Brute force ✓

Que. ⇝ DSA

[1st Basic approach
not optimized sol^n.

max

max

6 > max

(1 6 2 4 5 3) ✓

Que.

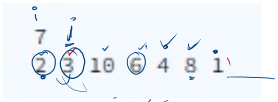
1. Brute force
2. optimized sol^n. ✗ 1

1 2 3 4 5 6

maximum difference between the two elements

Pair:

A[j]-A[i]



R > L

R - L =

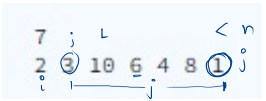
Brute force

10 - 3 = 7
6 - 3 = 3
4 - 3 = 1
8 - 3 = 5

3 - 2 = 1
10 - 2 = 8
6 - 2 = 4
4 - 2 = 2

8 - 2 = 6
1 - 2 = -1

8 - 6 = 2
8 - 4 = 4

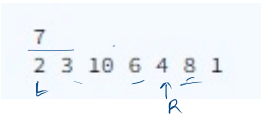


R - L

9 - 10 =

if (A[j] > A[i])
{
d = 3 - 2 A[j] - A[i]
}

Optimized soln



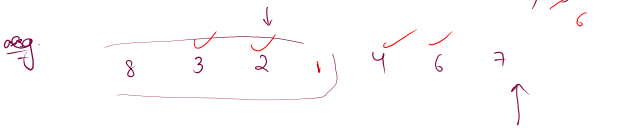
Logic:

mtn
↳ min. till now

mtn = A[0]; //2

★ to find best possible ans till now for this ele.
↳ should min till now to pair.

ans = 3 5 6



3

7 - 1 = 6

R - L
4 - 1 = 3

6 - 1 = 5

Sample Input 0

```
7
2 3 10 6 4 8 1
```

ans = ∅ 1

3 6 2 4 10 8 1

i

ans = ∅ 3 8

mtn = A[0] = 3 2

d = A[i] - mtn

R

6 - mtn

6 - 3 = 3

4 - 2 = 2

10 - 2 = 8

8 - 2 = 6

1 - 2 = -1


```
7
2 3 10 6 4 8 1
```

```java
import java.io.*;
import java.util.*;

public class Solution {

    public static void main(String[] args) {
        Scanner scn = new Scanner(System.in);

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

        int ans = 0;
        int mtn = A[0];              //max till now
        for(int i = 1; i < n; i++){
            ans = Math.max(ans, A[i] - mtn);
            mtn = Math.min(mtn, A[i]);
        }
        System.out.println(ans);
    }
}
```

```
7
2 3 10 6 4 8 1
```

