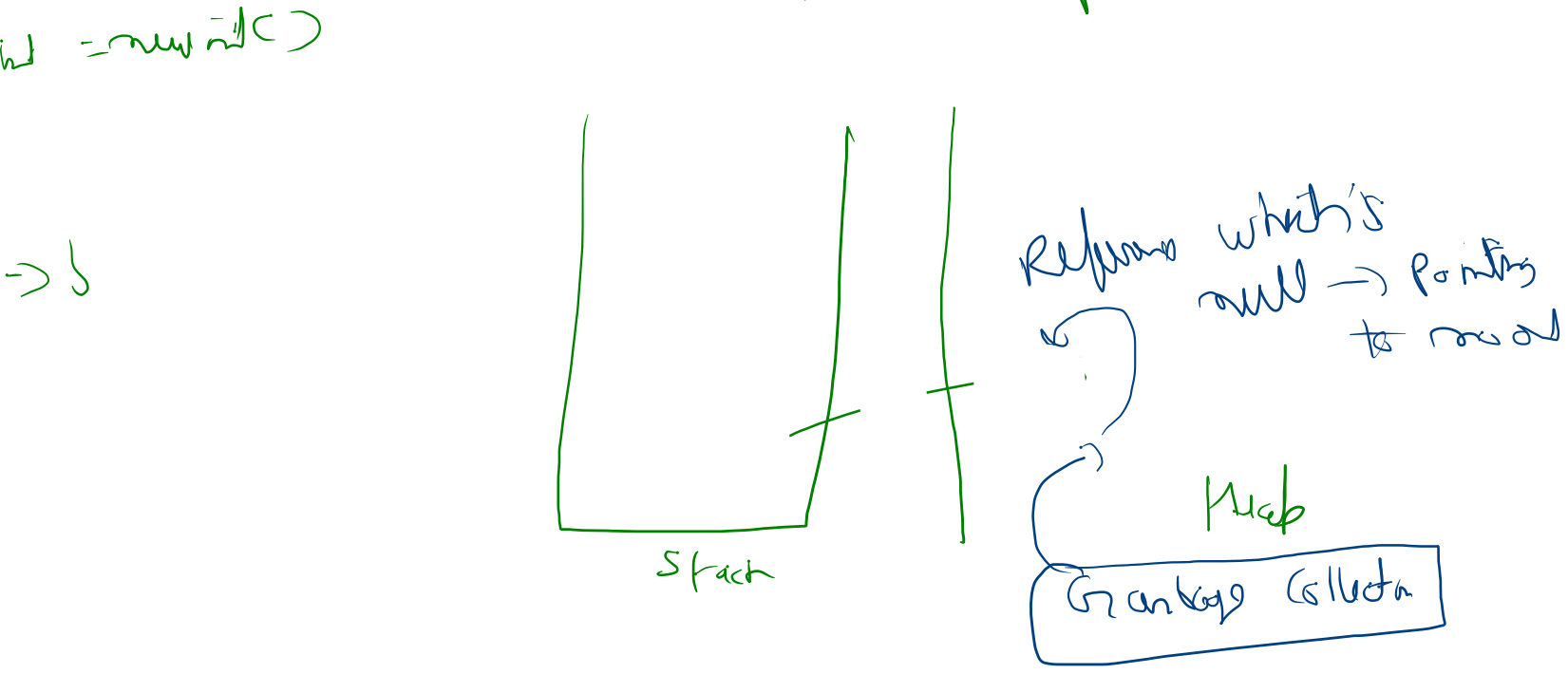
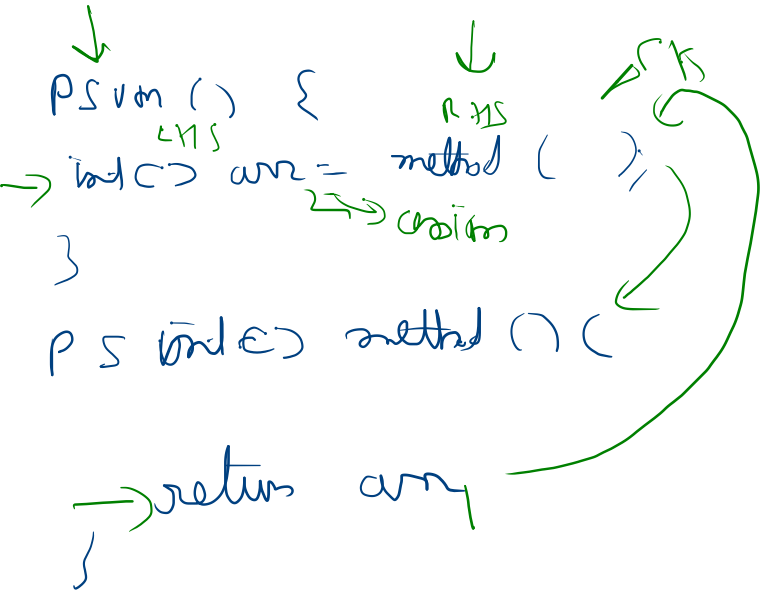
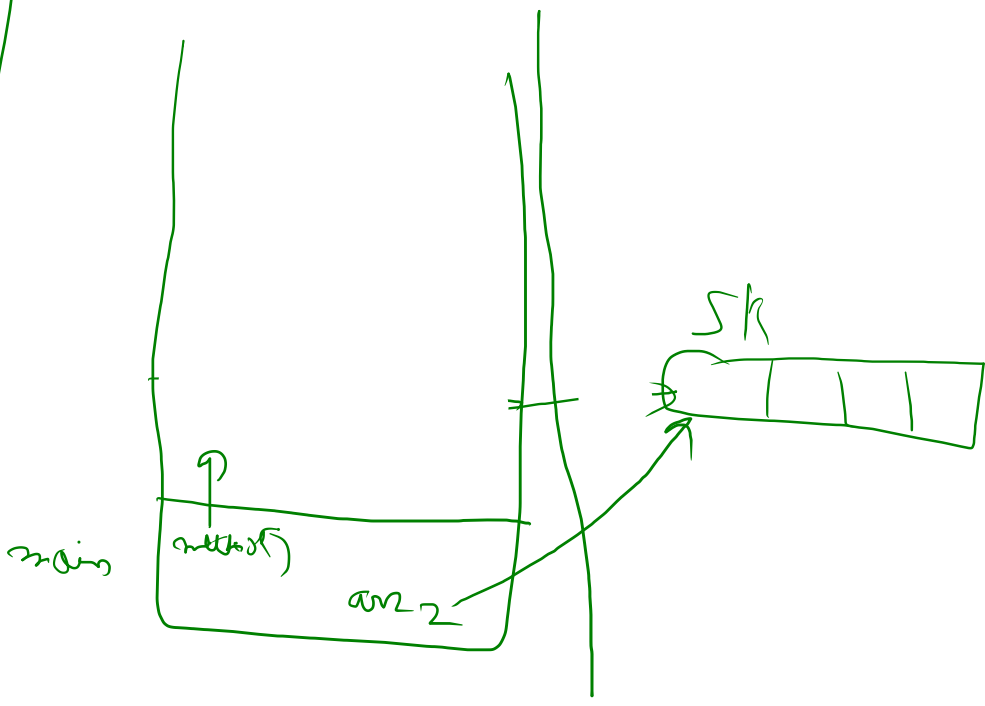
Time Complexity
&
Space Complexity




