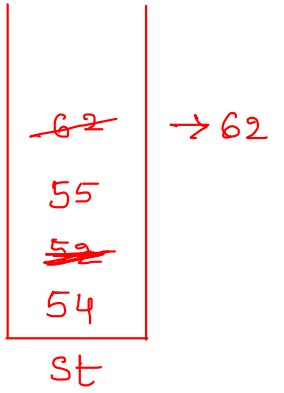
Consider the following sequence of operations on an empty stack.

push(54); push(52); pop(); push(55); push(62); s = pop();

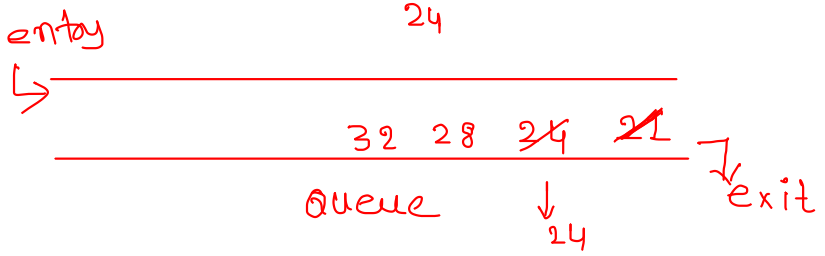
Consider the following sequence of operations on an empty queue.

enqueue(21); enqueue(24); dequeue(); enqueue(28); enqueue(32); q = dequeue();

The value of s + q is _____



LIFO



Suppose you are given an implementation of a queue of integers. The operations that can be performed on the queue are:

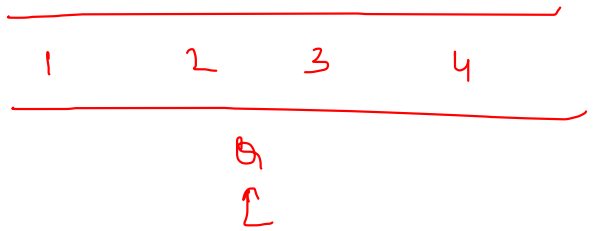
i. isEmpty (Q) — returns true if the queue is empty, false otherwise.

ii. delete (Q) — deletes the element at the front of the queue and returns its value.

iii. insert (Q, i) — inserts the integer i at the rear of the queue.

Consider the following function:

```
void f (queue Q) {
  int i ;
  if (!isEmpty(Q)) {
       i = delete(Q);
       f(Q);
       insert(Q, i);
  }
}
```



10-min's

What operation is performed by the above function f ?

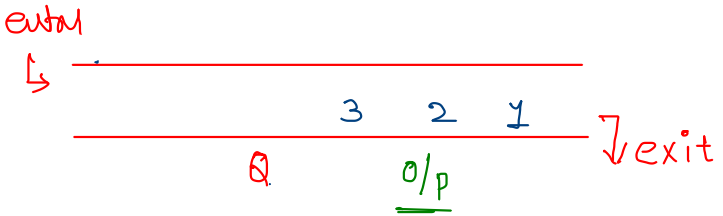
A Leaves the queue Q unchanged

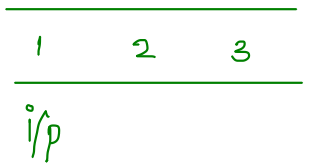
B Reverses the order of the elements in the queue Q

C Deletes the element at the front of the queue Q and inserts it at the rear keeping the other elements in the same order

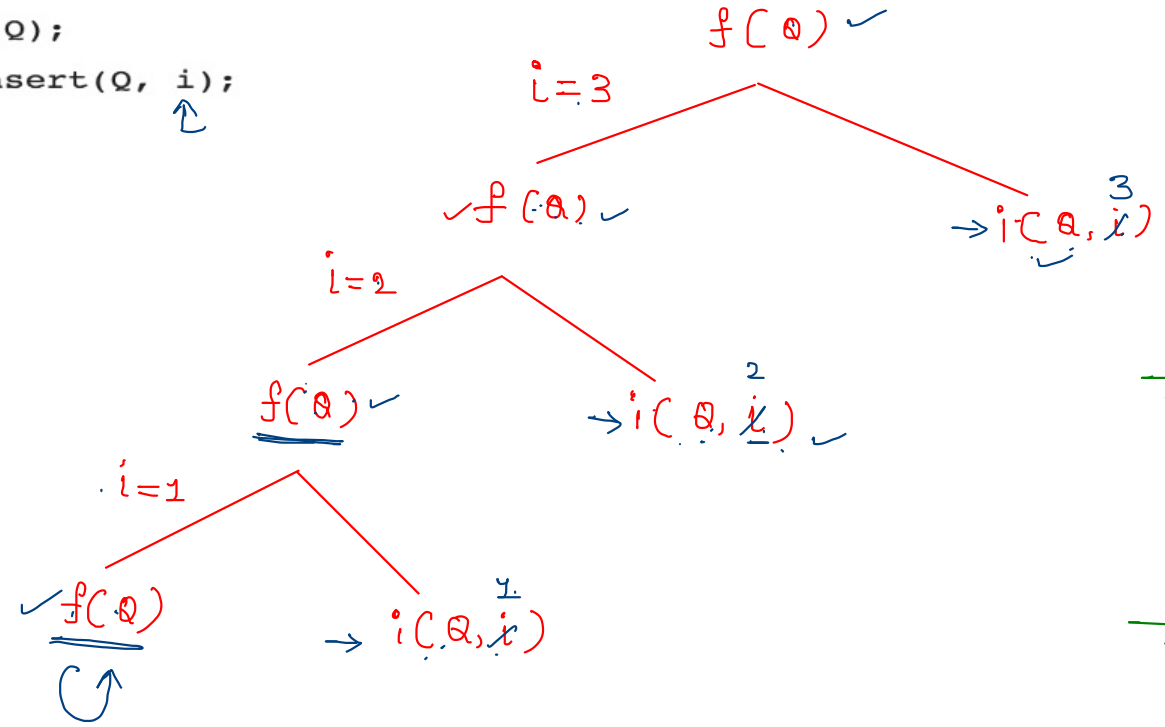
D Empties the queue Q

```
void f (queue Q) {
  int i ;
  if (!isEmpty(Q)) {
     i = delete(Q);
     f(Q);
     insert(Q, i);
  }
}
```





