HW_Find Me 6

arr1 = [5, -3, 2, -1]

i

arr2 = [7, 5, 4, 3, -3, 3, 1, -1]

j

abs = 3

check = true

-3

code

```
for (int i = 0; i < n; i++) {
    int curr = arr1[i];
    boolean check = false;
    for (int j = 0; j < m; j++) {
        if ( Math.abs(curr) == arr2[j] ) {
            check = true;
            break;
        }
    }
    if ( check == true ) {
        System.out.print(arr1[i] + " ");
    }
}
```

Pwe

⇒ Time complexity continue

→ Notations of T.C :- O(n)

- worst case :- when code take maximum time to get executed
- average case :- when code take average time to get executed
- Best case :- when code take minimum time to get executed

Note :- Big O notation always represents worst case scenario.

Ques find the given target in array

arr = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

target = 1 ⟶ no. of operations : 1

target = 10 ⟶ no. of operations: 10

target = 5 ⟶ " " : 5

approach

```
for (int i=0; i<n; i++){
    if(arr[i] == target){
        return true;
    }
}
return false;
```

Ex 1 :-

```
public static void main(){
    int n = scn.nextInt();

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

}
```

operations: n

T.C = O(n)

Ex2

operations = n^2

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

```
public static void main(){
    int n = scn.nextInt();
    for(int i=0; i<n ; i++){
        for(int j=0; j<n ;j++){
            Syso("Hi");
        }
    }
}
```

Ex3

```
public static void main(){
    int n = scn.nextInt();
    int m = scn.nextInt();
    for(int i=0; i<n ; i++){
        for(int j=0; j<m ;j++){
            Syso("Hi");
        }
    }
}
```

4 → int n = scn.nextInt();

5 → int m = scn.nextInt();

operations = n * m

T.C = O(n*m)

Ex 4

operation = n+m

T.C = O(n+m)

```
public static void main(){
    int n = scn.nextInt(); //4
    int m = scn.nextInt(); //5
    for(int i=0; i<n ; i++){ // n times
        Syso("Hi1");
    }
    for(int j =0; j<m ,j++){ // m time
        Syso("Hi2");
    }
}
```

Ex5

```
public static void main(){
    int n = scn.nextInt();

    for(int i=0; i<n ; i++){
        Syso("Hi1");
    }
    for(int j=0; j<n ,j++){
        Syso("Hi2");
    }
}
```

Operations :- n+n

:- 2*n

T.C ⇒ O(2*n)

⇒ O(n)

Ex 6

```
public static void main(){
    int n = scn.nextInt(); //2
    int m = scn.nextInt(); //3
    int p = scn.nextInt(); //2
    for(int i=0; i<n ; i++){
        for(int j=0; j<m ;j++){
            for(int k=0; k<p ; k++){
                Syso("Hi");
            }
        }
    }
}
```

operations :- n*m*p

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

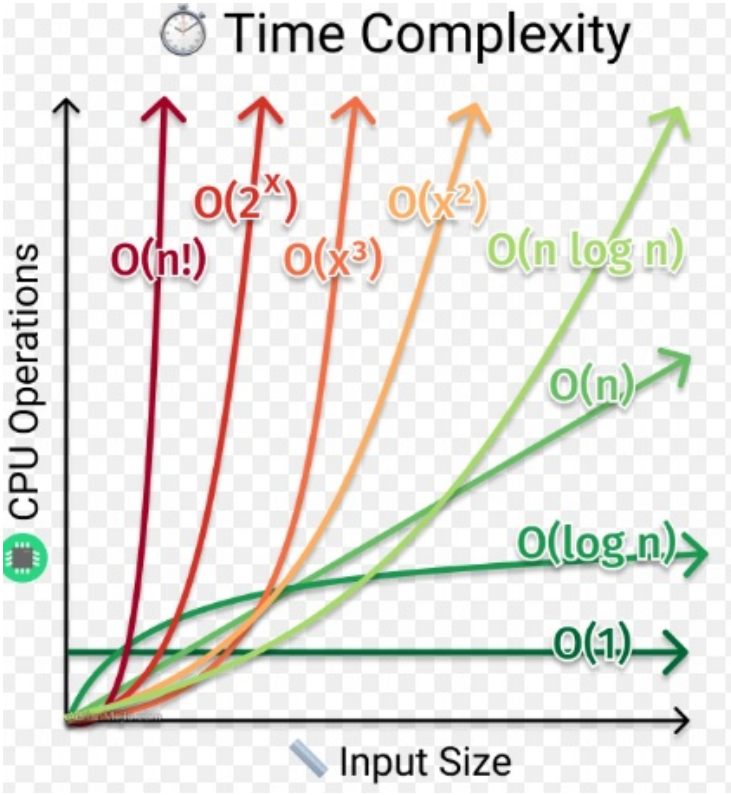
Ex7

```
public static void main(){
    int n = scn.nextInt();
    for(int i=0; i<n ; i++){
        for(int j=0; j<n ;j++){
            for(int k=0; k<n ; k++){
                Syso("Hi");
            }
        }
    }
}
```

operations :- $n * n * n$

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

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
⋮	⋮	⋮	⋮
$n=10^5$	10^5	10^{10}	10^{15}
	(0.5 - 1 sec)		(1 year)



