Time & Space complexity

Time complexity ↓ $O(n^3) \rightarrow O(n)$ Extra space

Space complexity ↓ 10^9

target = 8

1	2	3	4	5	6

$n * n = O(n^2)$ & $(10^9)^2 = 10^{18}$

$O(n) \leftarrow n$

arr → | 1 | 2 | 3 | 4 | 5 | 6 |

$= O(n \log n)$

$10^9 * \log 10^9$

$10^9 * 9 * \log 10$

$10^9 * 9$

$\approx 10^{10}$

7, 8, 2 3, 9

Time → dependent on system/platform or not

- → code writing time
- → compilation time
- → deployment time
- → Requirement gathering time.

5ms Mac

50ms windows

Time complexity $= O(n) = O(10)$ → length array

↳ data

n: 10 → 100 (n^2) → 1000 (n^3)

mac: 1ms → 10ms → 100ms

window: 5ms → 50ms → 500ms

O(n^2) → n logn

log n

TC = n log n

SC = O(1)

Space Complexity → line of code

Stack | Heap

new

tree, LL

arr [2|3|4|1]

[1|2|3|4] → Heap

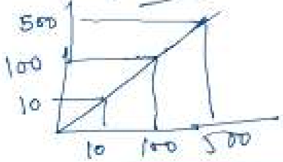
O(n)

arr → [1|2|3|4|5|6|7|8]

Linear Search

O(n)

32 elem



arr = [1|2|3|4|5|6|7|8]

$\frac{0+7}{2} = 3$

B.S → log n

32 → 5 iteration

TC → (1) Discard all smaller n powers

(2) Discard the constants.