!false



→ Reverse contents of stack, with out using extra))

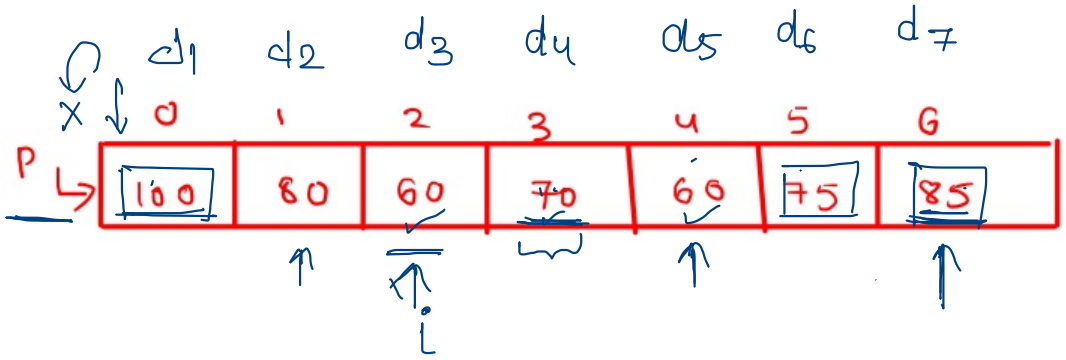
→ Reverse contents of queue without using extra))

→ Implement S using Q

→ Implement Q using S

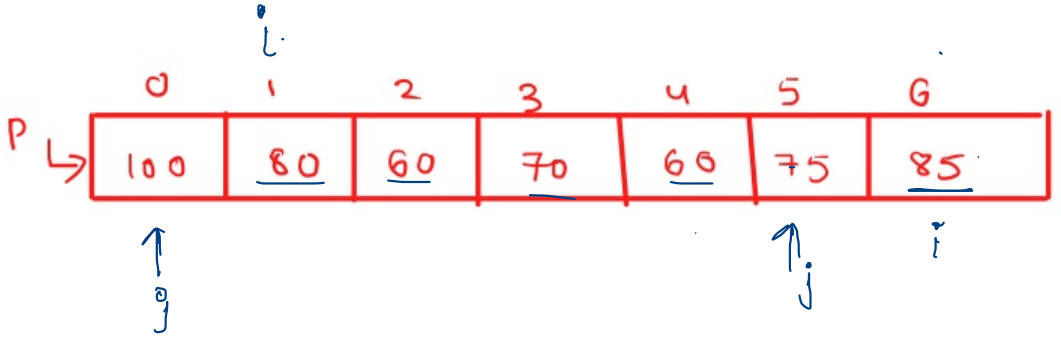
stock span problem

item → price



Including that day

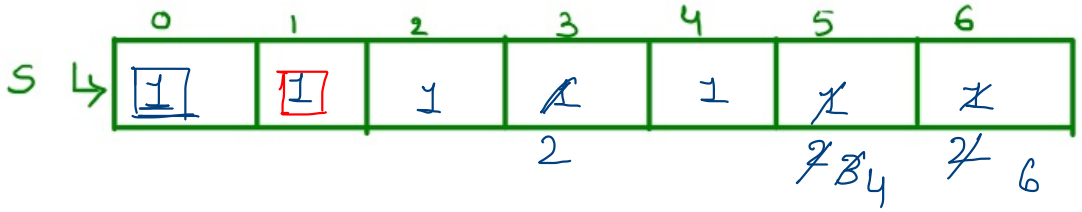
1 1 1 2 1 4 6 ⇒ o/p



$n = 7$

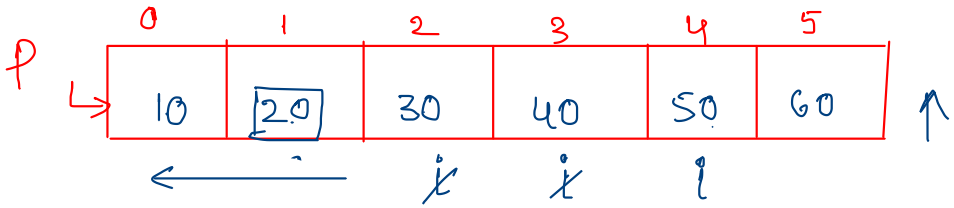
P[j] >/= P[i]

