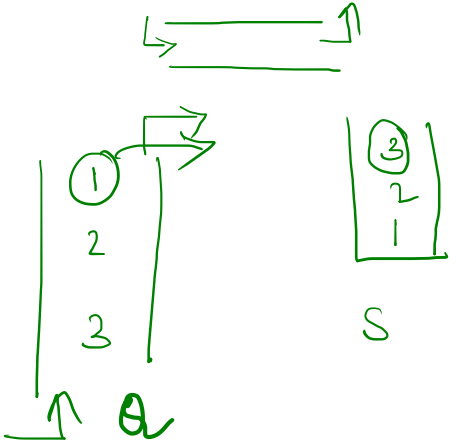
Day-7

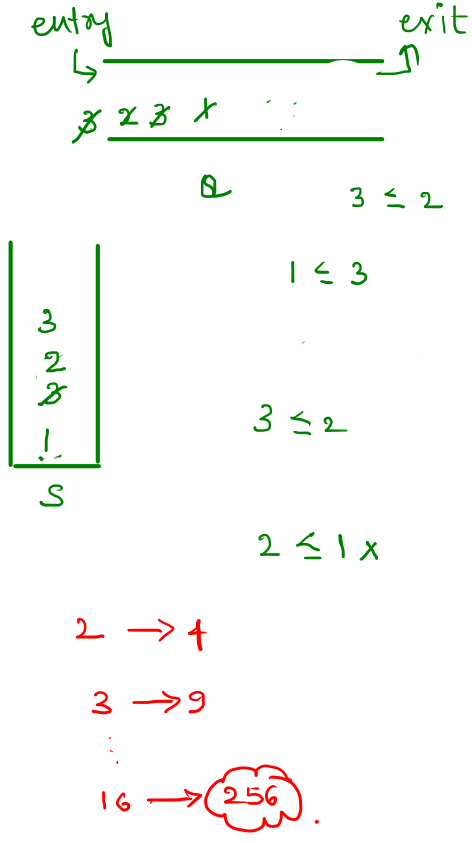
Let Q denote a queue containing sixteen numbers and S be an empty stack. Head(Q) returns the element at the head of the queue Q **without** removing it from Q. Similarly Top(S) returns the element at the top of S **without** removing it from S. Consider the algorithm given below.

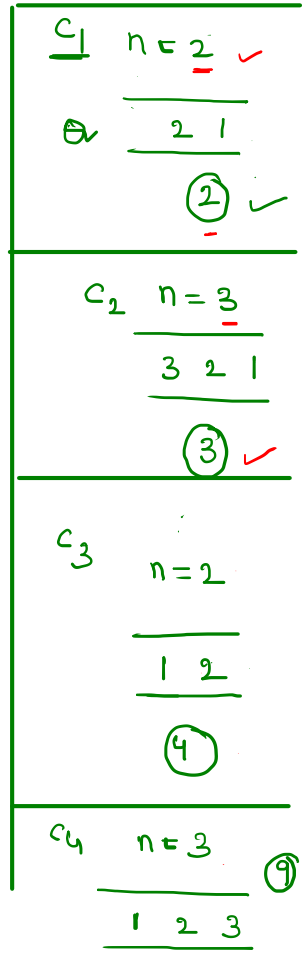
```
while Q is not Empty do
    if S is Empty OR Top(S) ≤ Head(Q) then
        x := Dequeue(Q);
        Push(S,x);
    else
        x := Pop(S);
        Enqueue(Q,x);
    end
end
```



The maximum possible number of iterations of the while loop in the algorithm is_____

```
while Q is not Empty do
    if S is Empty OR Top(S) ≤ Head(Q) then
        x := Dequeue(Q);
        Push(S,x);
    else
        x := Pop(S);
        Enqueue(Q,x);
    end
end
```