psvm ( ) {
int[] arr = method ( );
}
ps int[] method ( ) {
return arr;
}


Sk
main
method()
arr2

Time Comp

→ How much time your code takes to execute

Notation

Time in variable

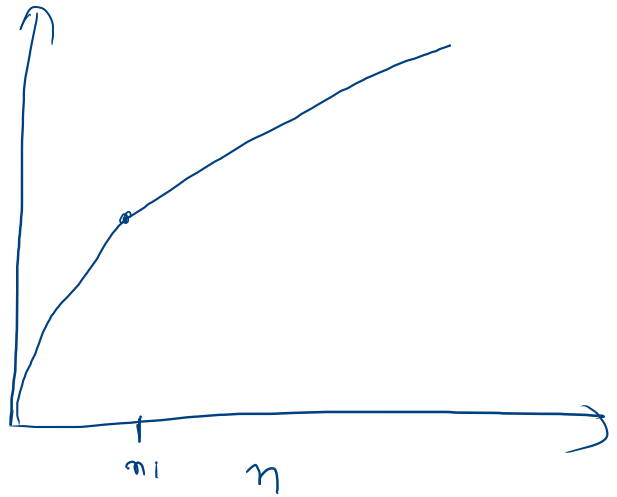
(1) Big oh notation ($O(\ )$)