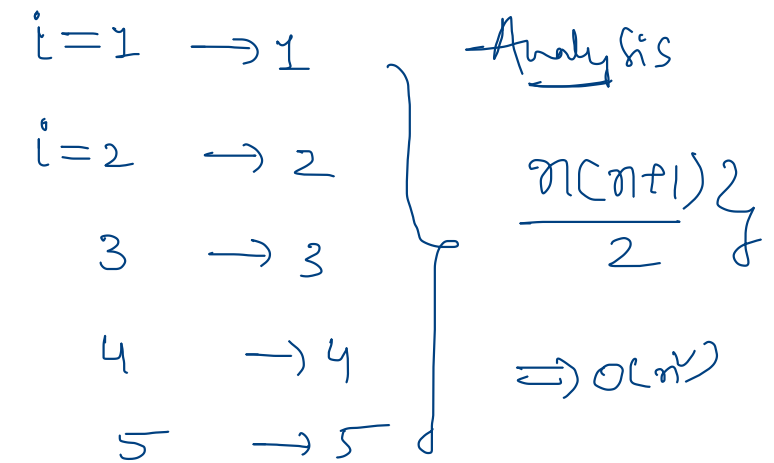
$\leq$



o/p

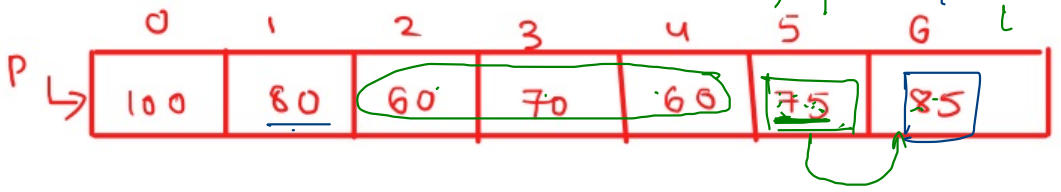
```
function calculateSpan(price[], n, S[])
{
        // Span value of first day is always 1
        S[0] = 1;


        for(int i = 1; i < n; i++)
        {
            S[i] = 1; //


           for(int j = i - 1; (j >= 0) && (price[j]<=price[i]); j--)
               S[i]++;
        }
}
```





Brute Force :-

$O(n^2)$



remember ⇒ stack

60 70 60 → 75 < 85

↳ > 75

⇒ $O(n^2)$?

{ repeative calculations

↳ some how if you can avoid.

then T.C ⇒ ↓ reduced

i=5 P[5] = 75

60 v/s 75

70 v/s 75 *

60 v/s 75

80 v/s 75

P[6] = 85

75 v/s 85

60 v/s 85

70 v/s 85 *

60 v/s 85

80 v/s 85

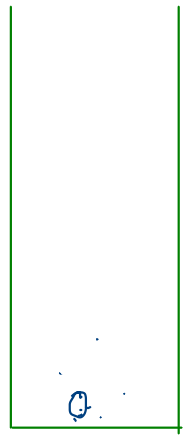
100 v/s 85 X

10 ↑

unit 1 | unit 2

Why?

BF ? → Stack

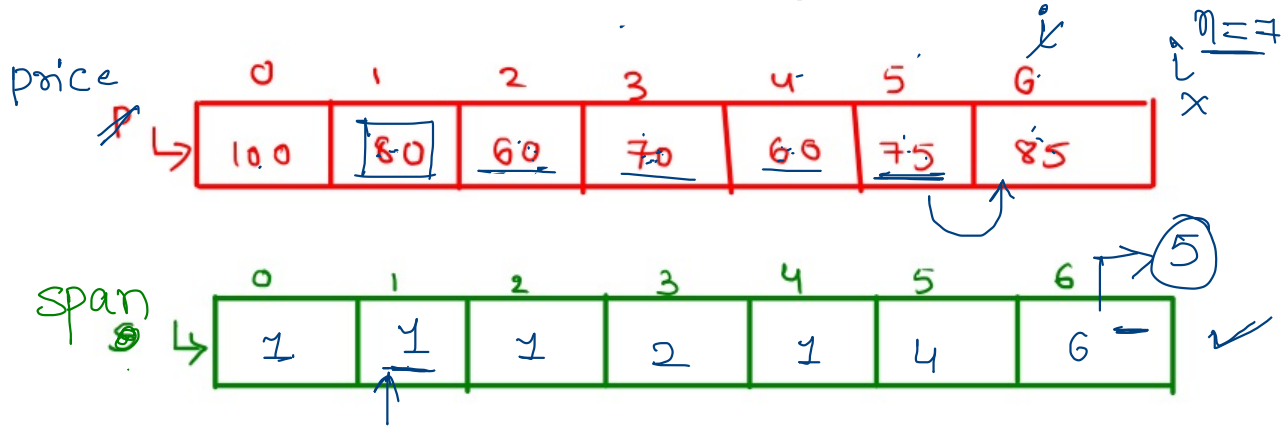


St

↳ Can contains all array indices

any value (0 to 6)

