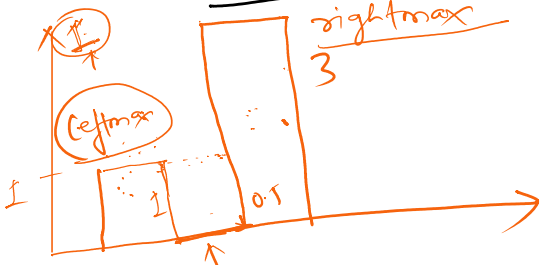
Store Maximum

height = [0,1,0,2,1,0,1,3,2,1,2,1]



leftmax

Rightmax

min(leftmax, rightmax) - height

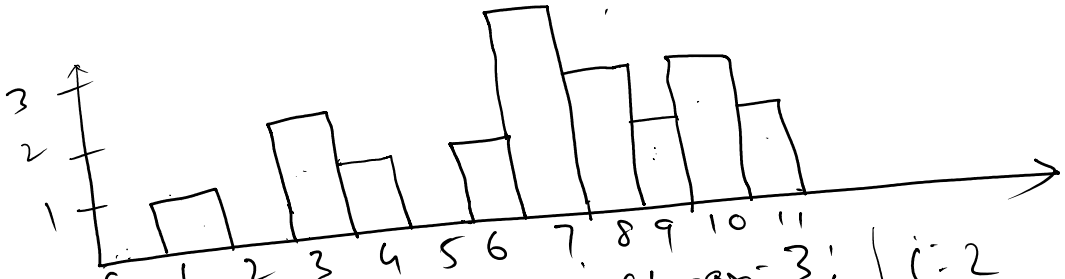
```
int n = arr.length; int totalwater=0;
for (int i=0; i<n; i++)                {
    int leftmax=0;
    int rightmax=0;
    for(int j=0; j<=i; j++) {
        if(arr[j]>leftmax) {
            leftmax = arr[j];
        }
    }
    for(int j=i; j<n; j++) {
        if(arr[j]>rightmax){
        rightmax = arr[j];
```

```
        rightmax = arr[j];
      }
    }
    int waterheight = Math.min(leftmax, rightmax) - arr[i];
    totalwater = totalwater + waterheight * 1
  }
  S.o.println(totalwater);
```

Dry Run

0 1 2 3 4 5 6 7 8 9 10 11

[0, 1, 0, 2, 1, 0, 1, 3, 2, 1, 2, 1]



i=0 leftmax=0; rightmax=3;
waterheight = 0-0=0
totalwater = totalwater + 0 = 0

i=1 leftmax=1, rightmax=3;
watersh = 1-1=0
totalwater=0

i=2
leftmax=1
rightmax=3
wh = 1-0 = 1
tw = 0+1 = 1

i=3, leftmax=2, rightmax=3
wh = 2-2=0
tw = 1+0*1 = 1

i=4
leftmax=2
rightmax=3
wh = 2-1 = 1
tw = 1+1 = 2

tw = 1+0 ...

=1

tw = 1+1 = 2

i = 9 leftmax = 3, rightmax = 2

min(3,2)

2 − 1 = 1

tw = 5 + 1*1 = 5+1 = 6

Time Complexity

↳ Time consumed by the program (code) for its execution..

```
for(int i=0;i<n;i++){
    int leftmax=0;
    int rightmax=0;
    for(int j=0;j<=i;j++){
      if(arr[j]>leftmax){
        leftmax=arr[j];
      }
    }
    for(int j=i;j<n;j++){
      if(arr[j]>rightmax){
        rightmax = arr[j];
      }
    }
    int waterheight = Math.min(leftmax,rightmax)-arr[i];
    totalwater+=waterheight*1;
  }
```

n = 1000

i = 0 to n

i = 0 → 0.05ms

0.05 × 1000

= 50ms

n = 10000

= 0.05 × 10000 = 500ms

Best Case → Ω (Big Omega)

Average Case → θ (Theta)

✓ Worst Case. → O (Big O)

Ex:- Write a program to find a number in an array

for(int i=0; i<arr.length; i++) { 1000 1 to 50

if(arr[i] == k) { arr[] = [100, 60, 80, 90, 5, 6, 1]

k = 100

if (arr[i] == k) {
S.o.pln(i);
break;
}
}

arr[] = [1...

K = 100
→ Best case

K = 80
→ Average case

K = 1
→ Worst case

Constant Time → O(K)

✓ int a=5; 1
✓ int b=6; 1
✓ int c=10000; 1

O(3) → O(constant)

→ constant time complexity

Linear Time - O(n)

```
for(int i=0; i<n; i++) {
    if(arr[i]==k) {
        S.o.pln(i);
        break;
    }
}
```

0 to n-1 → n iterations

↓ O(n)

O(n²) → Quadratic time complexity

```
int count=0; → (n)
for(int i=0; i<n; i++) {
    count=0;
    for(j=0; j<n; j++){
        if(arr2[j]==arr1[i]){
            count++;
```

n arr1 = [1, 2, 3, 4, 1]
n arr2 = [2, 2, 1, 1, 4]

res = [2, 2, 0, 1, 2]

