Content

→ Carry forward into

→ Solve problems.

Q. Count pairs 'ag'

Given an array of char[], cal no. of pairs (i,j) such that

$i < j$ && $s[i] = 'a'$ && $s[j] = 'g'$

Ex.

0	1	2	3	4	5	6	7
b	a	a	g	d	c	a	g

<2,3> <2,7>
<1,3> <6,7>
<1,7>
<6,3> ✗ i>j

ans = 5

Q.

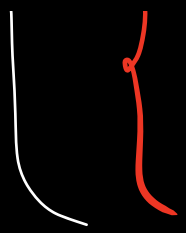
0	1	2	3	4	5	6
a	c	g	d	g	a	g

(0,2) <5,6>
(0,4)
(0,6)

ans = 4

Approach-1 Check all pairs.

```
count = 0
for(i = 0; i < n; i++)
{
    for(j = i+1, j < n; j++)
    {
        // i,j
```



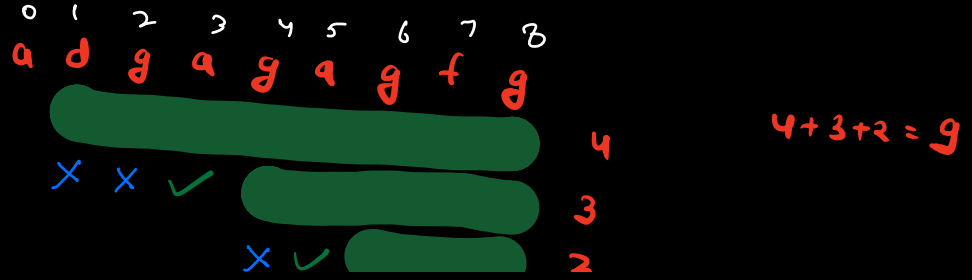
return count.

TC: O(N²)
SC: O(1)

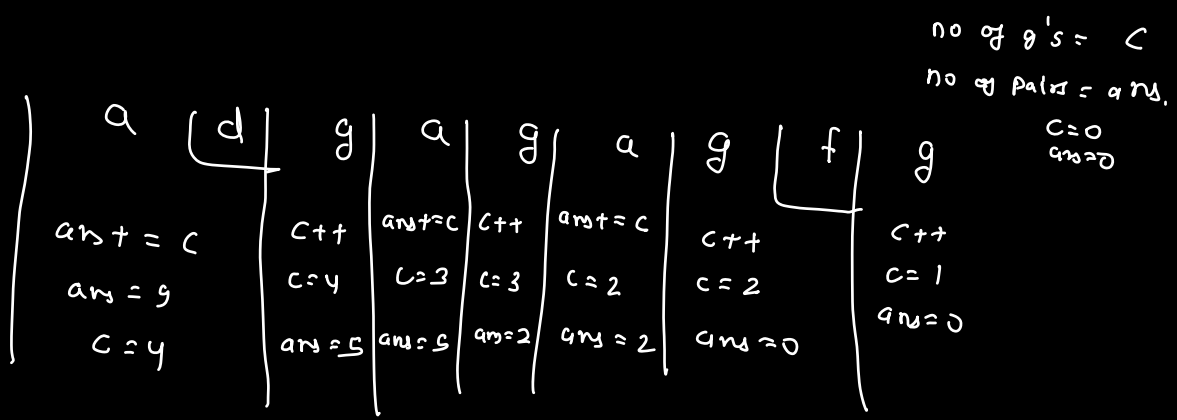
```
count = 0
for(i = 0; i < n; i++)
    if ( s[i] == 'a' )
        for( j = i+1, j < n; j++)
            // i,j
            if ( s[j] == 'g' ) count++
return count.
```

TC: O(N²)
SC: O(1)

Observation: for each 'a', count no of g's in right.



idea : Cal total no of g's from R~L.




```
c=0
ans=0
for(i=n-1 ; i>=0; i--)
{
    if( s[i]=='g' ]   c++
    if( s[i]=='a' ]   ans = ans+c
}
return ans.
```

Tc: O(N)
Sc: O(1)

idea2 : for every 'g', cnt no of 'a' in left

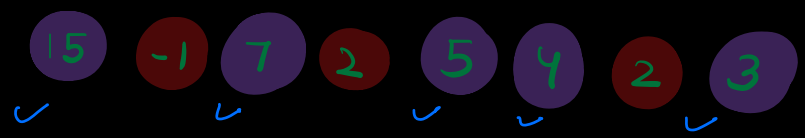
```
c= cnt of 'a' in left
for( i=0 ; i<n; i++
{
    if (s[i]=='a') c++
    if(s[i]=='g') ans+=c.
}
```

Q. Leaders in the array.

Given an arr, cal how many leaders are present.

