Store Maximum

lm :- left max height

rm = right max height

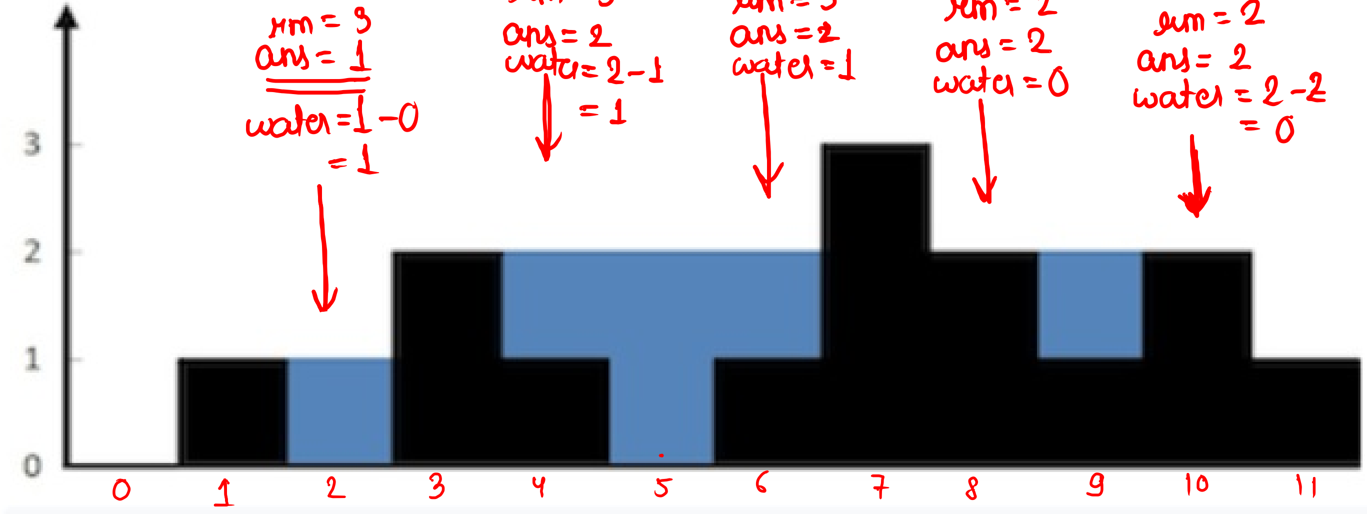
lm = 1
rm = 3
ans = 1
water = 1 - 0
= 1

lm = 2
rm = 3
ans = 2
water = 2 - 1
= 1

lm = 2
rm = 3
ans = 2
water = 1

lm = 3
rm = 2
ans = 2
water = 0

lm = 3
rm = 2
ans = 2
water = 2 - 2
= 0



lm = 0
rm = 3
ans = 0
water = 0

lm = 1
rm = 3
ans = 1
water = 0

lm = 2
rm = 3
ans = 2
water = 2 - 2
= 0

lm = 2
rm = 3
ans = 2
water = 2

lm = 3
rm = 3
ans = 3
water = 0

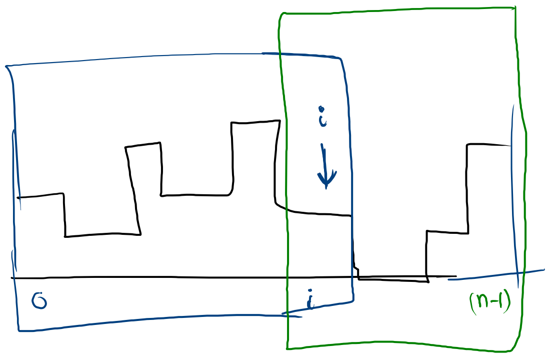
lm = 3
rm = 2
ans = 2
water = 2 - 1
= 1

lm = 3
rm = 1
ans = 1
water = 1 - 1
= 0

ans = 0 + 1 + 1 + 2 + 1 + 1
= 6

for each index

✓ left max (including itself)
✓ right max
✓ ans = min (lm, rm)
✓ water = ans − arr[i]

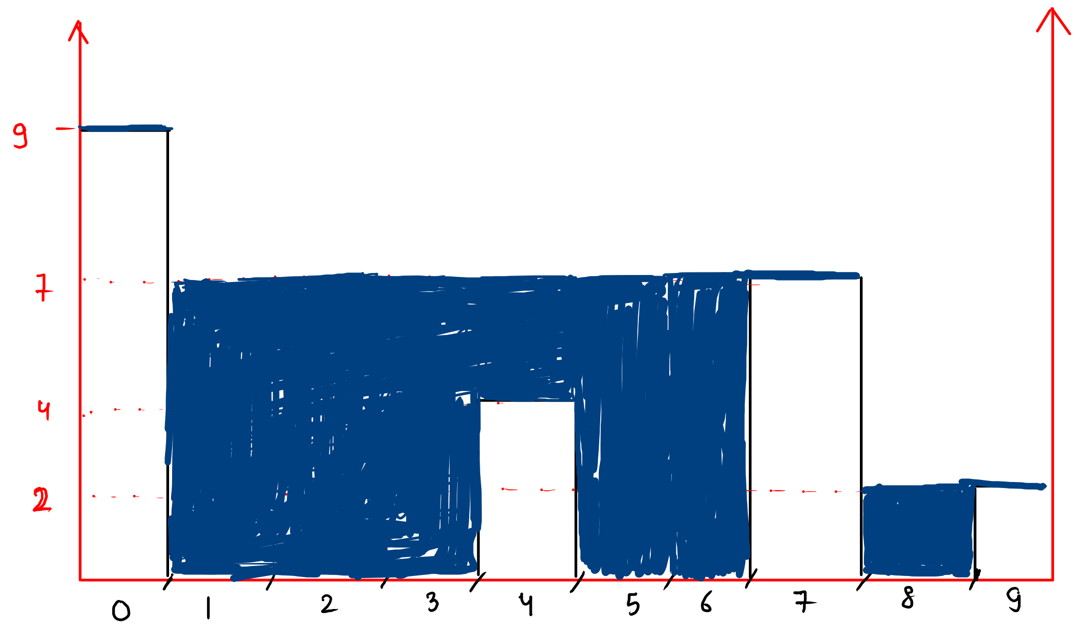


psudo code

```
1) traverse from 0 to n
    1.1) traverse from 0 to i
         and find max. value
         (left max)

    1.2) traverse from i to (n-1)
         and find max. value
         ( right max)

    1.3) ans = min (left max, right max)
    1.4) water = ans - arr[i]
                          ↳ current ele.
    1.5) result += water
```



for each index
↳ left max including itself
↳ right max " "
↳ ans = min (lm, rm)
↳ water = ans − arr[i]

ans = 0 + 7 + 7 + 7 + 3 + 7 + 7 + 2

code

```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;
}
```

⇒ **Time Complexity** (total time consumed by a program to get executed)

(TC)

M. Imp :- TC can only be calcuted using <u>no. of operations</u> performed.

Ex :-

```
main ( ) {
    Syso ("Hello");        // 1 operation
    Syso ("World");        // 1 operation
}
```

$\longrightarrow$ T.C notation

T.C = O(1)

$O(200) \cong O(1)$

(constant)

Ex1

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

i/p	$\longrightarrow$ operation
1	1
5	5
100	100
⋮	⋮
n	n

$\boxed{T.C \propto n}$

operations :- n

$T.C = O(n)$

Big O of n

Type of operations

- linear
- quadratic
- cubic
- logerithmic
- constant

Input	no. of operation
n	n
n	n^2
n	n^3
n	$\log(n)$
n	1