(2) Big Omega notation ($\Omega(\ )$)

(3) Big theta notation ($\Theta(\ )$)

① Big oh (O ( ))

① Upper bound → Worst ② Lower bound → best



Fx1 arr → n length

Fx2 → n → input

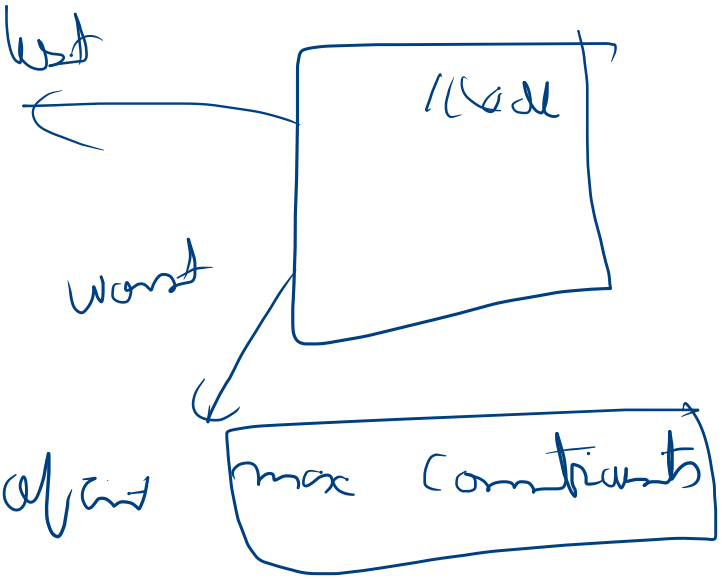
→ 1 2 3 4 → n

T(100) < T(10000)

Sum of n num

Sum of n natural number

$n \rightarrow 1$



$n \rightarrow$ max value of int

$n \rightarrow$ int

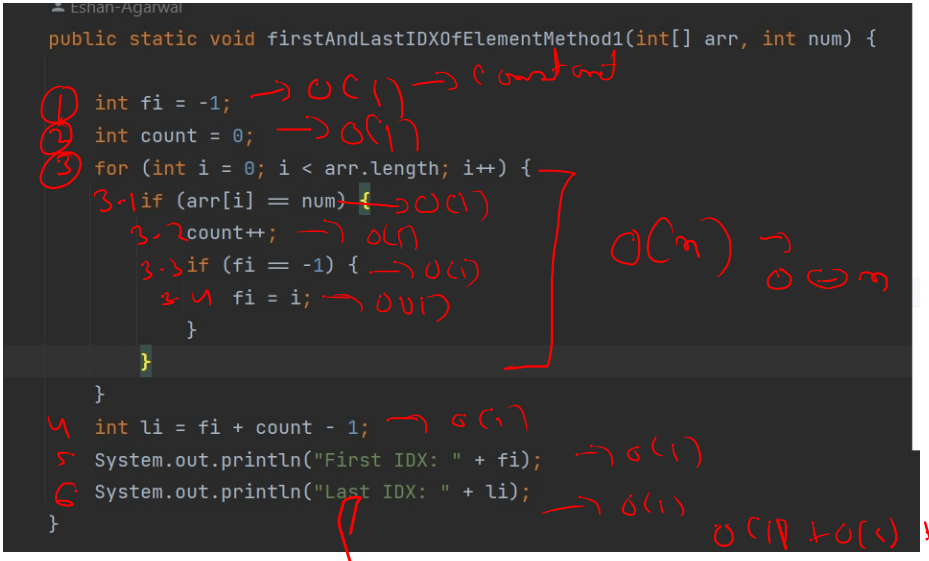
2 4 7 — — —

O( ) → Cares about upper bound

Primitive → stack
Non Primitive → heap

↓
worst time comp

O(1)
↓
Constant time
↓ in comp sense
n



O(n)