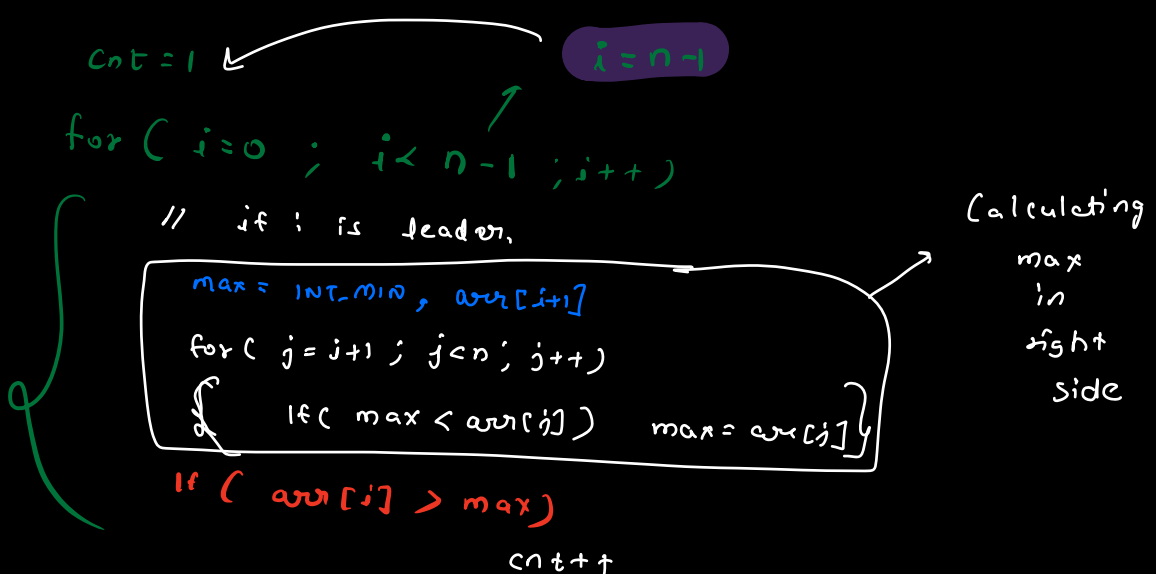
An ele is leader if it is strictly greater than all the elements in the right side.

Note : arr[N-1] is always leader.



Leader, if it is greater than max of right side.

Solution : for every element, check if it is a leader or not.



return cnt.

TC: $O(N^2)$

SC: $O(1)$

max in right cnt

15	-1	7	2	5	4	2	3
c=5	c=4	c=4	c=3	c=3	c=2	c=1	m=3
m=15	m=7	m=7	m=5	m=5	m=4	m=3	c=1

```
c = 1
m = arr[N-1]

for (i = n-2 ; i >= 0 ; i--)
{
    if (arr[i] > m)
    {
        c++
        m = arr[i]
    }
}

return c
```

TC: $O(N)$

SC: $O(1)$

Subarray Basics

1) Any continous part of arr is called subarray

→ A single is also subarray
→ Entire arr is also subarray
→ Empty can't be subarray.

	0	1	2	3	4	5	6	7	8
arr[9] =	-3	4	6	2	8	7	14	9	21

indices : [2,3,4,5] ✓

[3,4,6,7,8] ✗

[5] ✓

[0,1,2,.....8] ✓

[s, e] → indices [3, 6]

e - s + 1 = len

Usage of predefined functions

```
c = min(a, b)
```

TC: O(1)

```
a = min(9, 5)
a = 5
```

max(a, b) TC: O(1)

Q. Closest min max

Given an array. find the length of smallest subarray which contains both min & max ele of the array.

