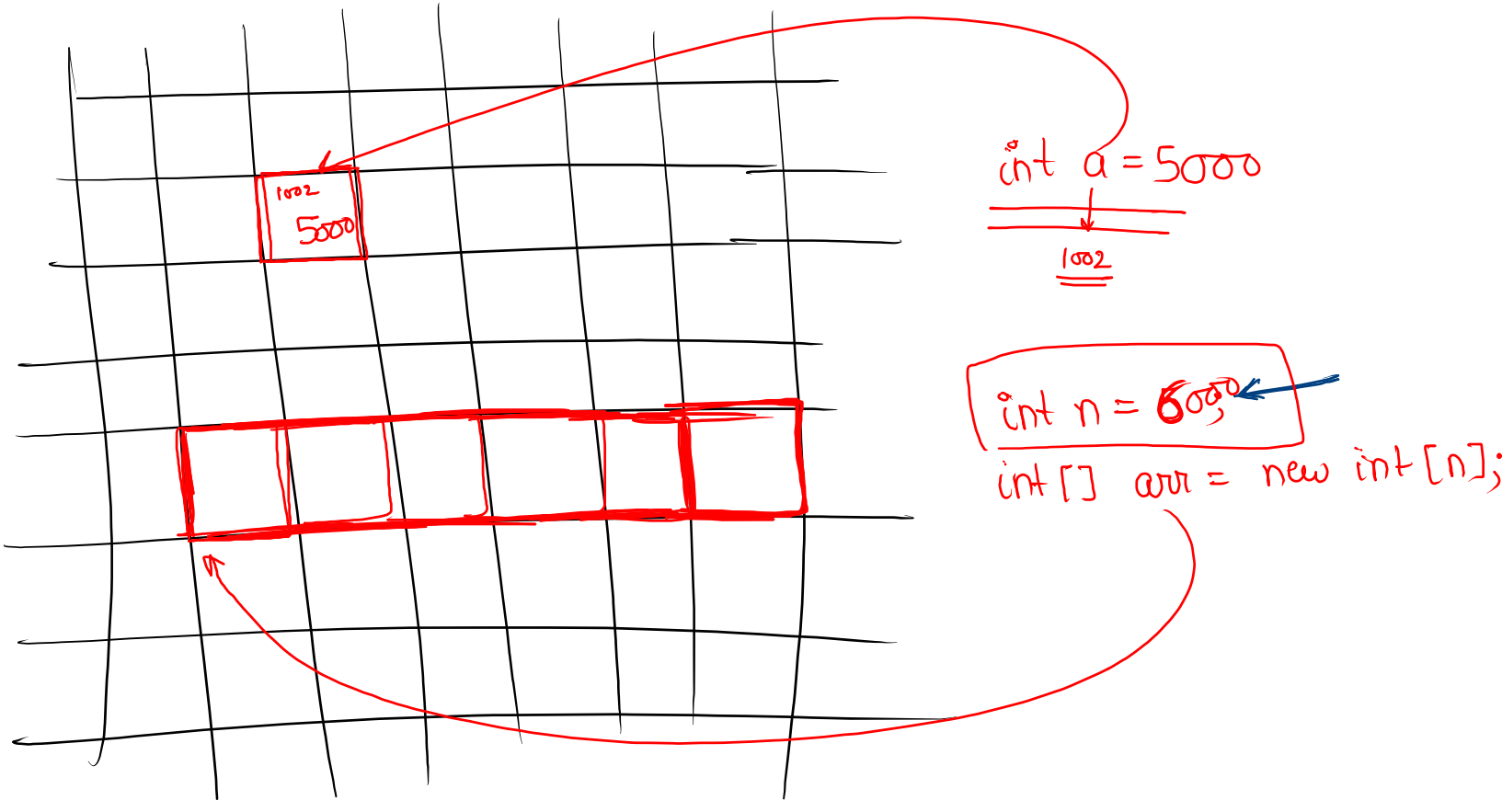
```
main (){
    Scanner ------;
    int n= -----;   // 1
    int m= -----;   // 1

    Syso(n+m);
}
```

variables = 2

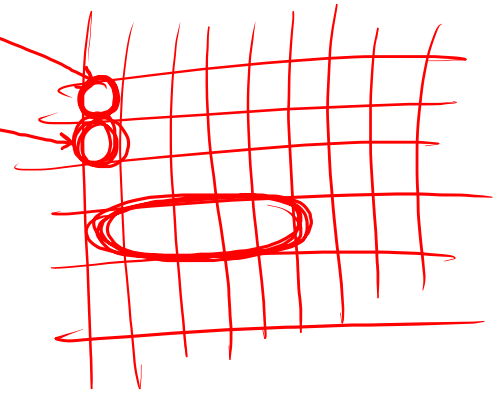
S.C = O(2)

≅ O(1)


1002
5000
int a = 5000
1002
int n = 6000;
int [] arr = new int [n];

Ex!-

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



m/m address :- n+2

S.C ≅ O(n+2)

≅ O(n)

Ex!-

```
main(){
    int n= ----;
    int m= ----;
    int[] arr = new int[n+m]
}
```

m/m address :- n+m+2

S.C ≅ O(n+m)

Note :-

S.C will always be just size of array (for module 1)

```
main() {
    int n = ____;
    String str = ____;
    int[] arr = new int[n];
}
```

S.C = O(n)

Ex:-

```java
public static void main(String[] args) {
    Scanner scn = new Scanner(System.in);
    int n = scn.nextInt();
    int[] arr1 = new int[n];
    for (int i = 0; i < n; i++) {
        arr1[i] = scn.nextInt();
    }

    int m = scn.nextInt();
    int[] arr2 = new int[m];
    for (int i = 0; i < m; i++) {
        arr2[i] = scn.nextInt();
    }

    checkDoubleOccurance(arr1, n, arr2, m);
}
public static void checkDoubleOccurance(int[] arr1, int n, int[] arr2, int m) {

    for (int i = 0; i < n; i++) {
        int count = 0;
        for (int j = 0; j < m; j++) {
            if ( arr1[i] == arr2[j] ) {
                count++;
            }
        }
        if (count == 2) {
            System.out.print( arr1[i] + " " );
        }
    }

}
```

logic

operations = n * m

T.C = O(n * m)

```java
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(trappingRainWater(arr, n));
}
public static int trappingRainWater(int[] arr, int n) {
    int result = 0;
    for (int i = 0; i < n; i++) {
        int leftMax = Integer.MIN_VALUE;
        for (int j = 0; j <= i; j++) { // including itself
            if ( arr[j] > leftMax ) {
                leftMax = arr[j];
            }
        }
        int rightMax = Integer.MIN_VALUE;
        for (int j = i; j < n; j++) {
            if ( arr[j] > rightMax ) {
                rightMax = arr[j];
            }
        }
        int ans = Math.min( leftMax, rightMax );
        int water = ans - arr[i];
        result += water;
    }
    return result;
}
```

n
i
(n-i)

operations = ~~i + (n-i)~~
= n

total operations = n * n

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