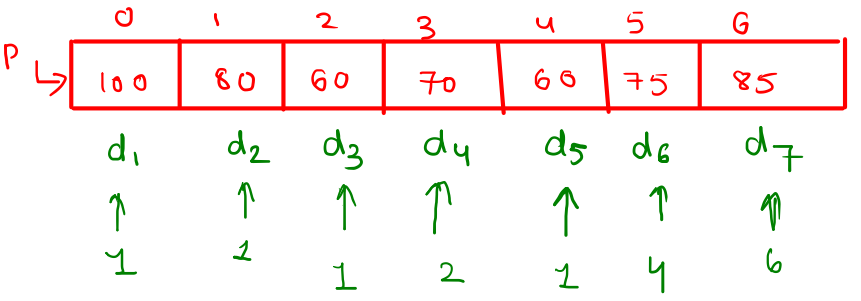


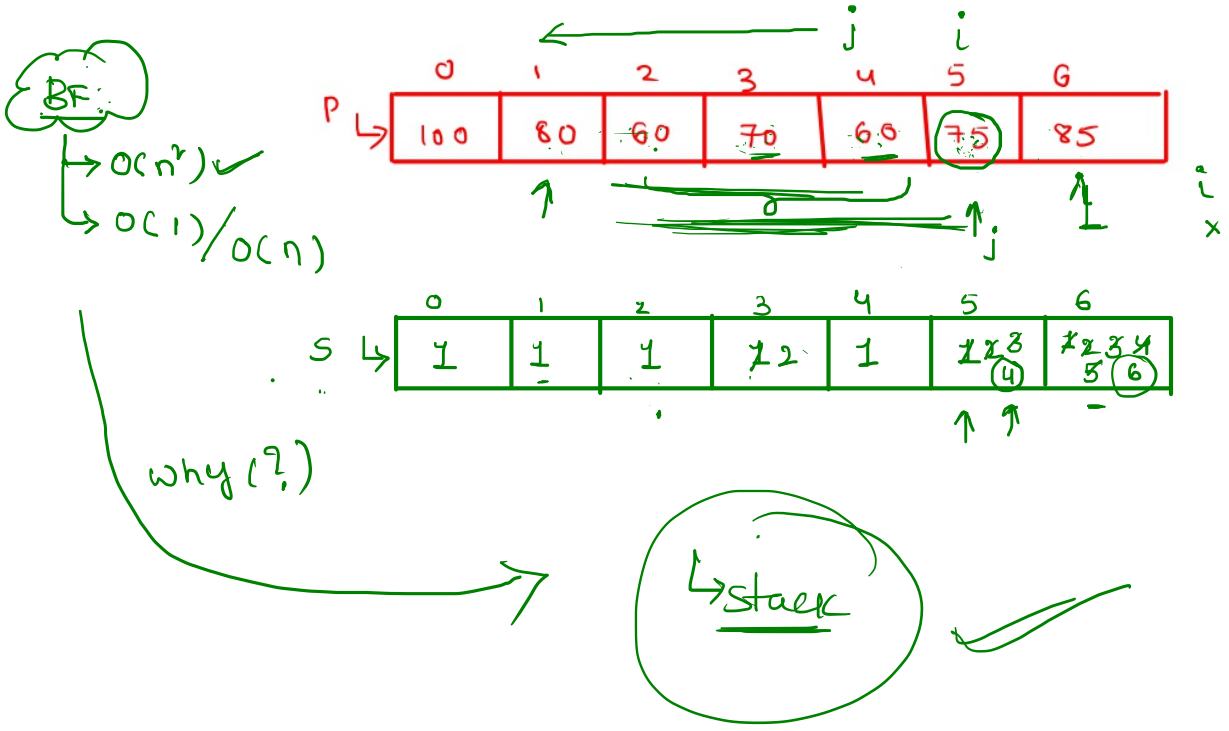
Stocks continious

1. Stock-Span Problem

p[]={ 100, 80, 60, 70, 60, 75, 85 }

s[] ={ 1, 1, 1, 2, 1, 4, 6 }





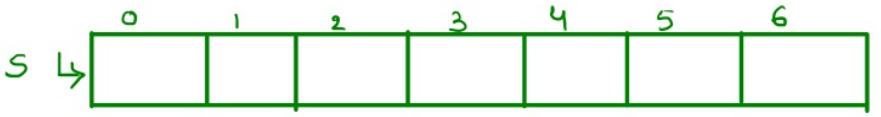
2 → loops

$loop_1$ → 0 L

i = 1 to n

$loop_2$ → 1 L

2 → Nested loops

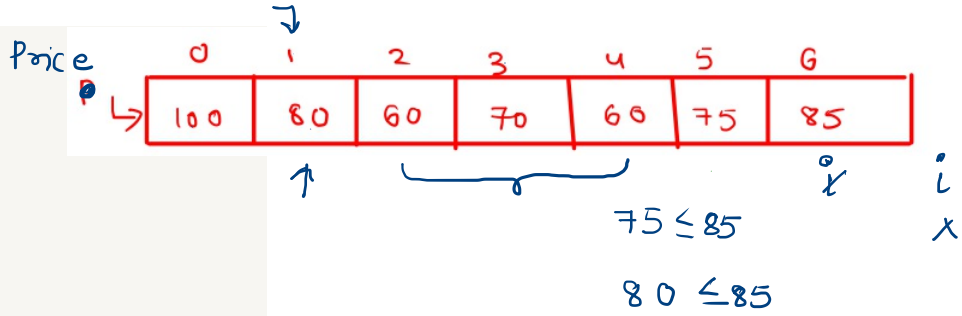

0
1
2
3
4
5
6
S

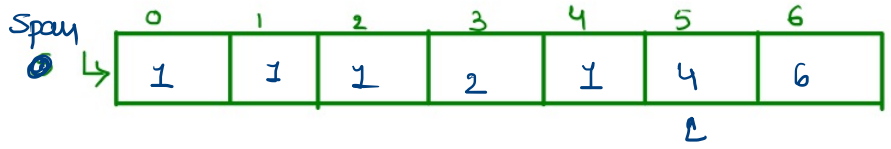
```
findSpan(price[],n,span[])
{
    Stack st
      st.push(0)
    span[0]=1
    for(i=1;i<n;i++)
    {
       while(!isEmpty() && price[st[top]]<=price[i])
          {
               ele=st.pop()