fi=-1

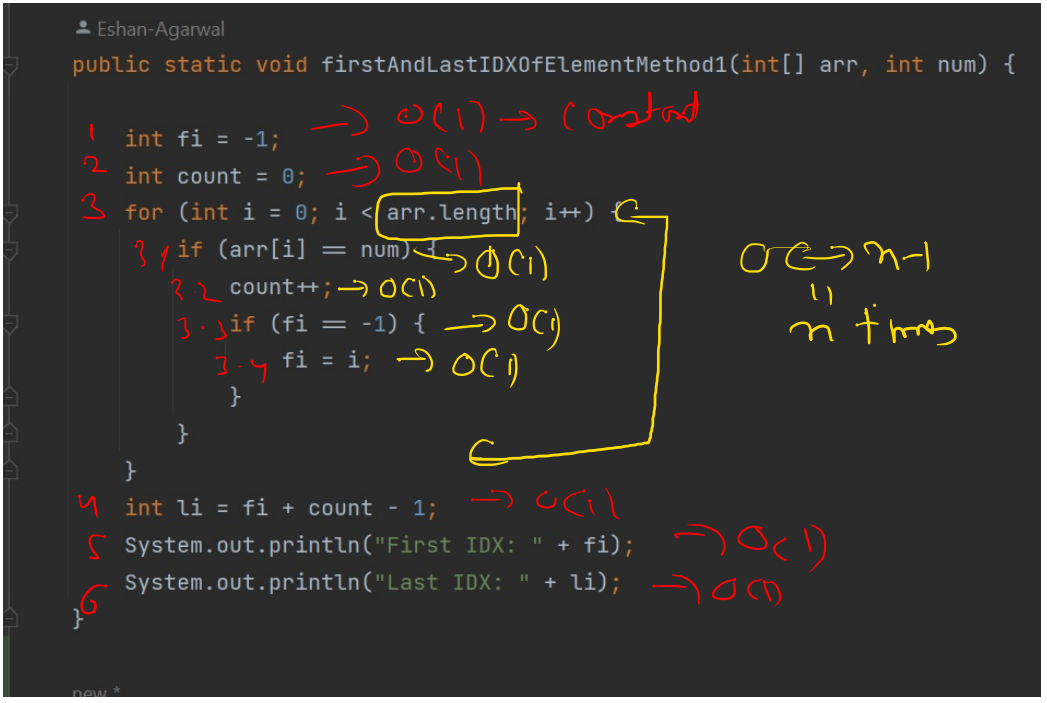
O(1)+O(n)+O(n)+O(1)+O(1)+O(1)

yellow → n times
O(1)

Red → 1 time
O(1)

neglect
O(1) × 1

+ O(1) × n
↳ O(n)



10 x C O(1)

O(10)

↳ n ⇒ sum of values

```
new *
public static void method() {
    int a = 1;
    int a = 1;
    int a = 1;
    int a = 1;
    int a = 1;
    int a = 1;
    int a = 1;
    int a = 1;
    int a = 1;

}
```

T(10) < T(100)

(2) Big Omega notation (Ω)

→ Lower bound → best-time comp

$\Omega(n)$

$n \rightarrow 1$

```java
public static void firstAndLastIDXOfElementMethod1(int[] arr, int num) {

    int fi = -1;
    int count = 0;
    for (int i = 0; i < arr.length; i++) {
        if (arr[i] == num) {
            count++;
            if (fi == -1) {
                fi = i;
            }
        }
    }
    int li = fi + count - 1;
    System.out.println("First IDX: " + fi);
    System.out.println("Last IDX: " + li);
}
```

(3) Big Theta notation (Θ)

→ tightest bound → best of all worst case times

Time Comp Practice

```java
// Time Com = O(n)
public static void singleLoop(int n) {

    for (int i = 0; i < n; i++) {
        System.out.print(i + " ");
    }
    System.out.println();

}
```

→ O(n)

→ 0 ⟷ n

i = 0

1

2

3

.

.