100 ≤ 80



i - st.top() price[st.top()] v/s price[i]

```
for(i=1; i<n; i++)
{
```

x → while(price[st.top()] ≤ price[i] && !st.isEmpty())
{

st.pop()

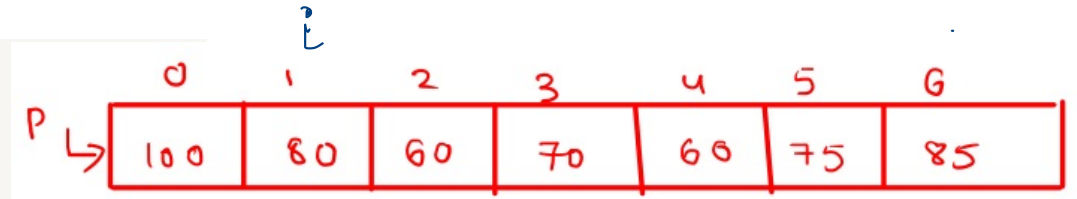
}

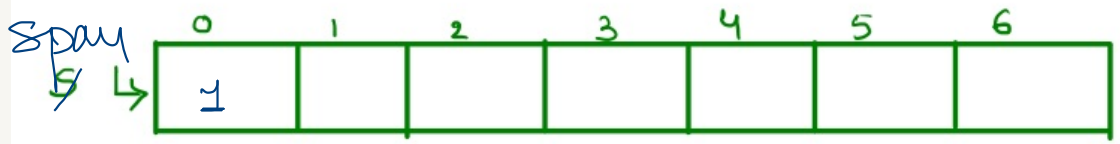
→

}

```
findSpan(price[],n,span[])
{
      Stack st
      st.push(0)
      span[0]=1
      for(i=1;i<n;i++)
      {
          while(!isEmpty() && price[st[top]]<=price[i])
          {
                st.pop()
           }
          span[i]=isEmpty()? i+1 : i-st[top]
          st.push(i)
       }
}
```

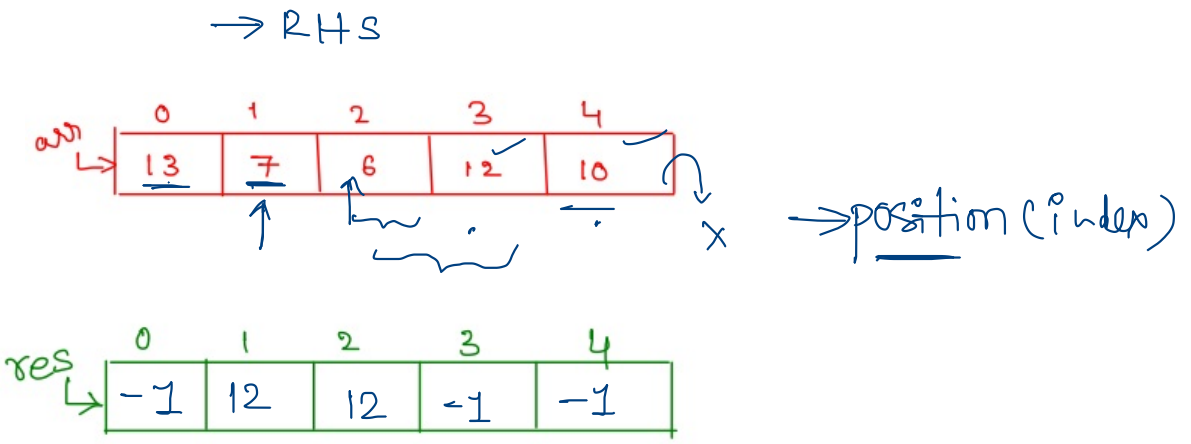
st.peek()
st.top()



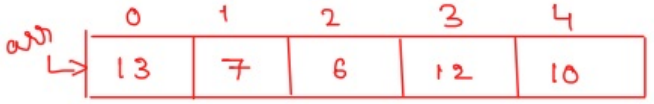


11:05

Next Greater Element to right



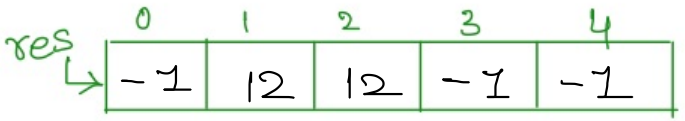
Brute Force :-



arr[j] >= arr[i]

temp = 12

break;