           }
          span[i]=isEmpty()? i+1 : i-st[top]
          st.push(i)
       }
}
```

→ O(n)

→ O(n)

n x n ⇒ n^2





EA → push, pop, top

→ push(10), push(5), push(2), push(6),

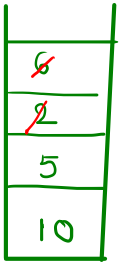
Design a stack such that getMin() is in O(1) O(1) space

getMin(). POP(), POP(), getMin()

↳ 2 ↳ 5

O(1)

10, 5, 2, 6, getMin()



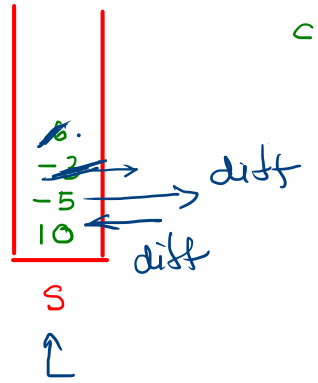
S_1

AP1

↳ take S_2

10, 5, 2, 6, getmin(), pop(), pop(), getmin()

2 5



curr_min = ~~10~~ ~~5~~ ~~2~~ 5

$-3 < 2$

$2-(-3)$
$= 5$

O(1)

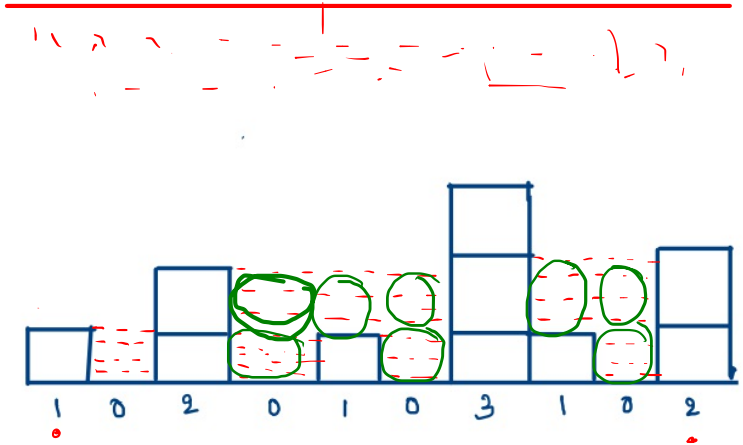
curr_min = curr_min - st.top()