.

n

n times

O(n)

```java
// Time Com = O(n)
public static void singleLoop(int n) {

    for (int i = 0; i < n; i++) {
        System.out.print(i + " ");
    }
    System.out.println();
    for (int i = 0; i < n; i++) {
        System.out.print(i + " ");
    }
    System.out.println();
    for (int i = 0; i < n; i++) {
        System.out.print(i + " ");
    }
    System.out.println();

}
```

→ O(n)

→ O(n)

→ O(n)

Sequentially not nested

O(n) + O(n) + O(n)

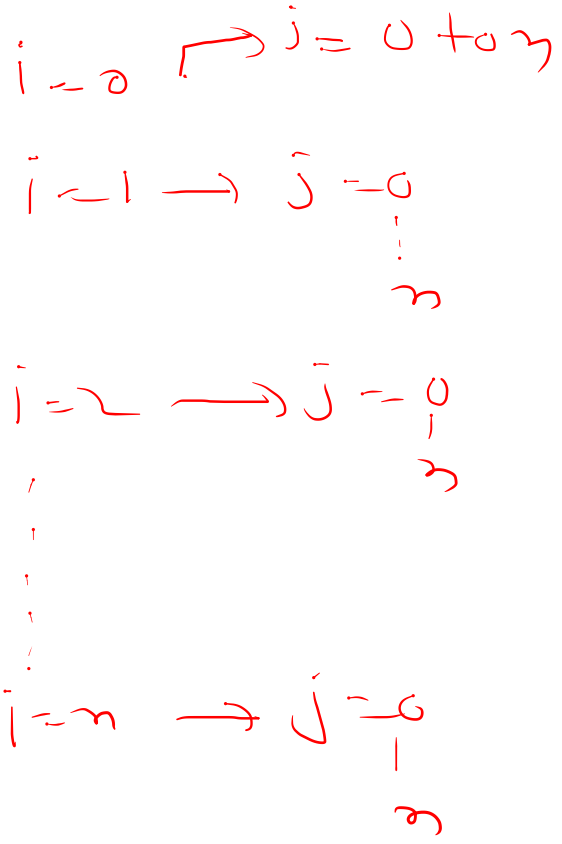
= O(3n) → Precise

O() → ignore constant ⇒ O(n)

```java
// Time Com = O()
public static void doubleLoop(int n) {

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            System.out.print(i + " " + j );
        }
    }
    System.out.println();

}
```



$O(n \times n)$

$O(n^2)$

n^1 / ...

$O(n^2)$ ✓

```java
// Time Com = O()
public static void doubleLoop1(int n) {

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < i; j++) {
            System.out.print(i + " " + j );
        }
    }
    System.out.println();

}
```

$n - 1$

$n \times n-1 \Rightarrow n^2 - n$

$0 + 1 + 2 + 3 + 4 + n$

$1 + 2 + 3 + 4 + \dots n-1$

$n(n-1) \Rightarrow O(n^2 - n)$

$i = 0 \rightarrow j = 0$ (0 times)

$i = 1 \rightarrow j \rightarrow$ 1 time

$i = 2 \rightarrow j \rightarrow$ 2 times

$i = 3 \rightarrow j \rightarrow$ 3 times

$i = 4 \rightarrow$ 4 times

$i = 5 \rightarrow$ 5 times

$i = n-1 \rightarrow j = 0$ to $n-2 \Rightarrow n-1$

$0 \times 0 +$

$1 \times 1 +$

$1 \times 2 +$

1×3

1×4

1×5

n

$n + 1$

$n(n-1)$ ✓

$n(n+1)$

$O(n(n-1)) \Rightarrow O(n^2 - n)$

$\Rightarrow O(n^2) - O(n)$ ignore

always ignore lower order term

$\Rightarrow O(n^2)$

$n(n-1) \rightarrow O(n^2)$

```java
// Time Com = O()
public static void doubleLoop2(int n) {

    for (int i = 0; i < n; i++) {
        for (int j = i; j < n; j++) {
            System.out.print(i + " " + j );
        }
    }
    System.out.println();

}
```

i = 0 → j = 0 to n-1 → n times

i = 1 → j = 1 to n-1 → n-1

i = 2 → j = 2 to n-1 → n-2

i = 3 → j = 3 to n-1 → n-3

⋮

i = n-2 → j = n-2 to n-1 → 2

i = n-1 → j = n-1 to n-1 → 1

n + n-1 + n-2 + n-3 + ... + 3 + 2 + 1

n(n+1)