Ex

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

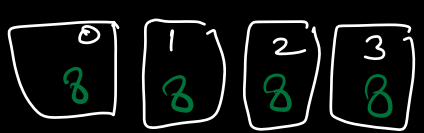
min = 1
max = 6

ans = 4

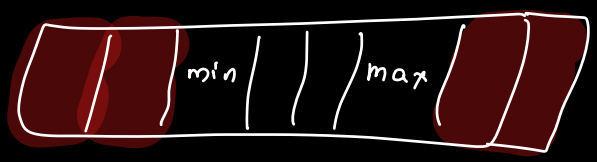
0	1	2	3	4	5	6	7	8	9	10
2	2	6	4	5	1	5	2	6	4	1

min = 1
max = 6

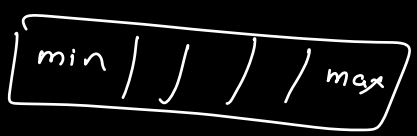
ans = 3



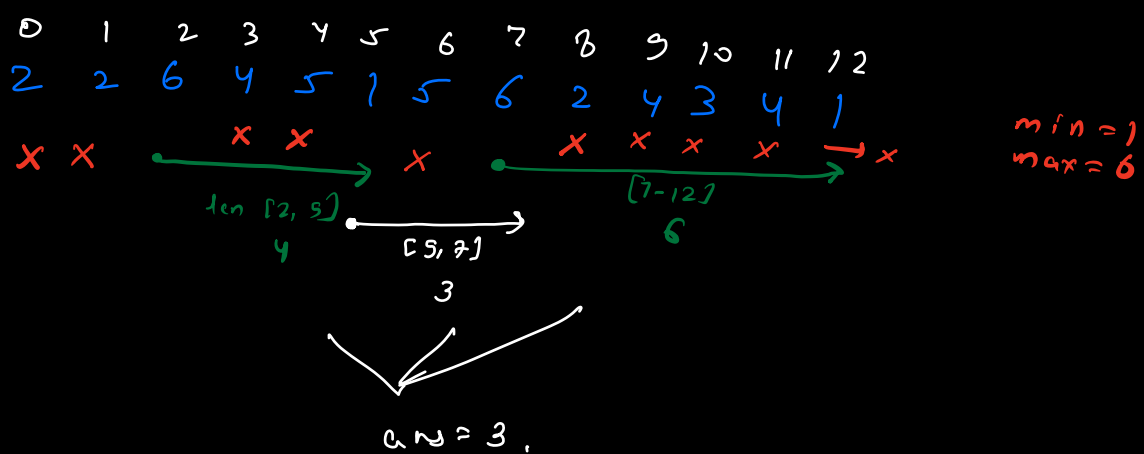
ans = 1



Obs: Find any subarray ; min & max should be on either end.



for each min, find nearest max
for each max, find nearest min
→ take best of these.



ans = 3.

mini → ToDo

maxi → ToDo

ans = INT_max / N / 0 / 1

```
for( i=0; i< n; j++ )
{
    if ( arr[i] == mini )
    {
        for(j=i+1; j<n; j++)
        {
            if(arr[j] == maxi)
            {
                // i, j
                ans = min(ans, j-i+1)
                Break
            }
        }
    }
    if ( arr[i] == maxi )
    {
        for(j=i+1; j<n; j++)
        {
            if(arr[j] == mini)
            {
                // i, j
                ans = min(ans, j-i+1)
                Break
            }
        }
    }
}
```

return ans;

ans, mini, maxi

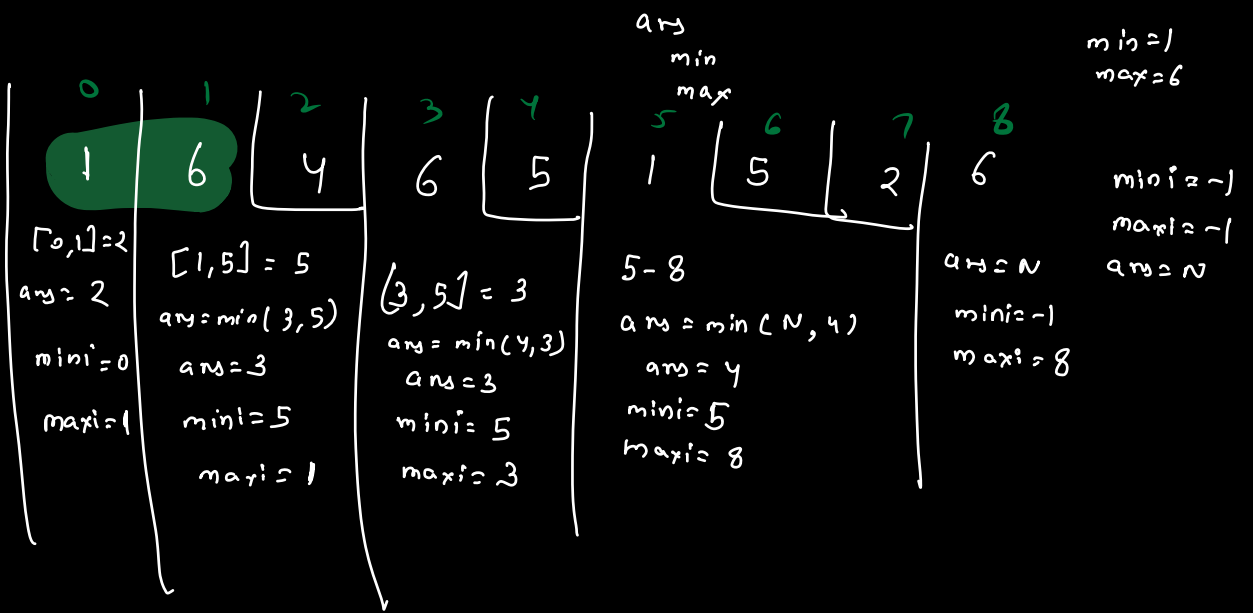
TC : O(N² + N + N) ≈ O(N²)