$3n^2 + 2n^1 + 1n^0$

$3n^2 = (n^2)$

for { $O(n^2)$ }

for { $O(n)$ }

sysout(); $O(1)$

$= O(n^2) + O(n) + O(1)$

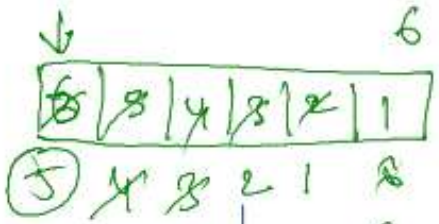
$= O(n^2)$

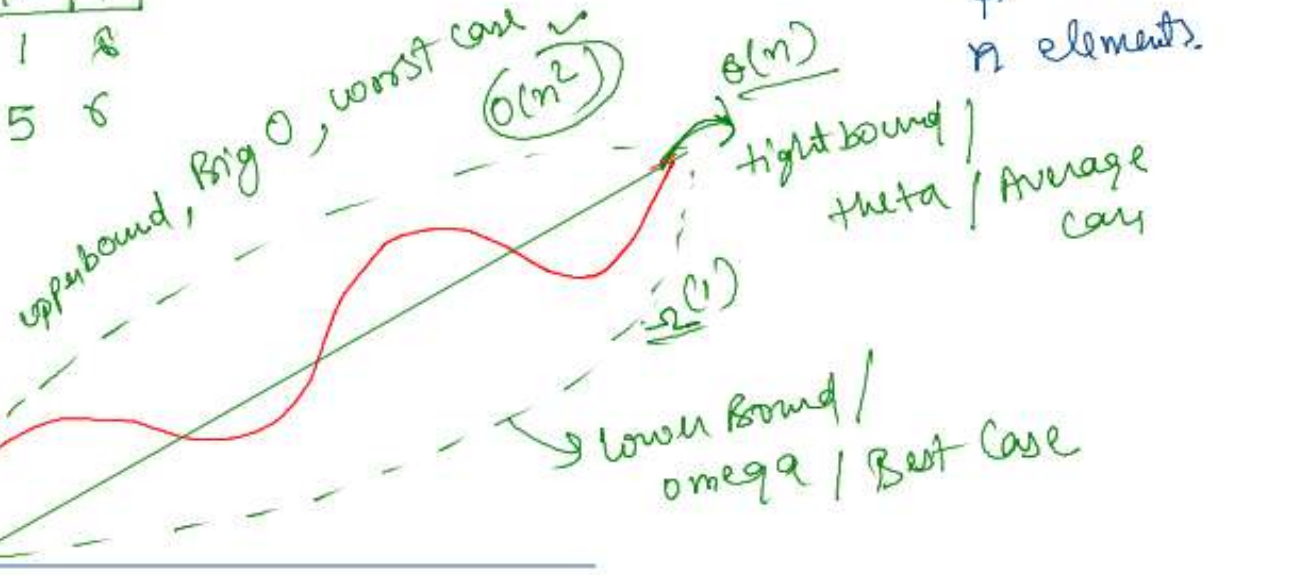
$O(n^2)$

$O(n^2)$

$= an^2 + bn + c$

where a, b, c are constants & n elements.





1 sec

for {

100 sec

$O(n^2)$

$2 * \infty$

$= \infty$

}

for {

100

$O(n^2)$

}

100 ← Sys( );

$O(n^2)$

exponential   cubic growth

TC ⇒ [log(n) + n^2 + 2^n + n^3]   O( ) → Big O

3 + 64 + 2^8 + 8^3

Big O

$O(2^n)$

2^9 → 516

9^3 → 2543

n = 10^9

2^{10^9}

② → nlogn + √n + logn

n = 2048

2048 * 11   √2048   11

= O(nlogn)

1000

(1000) = 10^6 = 2^{1000}

$2^{10} = 1024$

2000 = 2000n^0 + 3000n^0 + 100n^0

= O(n^0)

= O(1)

$(1024)^{100}$

$(1000)^{100}$

$(10^3)^{100}$

= 10^{300}

{ 2000 }

{ 1000 }

{ 100 }

O(1)   = O(n)

1	2	3	4
0	1	2	3

arr[2] = 20;

