Arrays: Carry forward

Question-1

Given a char array s, calculate # of pairs (i, j) such that i<j and s[i] = 'a' and s[j] = 'g'.

All chars are lower-case [a, z]

eg

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

(1,3) (2,3) (2,7) (6,7) (1,7) ans=5

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

(2,3) (2,4) (5,7) (6,7) (2,7) ans=5

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

(0,2) (0,4) (0,6) (5,6) ans=4

Iterate over every pair and check whether its valid or not.

```
ans =0
for (i=0; i<n; ++i) {
    for (j=i+1 ; j<n ; ++j) {
        if ( s[i]=='a' && s[j]=='g') {
            ans++
        }
    }
}
print(ans)
```

TC: $O(N^2)$

SC: $O(1)$

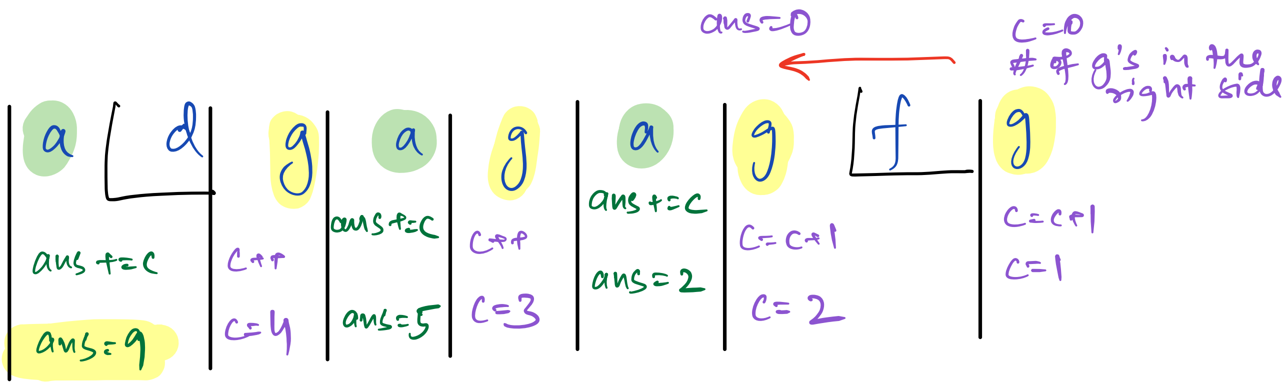
Observation 1 : Break if s[i] != 'a'

```
ans=0
for(i=0; i<n; ++i) {
    if (s[i] =='a') {
        for (j=i+1; j<n; ++j)) {
            if (s[j] == 'g')
                ans++
        }
    }
}
print(ans)
```

TC: $O(N^2)$

SC: $O(1)$

Observation 2 : We need count of g's in the right side of every a . finally sum all of them.



```
ans=0 , c=0

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

TC : O(N)

SC : O(1)

HW: Can you traverse from left to right?

Question 2 : Leaders in an Array

Given an Array A[N], you have to find count of leaders in array.

An element is a leader if it is strictly greater than entire right side.

Note: A[N-1] is always a leader.

eg 15 -1 7 2 5 4 2 3 count=5

←

10 7 9 3 2 4 count=3

8 -2 4 7 6 5 1 count=5

Code