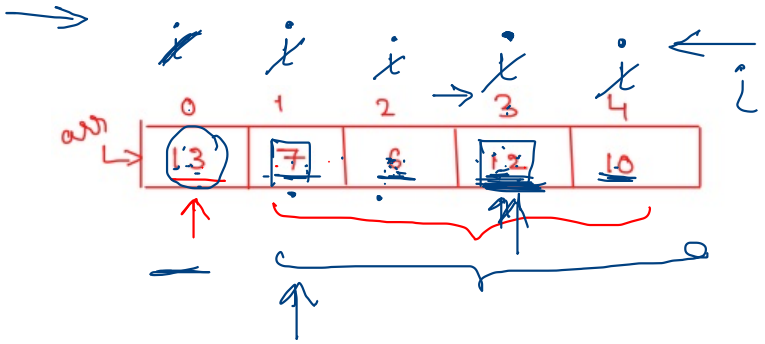


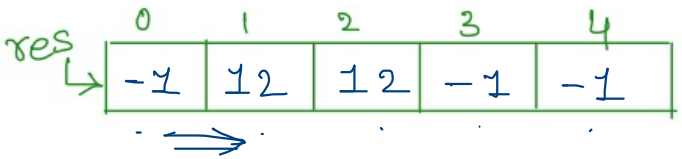

```
⇒ for(i=0; i<=n-2; i++)
  {   temp=-1
      for(j=i+1; j<n; j++)
      {
          if(arr[j] >= arr[i])
          {   temp = arr[j]; break;
          }
      }
      res[i] = temp;
}
```

O(1) ✓

O(n²)

↳ TLE





$7 \leq 13$

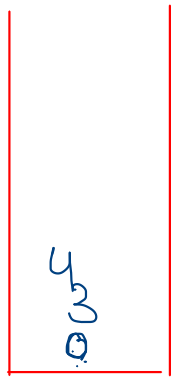
curr = 10

$10 \leq 12$ ✓

× if(curr ≤ arr[st.top()])
{
st.push(i)
}

$12 \leq 6$

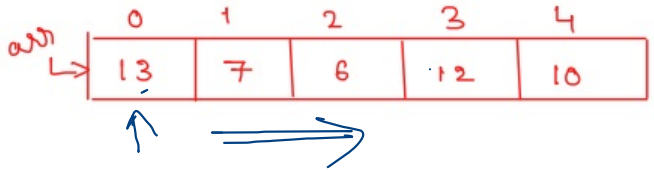
else ~~6 ≤ 7~~
{ while(!st.isEmpty() &&
curr > arr[st.top()])
{
index = st.pop() ✓
res[index] = curr
}
}



st → push ~~out~~ indices to stack.

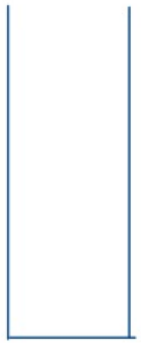
$12 > 13$ ×

```
function fun(arr[],n,res[])
{
        let st be a stack
        for(i=0;i<n;i++)
                res[i]=-1

        for(i=0;i<n;i++)
        {
                curr=arr[i]
                if(st.isEmpty())
                        st.push(i)
                else
                {
                        if(curr<=arr[st.peek()])
                                st.push(i)
                        else
                        {
                                while(!st.isEmpty() && curr>arr[st.peek()])
                                {
                                        index=st.pop()
                                        res[index]=curr
                                }
                                st.push(i)
                        }

                }
        }
        return res
}
```



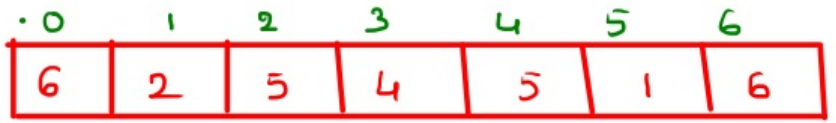
Design a stack such that getMin() is in O(1)

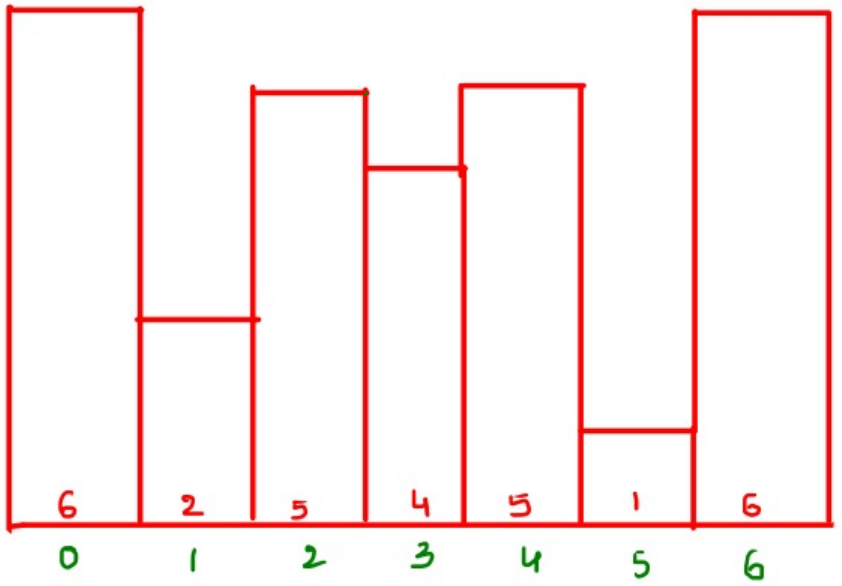
10, 5, 2, 6, getMin()