0 to n → n.

```
        count++;
      }
    }
  arr1[i] = count;
}
```

0 to n -> n.

i : 0 to n -> n

i : 1, 0 to n -> n

i : 2, 0 to n -> n

i : 3, 0 to n -> n

⋮

i : n-1, 0 to n -> n

n + n + n + n ⋯

n

$n \times n = n^2$

$O(n^2)$

```
for(int i=0; i<n; i++){
```

→ (n)

```
}
for(int i=0; i<n; i++){
```

→ (n)

n + n + n

= 3n.

f(n)

O(n)

}

for(int i=0; i<n; i++){ → O(n)

→ (n)

}

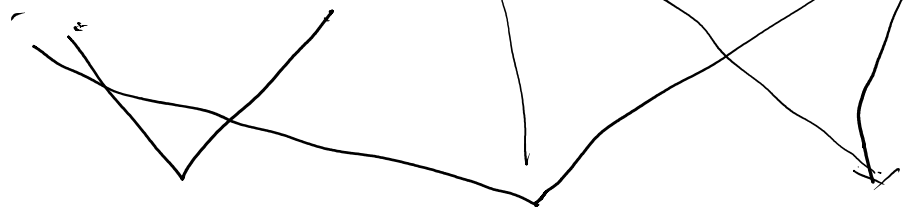
Logarithmic Time Complexity → O(logn)

Binary Search.

K = 80.

arr: 10, 40, 60, 70, 75, 80, 90

0 1 2 3 4 5 6



K > arr[3] →

arr(5) = K.

n = 16.

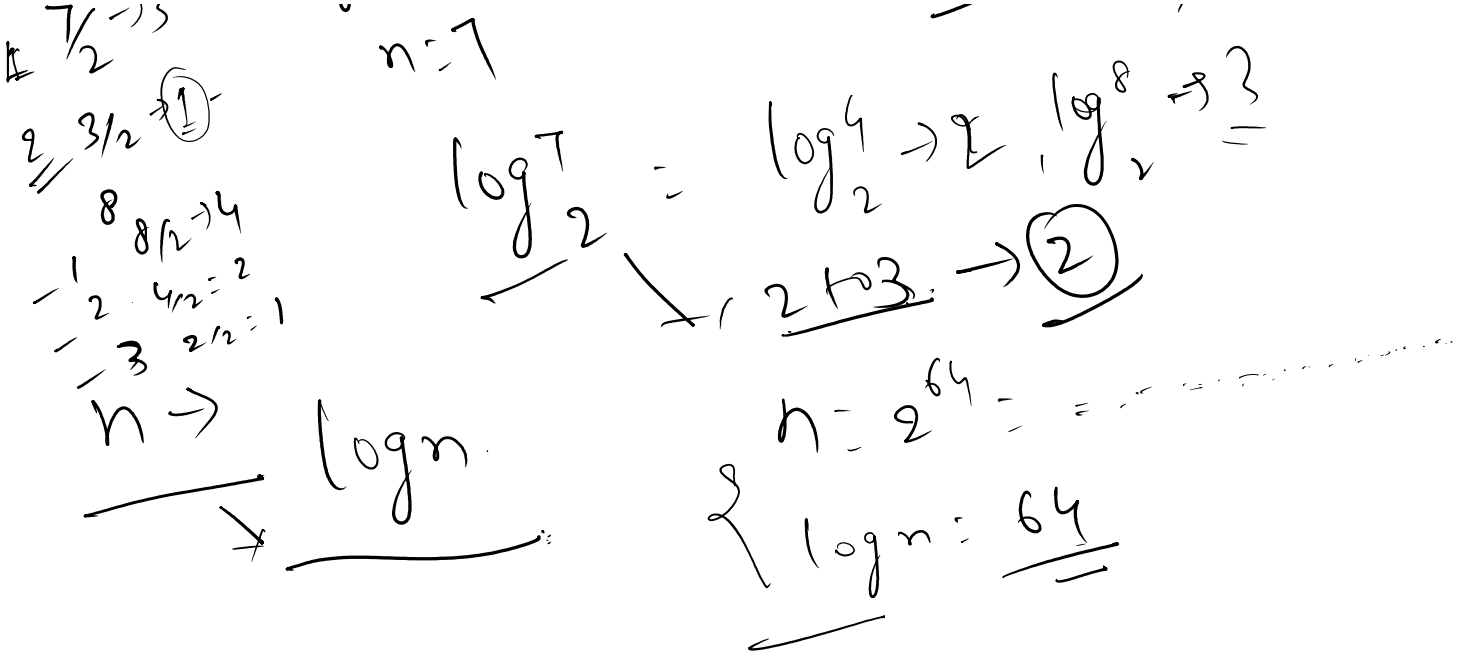
$\log_2^{16} = \log_2^{2^4}$

logn → 4 → $\log_2^{2^3} = 3\log_2^2$

n = 7

$7/2 \to 3$

$\log 8 \to 3$




```
while(left<=right){
int mid=(left+right)/2;
if(k==arr[mid]) return mid;
if(k<arr[mid]){
    right=mid-1;
}if(k>arr[mid]){
   left=mid+1;
}
}
```