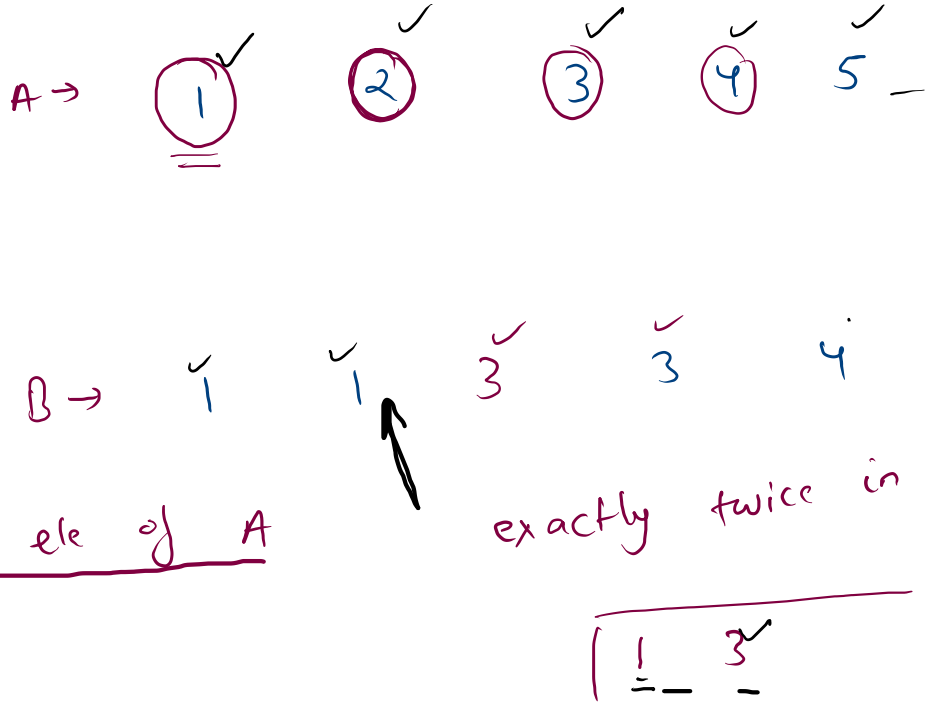
A[i] - mtn

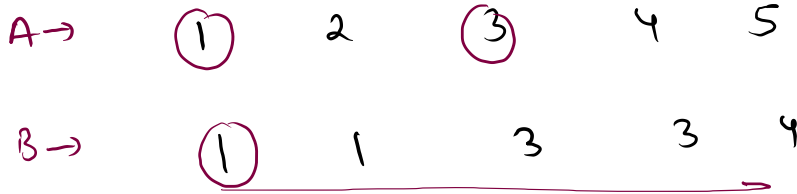
3 - 2 = 1 = d

Double Occurence.

Sample Input 0

```
n 5
  1 2 3 4 5
m 5
  1 1 2 3 4
```






```java
//logic

for(int i = 0 ;i < A.length; i++){
    //A[i] -> A[0] = 1
    int count = 0;

    for(int j = 0; j < B.length; j++){
        if(A[i] == B[j]){
            count++;
        }
    }
    if(count == 2){
        System.out.print(A[i] + " ");
    }

}
```

1

C = 0

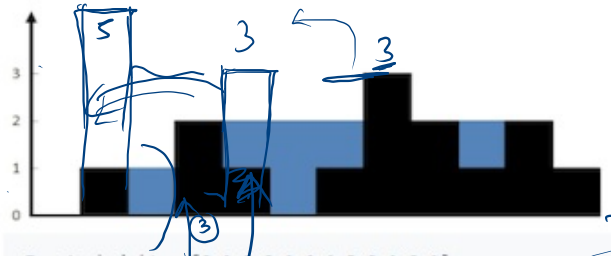
1 by 1

[for 1 ele of A iterate entire B.

```java
public static void main(String[] args) {
    Scanner scn = new Scanner(System.in);
    int n = scn.nextInt();      //size of A array
    int [] A = new int[n];
    for(int i = 0; i < n; i++){
        A[i] = scn.nextInt();
    }
    int m = scn.nextInt();      //size of B array
    int [] B = new int[m];
    for(int i = 0; i < m; i++){
        B[i] = scn.nextInt();
    }
    //logic
    for(int i = 0 ;i < A.length; i++){
        //A[i] -> A[0] = 1
        int count = 0;

        for(int j = 0; j < B.length; j++){
            if(A[i] == B[j]){
                count++;
            }
        }
        if(count == 2){
            System.out.print(A[i] + " ");
        }

    }
```

Store maximum

Example 1:




```
Input: height = [0,1,0,2,1,0,1,3,2,1,2,1]
Output: 6
Explanation: The above elevation map (black section) is represented by
array [0,1,0,2,1,0,1,3,2,1,2,1]. In this case, 6 units of rain water
(blue section) are being trapped.
```

5, 3 = 3

rm = lm

3 == 3

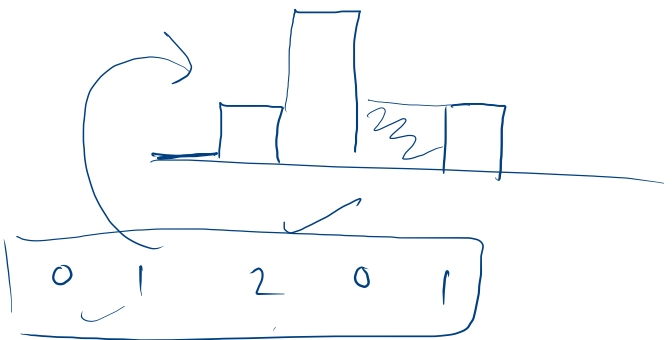
5 == 3

1, 3

1 - 2 = -1

ans = 0

min (left max , right max) - A(i)



```java
for(int i = 0; i < n; i++){
    //Step 1. : find left max
    int leftMax = A[i];
    for(int j = i-1; j >= 0; j--){
        leftMax = Math.max(leftMax, A[j]);
    }

    //Step 2: rightMax
    int rightMax = A[i];
    for(int j = i + 1; j < n; j++){
        rightMax = Math.max(rightMax, A[j]);
    }


    //ans for each cell
    ans += Math.min(leftMax, rightMax) - A[i]

}

System.out.println(ans);
```