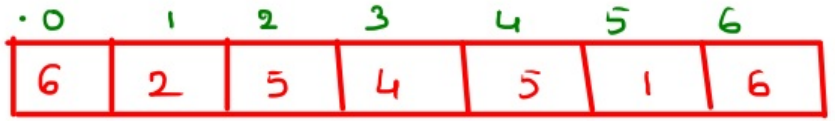


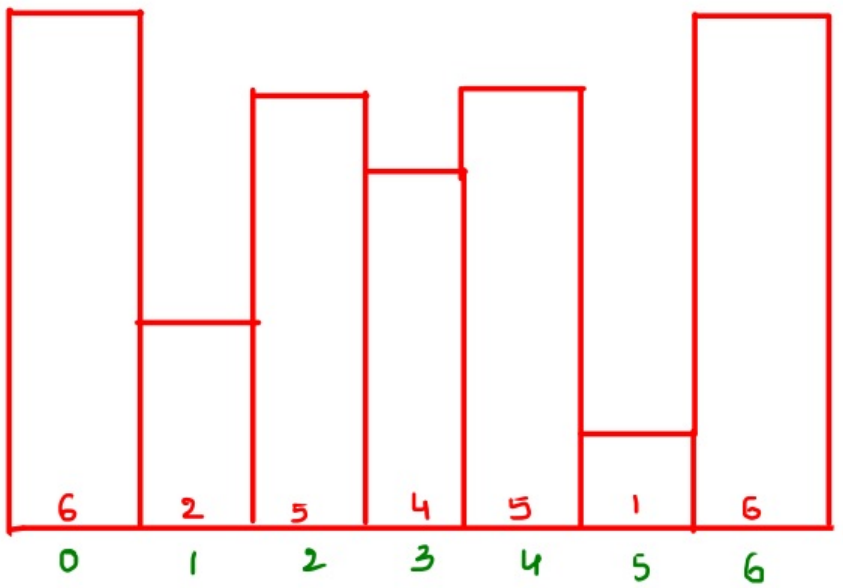
```
design a stack such that getMin() in O(1)
Let St be a global stack

add(data)
{
  if(s.isEmpty())
  {
      s.push(data)
      curr_min=data
  }
  else
  {
       if(data<curr_min)
       {
            s.push(data-curr_min)
            curr_min=data
       }
       else
         s.push(data)
}
delete()
{
      if(s.peek()<curr_min)
      {
          curr_min=curr_min-s.peek()
      }
      return s.pop()
}
```