best

$O(1)$

$O(\log(n))$

$O(n)$

$O(n\log(n))$

$O(n^2)$

$O(n^3)$

$O(2^n)$

$O(n!)$

worst

⇒ Space Complexity

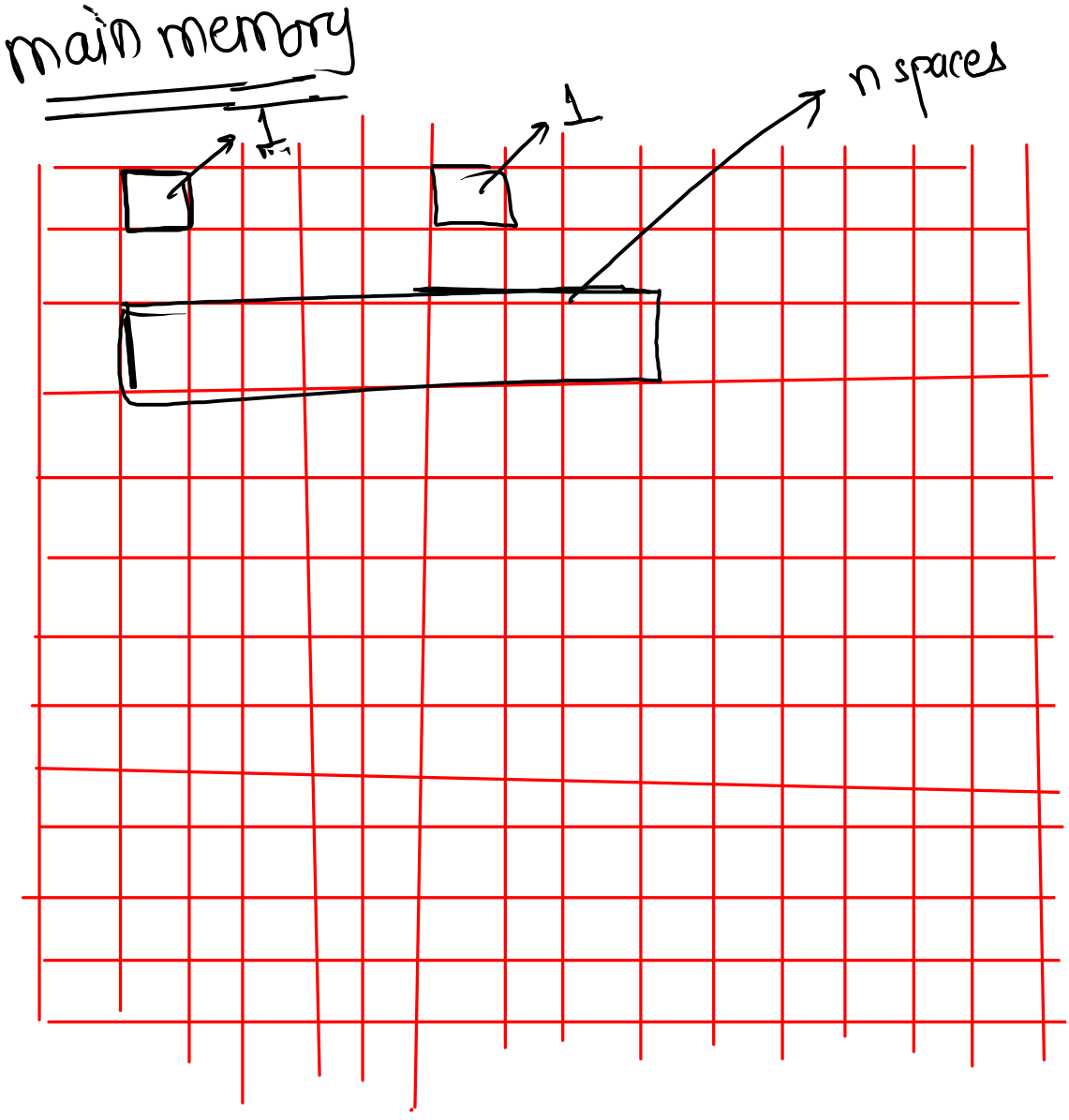
- amount of memory addresses consumed
- no. of variables

Ex:-

```
main() {
    int n = sc.nextInt();
    int m = sc.nextInt();
    int[] arr = new int[n];
}
```

m/m address = n+2

S.C = O(n)


main memory
1
1
n spaces

Ex:-

```
main() {
    int n = sc.nextInt();
    int m = sc.nextInt();
    int[] arr = new int[n*m]

}
```

m/m address = (m * n) + 2

S.C = O(m * n)

Ex:-

```
main() {
    int n = sc.nextInt();
    int m = sc.nextInt();
    int[] arr = new int[n]
    int[] arr = new int[m]
}
```

m/m addresses = m+n+2

S.C = O(m+n)

Ex:-

```
main() {
    int n = sc.nextInt();
    int m = sc.nextInt();

    int[] arr = new int[n]

    int[] arr = new int[n]

}
```

m/m addresses = $(2 * n) + 2$

S.C = $O(n)$

Note:-

Our space complexity will always be size of our array

Note:-

Our space complexity will always be size of our array

Code

```java
public static int trappingRainWater(int[] arr, int n) {
    int ans = 0;
    for (int i = 0; i < n; i++) {

        int leftMax = Integer.MIN_VALUE;
        for (int j = 0; j <= i; j++) {
            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 water = Math.min(leftMax, rightMax);
        int actualWater = water - arr[i];

        ans += actualWater;
    }
    return ans;
}
```

// i times

// (n - i) times

operation = n * n

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

$S.C = O(n)$