i=1

2 1 0 1 3

2<5

1 1 0 1 3

3, 3

3 - 3 = 0

ans = 1

1, 3

1 - 1 = 0

1, 3

1 - 1 = 0

1, 3

1 - 1 = 0

1, 3

1 - 0 = 1

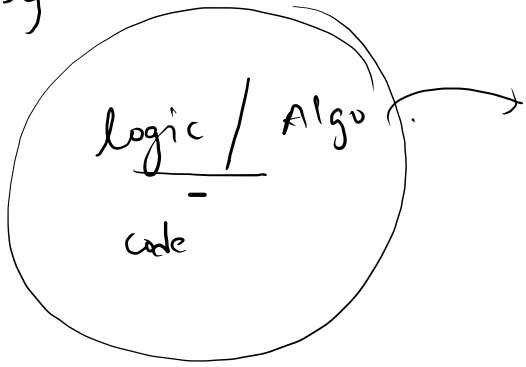
Time Complexity.

↳ not exactly time =

how much relative steps your program takes to complete.

Time complexity

time taken by



to complete execute in terms of user input.

int var = 10; } time to execute.

3 notations.

Best Case ← ω (omega)

Avg. Case ← Θ (theta)

Worst Case ← O (Oh)

Big

Constant time. O(1)

- Create variable — int var = 10; — O(1)
- Math. — c = a + b; — O(1)
  c = a / b — O(1)
- if (==) — O(1)
- else (— O(1)
- int [ ] A = new int [10] — O(1)

```
public static void main(String[] args) {
    int a = 10;
    int b = 5;
    if(a > b){
        System.out.println("A");
    }
    else{
        System.out.println("B");
    }
}
```

Linear time complexity. $O(n)$

(n) i/p.