SC : O(1)

0 min 0 min 0 min 0 max 0 min 0 max

from right to left

maintain indexes min & max.



ans = 2

mini = -1
maxi = -1

minv → ToDo
maxv → ToDO

ans = N.

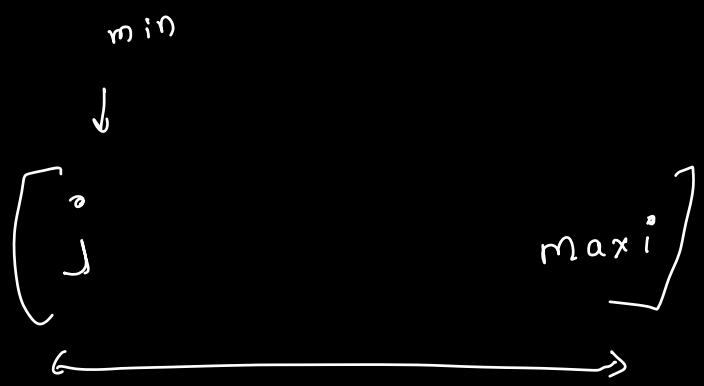
```
for( i = N-1 ; i >=0 ; i--)
{
    if( arr[i] == minv)
    {
        if( maxi != -1)
        {  len = maxi - i +1
           ans = min (ans, len)
        mini = i
    if( arr[i] = maxv)
    {
        if( mini != -1)
        {  len = mini - i +1
           ans = min (ans, len)
        maxi = i
```

0 1 2
5 5 5

i = 2

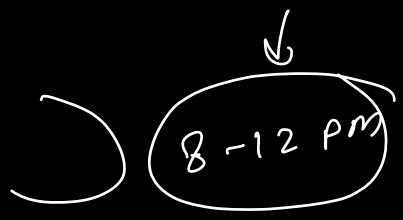
mini = 2
maxi = -1

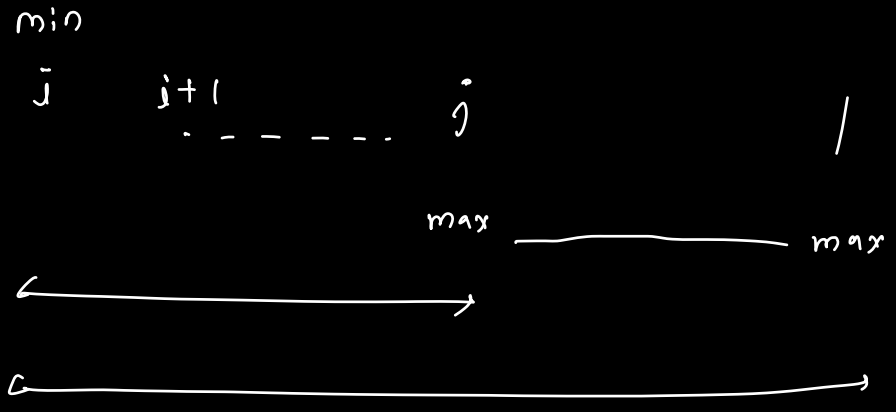
return ans;

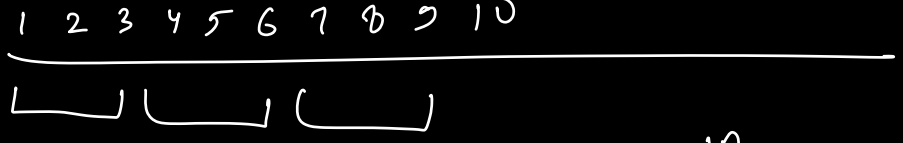


2013

len = maxi - j + 1







$10 - 3 - 3 - 3 = 1$

$\frac{10}{3} = 3$