```
leader = a[n-1]
 ans = 1
for ( i = n-2 ; i >= 0; --i) {
      if (a[i] > leader )
            ans ++
            leader = a[i]
}
print(ans)
```

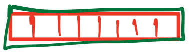
TC : O(N)

SC : O(1)

Subarray

Continous part of an array is called subarray

→ A single element is a subarray ✓


→ full array is a subarray ✓

→ Empty array is not subarray

eg a[9] =

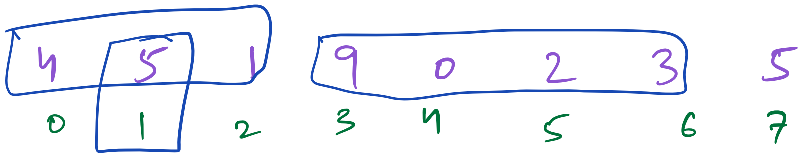
-3	4	6	2	8	7	14	9	21
0	1	2	3	4	5	6	7	8

indices [2,3,4,5] ⇒ subarray ✓

[3,4,6,7,8] ⇒ subarray ✗

[1,2,3] ⇒ ✓

[5] ⇒ ✓



[5] ✓

[4,5,1,0] ✗

[9,0,2,3] ✓

[4,5,1] ✓

If I have a subarray from index i to index j

Can I write → [i,j] YES

because all indices are contions.

Length of a subarray [i,j] ⇒ $j-i+1$

[1,2,3,4] ⇒ 4

[1,4] ⇒ 4

You can use these pre-defined fuctions

→ min(a,b) TC: O(1) SC: O(1)

→ max (a,b) TC: O(1) SC: O(1)

→ sort ( ) array TC: O(N logN) SC: O(N)

Closest Min Max

Given an array, find the length of the smallest subarray which contains both Min & Max of array.

size=4

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

Min = 1
Max : 6

[3,6] len = 6-3+1 = 4

2 2 6 4 5 1 5 2 6 4 1

Min=1
Max=6

ans = 3

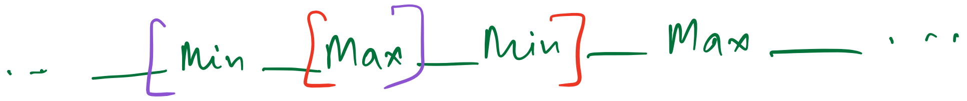
8 8 8 8 8

Min= 8
Max=8

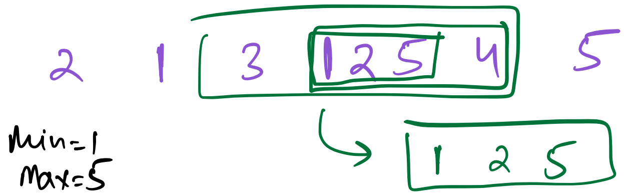
ans = 1

Observation

1. We only need to have 1 min and 1 max.



2. Min & Max should be present at corners?

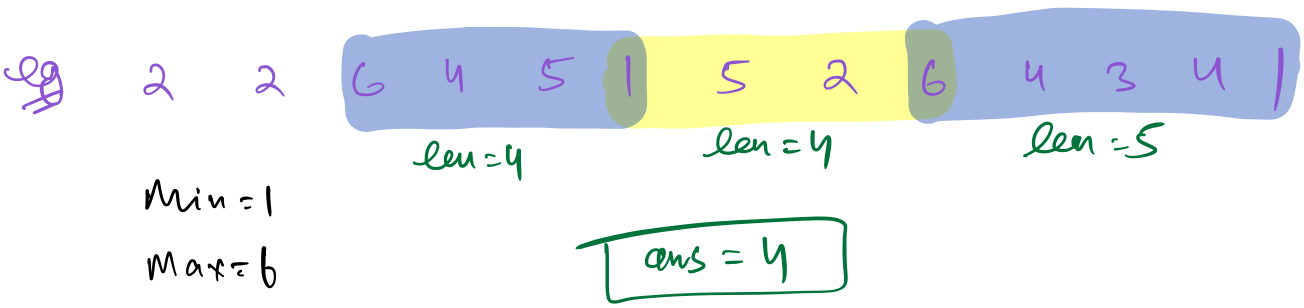


You can shrink your subarray until max & min are not at corners.

3. 2 cases:

→ [Min . . . . Max] → for every min value, give me the closest max value in right.

→ [Max . . . Min]



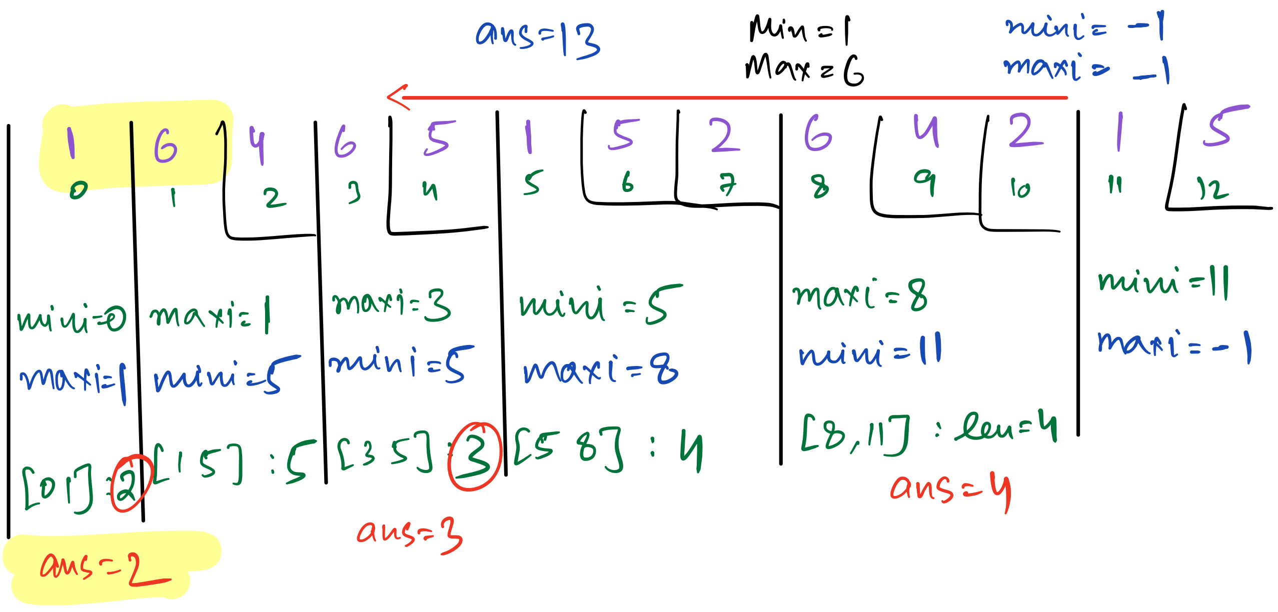
Bruteforce

```
ans = N
// iterate & get Min & Max
if (Min == Max)
    return 1

for (i=0; i<n; ++i) {
    if (a[i] == Min) {
        for (j=i+1; j<n; ++j ) {
            if (a[j] == Max)
                ans= min(ans, j-i+1)
                break
        }
    }
    else if (a[i] == Max) {
        for (j=i+1; j<n; ++j ) {
            if (a[j] == Min)
                ans= min(ans, j-i+1)
                break
        }
    }
}
return ans
```

TC: $O(N^2)$

SC: $O(1)$



Code

```
// iterate & find Min & Max value → TC: O(N)
                                   SC: O(1)
if ( Min == Max)
      return 1
mini =-1 , maxi =-1 , ans = N

for (i=n-1 ; i>=0; --i ) {
    if (a[i] == Min) {
        mini = i
        if (maxi != -1)
            ans = min ( ans, maxi-mini+1 )
    }
    elif ( a[i] == Max ) {
        maxi = i
        if (mini != -1)
            ans = min (ans , mini-maxi+1)
    }
```

TC: O(N)

SC: O(1)

```
}
 return ans
```