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

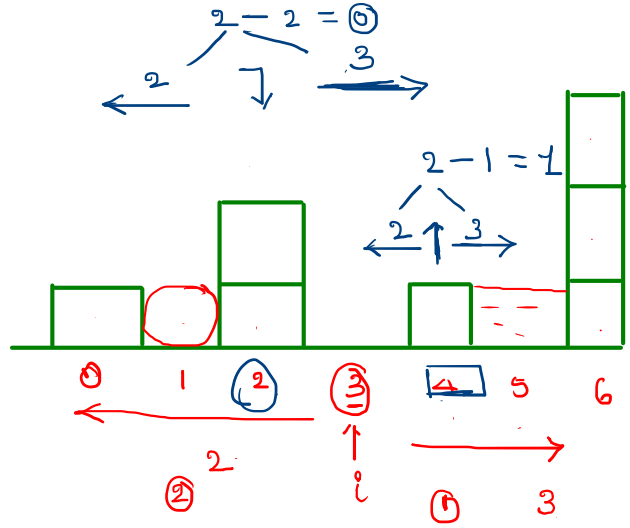
```
add(data)
{
  if(s.isEmpty())
  {
      s.push(data)
      curr_min=data
  }
  else
  {
       if(data<curr_min)
       {
            s.push(data-curr_min)
            curr_min=data
       }
       else
         s.push(data)
}
delete()
{
      if(s.peek()<curr_min)
      {
          curr_min=curr_min-s.peek()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

PBL (GIA)

Trapping Rain water problem



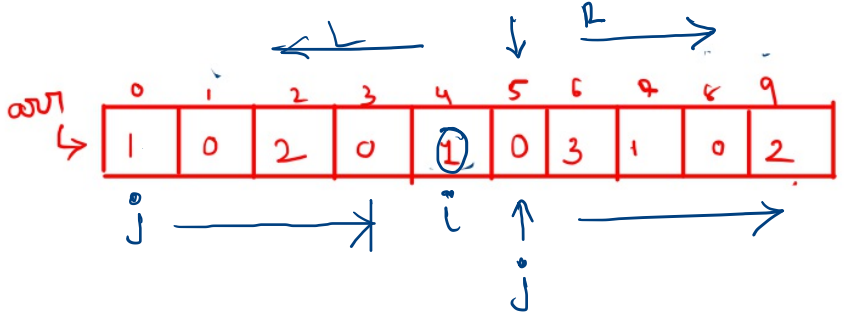
0 + 1 + 0 + 2 + 1 + 2 + 0 + 1 + 2 → 0



(3) min(3, 2) = 2

(4) 2 − arr[i] = 2 − 0 = 2

```
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;
}
```



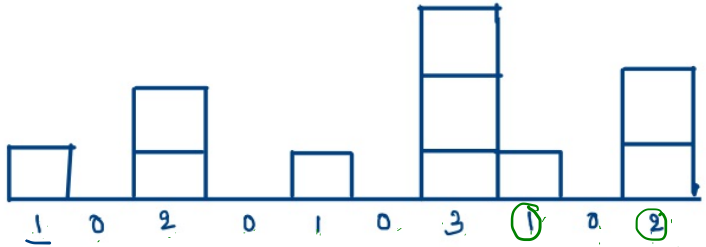
left = x · 2

right = x 3

$n[n+n] = n^2+n^2 = 2n^2$

$\therefore O(n^2)$ T.C

O(1) S.C



L → R

	0	1	2	3	4	5	6	7	8	9
Left →	1	1	2	2	2	2	3	3	3	3

R → L

	0	1	2	3	4	5	6	7	8	9
Right →	3	3	3	3	3	3	3	2	2	2

$\rightarrow loop_1$

$+$

$\rightarrow loop_2$

$+$

$\rightarrow loop_3$

T.C $\Rightarrow$ $n+n+n = 3n \Rightarrow O(n)$

S.C $\Rightarrow$ $n+n = 2n \Rightarrow O(n)$

```
res = 0
for(i=0; i<n; i++)
{
  res = res + ( min(left[i], right[i])
                - arr[i])
}
return res
```

```
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;
}
```