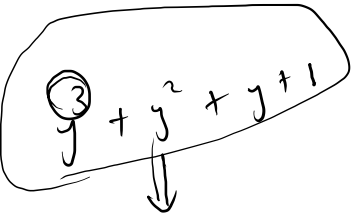
.... n elements.

```
int n = scn.nextInt();  O(1)

int [] A = new int[n];  O(1)

                        O(n)
for(int i = 0; i < n; i++){
    A[i] = scn.nextInt();
}

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

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

$+ O(1)$

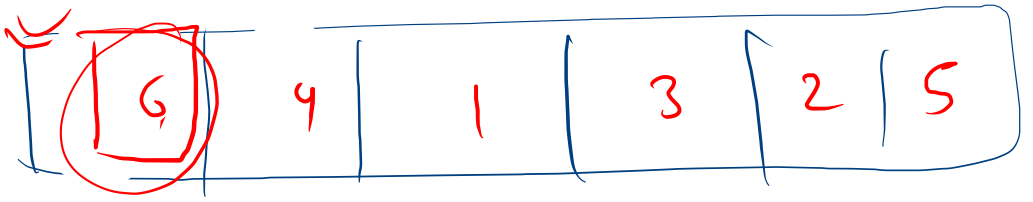
print each element



$3 \quad y^3 + y^2 + y + 16 \Rightarrow 3 \approx O(n)$

$O(n)$

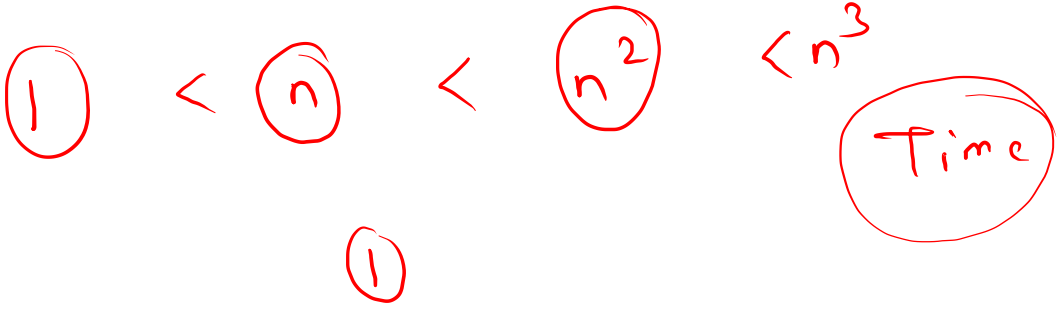
$O(n+2) \approx O(n)$



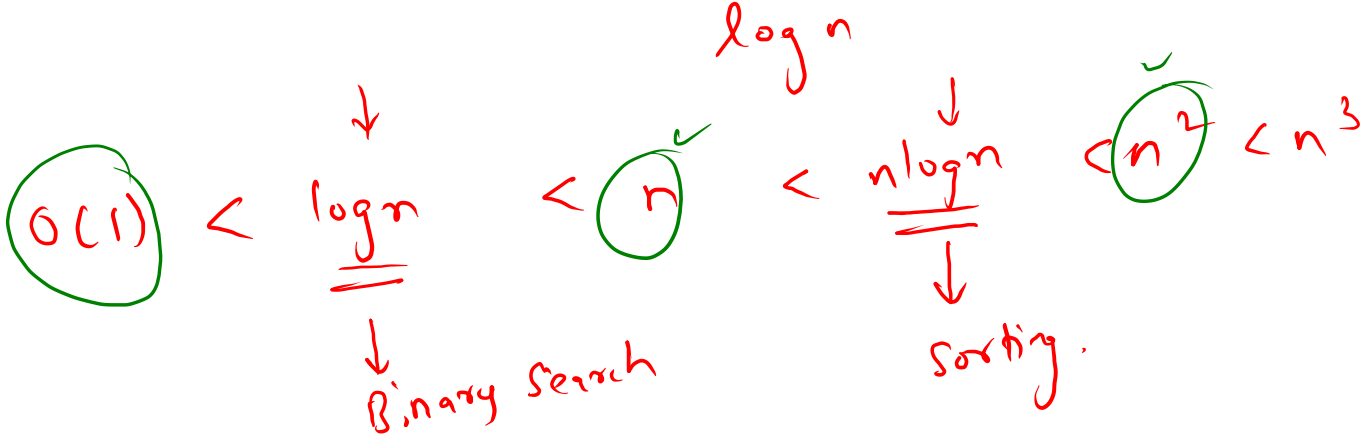
best.

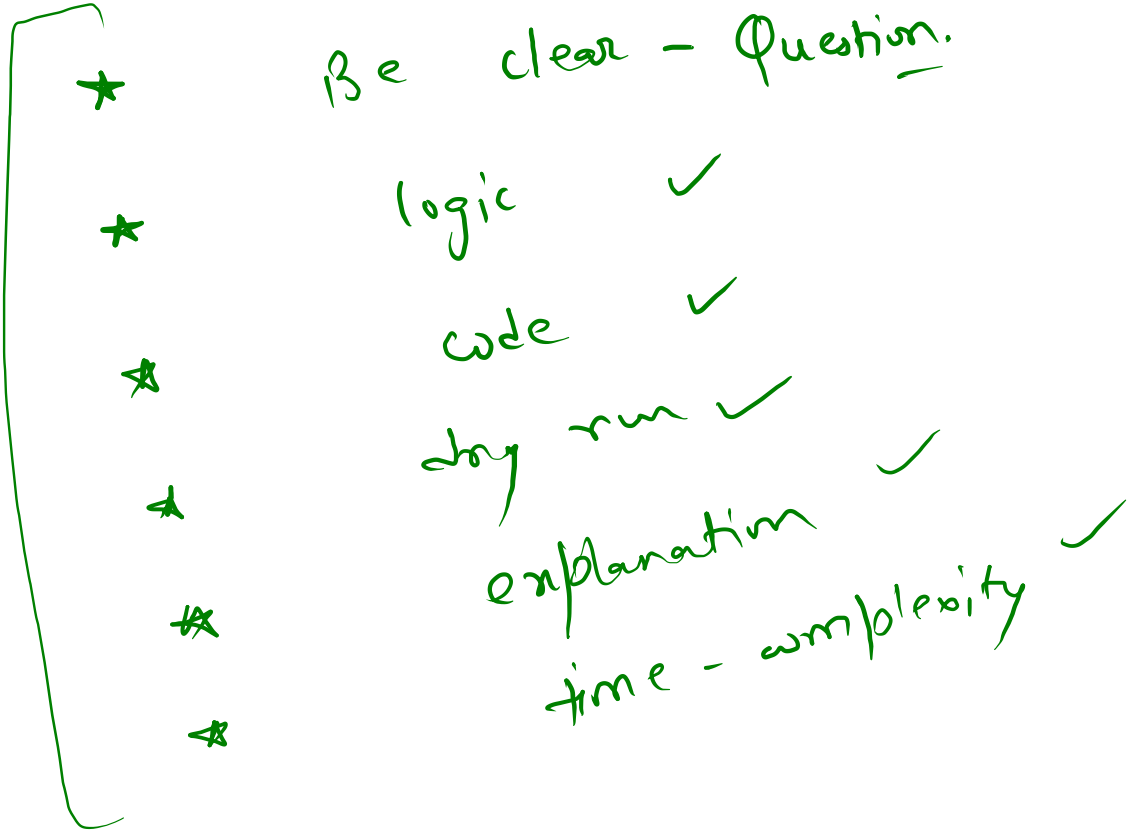
$x = 6$


space
n
star
n



1




Be clear - Question.
logic ✓
code ✓
dry run ✓
explanation ✓
time - complexity ✓