n = 35

→ for(int i = 2; i < n ; i++){

if(n % i == 0){ 1ms

SOP("Not prime"); 1ms

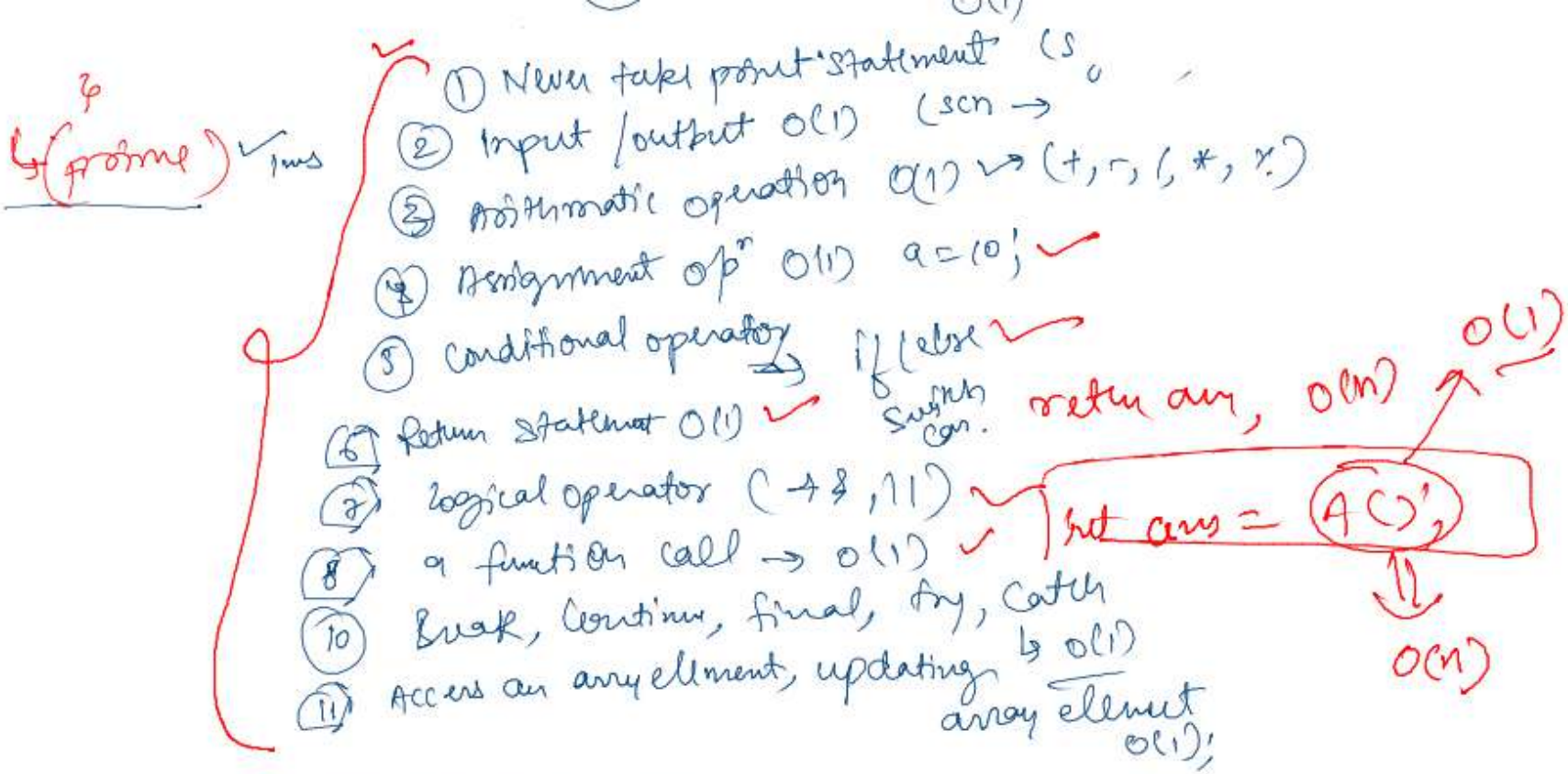
} 1ms

n − 2

O(1) * (n)

= O(n)

O(1)




```
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();
    }

    int max = Integer.MIN_VALUE;
    for(int i=0;i<n;i++){
        for(int j = i;j<n;j++){
            if(arr[j]>arr[i]){
                int diff = arr[j]-arr[i];
                if(diff>max){
                    max= diff;
                }
            }
        }
    }

    System.out.println(max);
    /* Enter your code here. Read input from STDIN. Print output to
```

O(1)
O(n)
O(n)
O(1)
O(n²)
O(1)
O(1)
O(1)
O(1)
O(1)
O(n²)
O(1)

i=0 n = 0
i=1 n = n
i=3 n = 3n
⋮
i=n n = n+n = n²

m > n

m * n ✓ 100 10

n^2 = i = 100

n log n

while (i = 0 to n) {

for (i = 0 to n) {

O(log n)

i *= 2;

}

i++;

}

i = 0 log n

i = 1 log n

i = 2 log n

i = n n log n

5 → int a = 10;

int b = 20;

int c = 30;

int d = 40;

int e = 50;

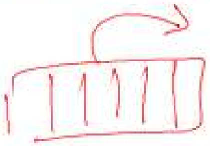
int [] arr = new int [n] 1000

— θ(n)

12B

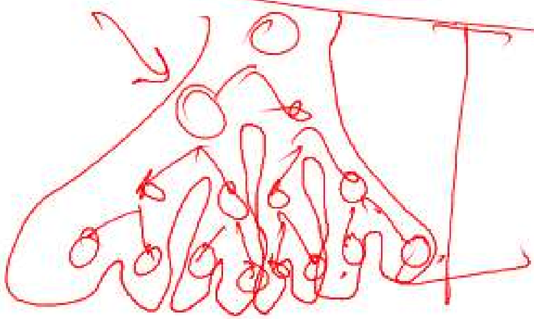
a | b | c → O(1)

→ O(n)



TC ⇒ # of recursion calls *

size of tree



1 2 — n

for (row) {

for (col) {

}

for (col) {

for (col) {

}

= row * col ⇒ n m

n * n

n^2