Largest Rectangle Area in a Histogram

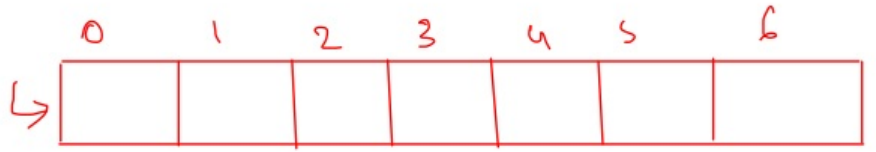




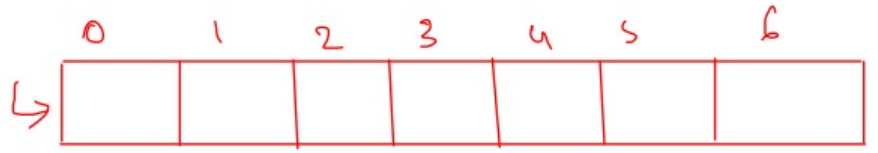

0 1 2 3 4 5 6
6 2 5 4 5 1 6


6 2 5 4 5 1 6
0 1 2 3 4 5 6

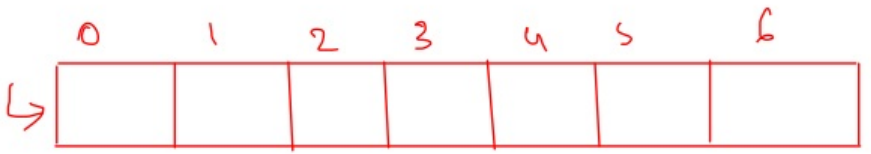
left



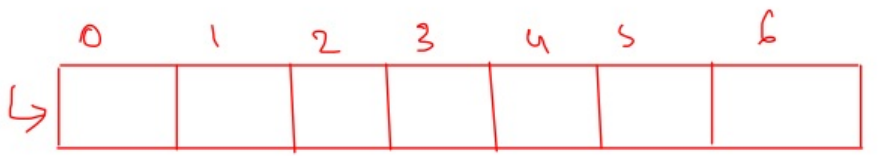
right



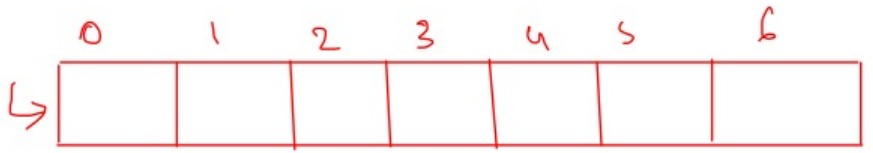
```
int maximumArea(int arr[],int n)
{
    stack<Integer> st = new Stack<Integer>
    int right[n] //nse index on right
    st.push(arr.length-1);
    right[arr.length-1]=arr.length;
    for(i=n-1;i>=0;i--)
    {
          while(st.size>0 && arr[i]<arr[st.peek()])
          {
              st.pop()
          }
          if(st.size()==0)
              right[i]=arr.length()
          else
              right[i]=st.peek()
          st.push(i)
    }
```

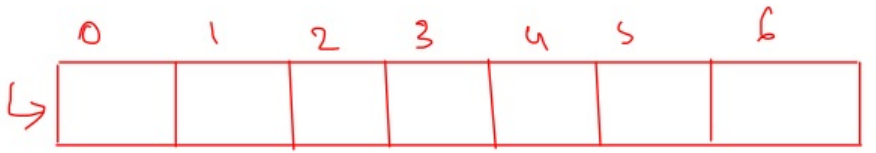


```
int left[n] // nse index on left
st=new Stack<>()
st.push(0)
left[0]=-1
for(i=1;i<arr.length;i++)
{
     while(st.size()>0 && arr[i]<arr[st.peek()])
     {
            st.pop()
     }
     if(st.size()==0)
     {
           left[i]=-1
      }
      else
      {
           left[i]=st.peek()
      }
   st.push(i)
}
```

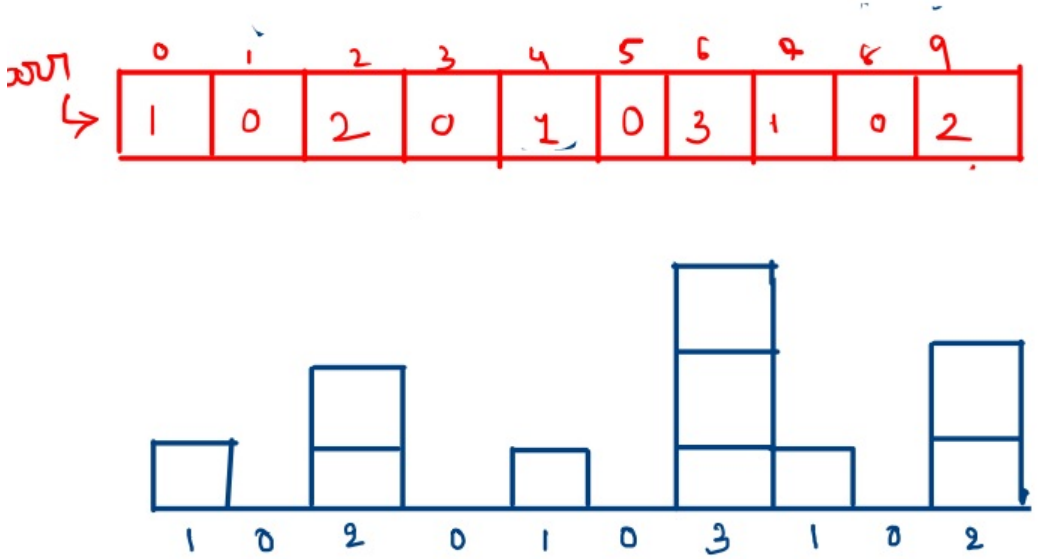


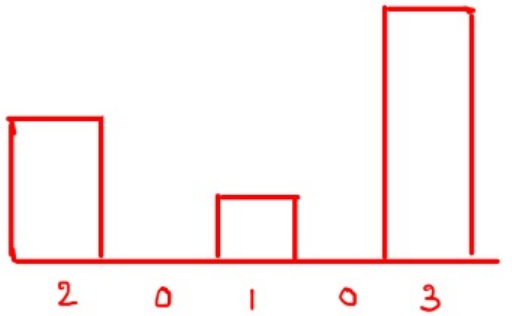
```
    maxArea=0
    for(i=0;i<arr.length;i++)
    {
        width=right[i]-left[i]-1;
        area=arr[i]*width
        if(area>maxArea)
        {
               maxArea=area
        }
    }
  return maxArea;
}
```



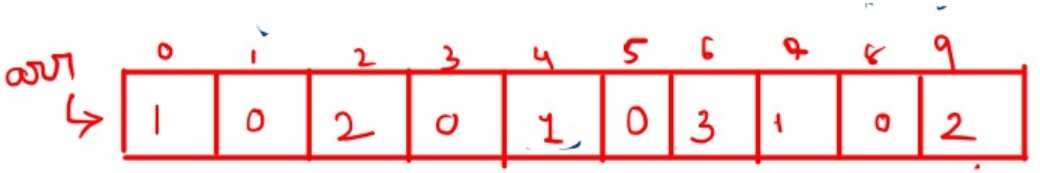


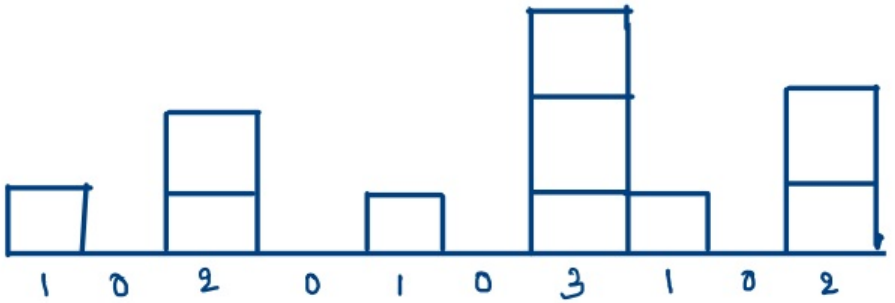
Trapping Rain water problem

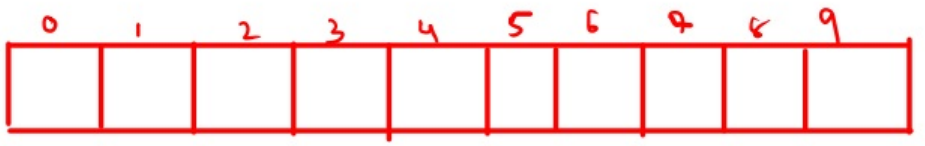


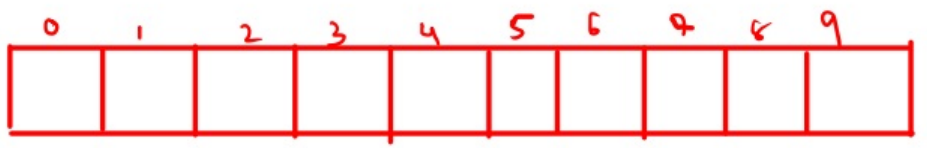


```
function maxWater(arr[], n)
{

    res = 0;

    for(i = 1; i < n - 1; i++)
    {

        left = arr[i];
        for(j = 0; j < i; j++)
        {
            left = Math.max(left, arr[j]);
        }

        right = arr[i];
        for(j = i + 1; j < n; j++)
        {
            right = Math.max(right, arr[j]);
        }

        res += Math.min(left, right) - arr[i];
    }
    return res;
}
```



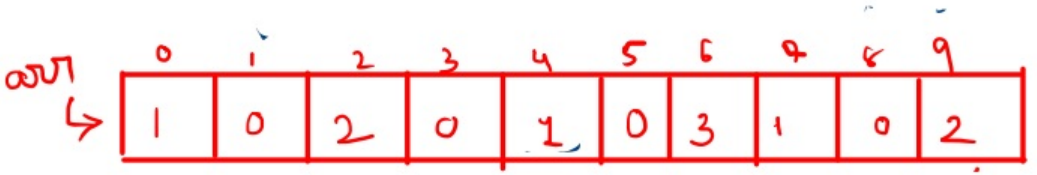

1 0 2 0 1 0 3 1 0 2


0 1 2 3 4 5 6 7 8 9


0 1 2 3 4 5 6 7 8 9

```
function findWater(arr[],n)
{

    let left[n], right[n]

    water = 0;

    left[0] = arr[0];
    for (i = 1; i < n; i++)
    {
        left[i] = Math.max(left[i - 1], arr[i]);
    }

    right[n - 1] = arr[n - 1];

    for (i = n - 2; i >= 0; i--)
    {
        right[i] = Math.max(right[i + 1], arr[i]);
    }


    for (i = 0; i < n; i++)
        water += Math.min(left[i], right[i]) - arr[i];

    return water;
}
```

```
function findWater(arr[], n)
{
        result = 0, left_max = 0, right_max = 0, lo = 0, hi = n -1;

        while (lo <= hi)
        {
            if (arr[lo] < arr[hi])
            {
                if (arr[lo] > left_max)
                    left_max = arr[lo];
                else
                    result += left_max - arr[lo];
                lo++;
            }
            else
            {
                if (arr[hi] > right_max)
                    right_max = arr[hi];
                else
                    result += right_max - arr[hi];
                hi--;
            }
        }
       return result;
}
```



→ n x n → searching

1 2 2 3 3 1 4 1 1 5 6

O(1)

O(n)

~~O(1)~~

O(1)

Java HashMap()

key	value
1	1
2	~~1~~ 2
3	1

object