Arrays: Carry forward

Question

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

ans = 5

(2,3), (6,7), (1,3), (1,7), (2,7)

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

ans = 5

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

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

ans = 4

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

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
    }
}
```

TC : $O(N^2)$

SC : $O(1)$

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

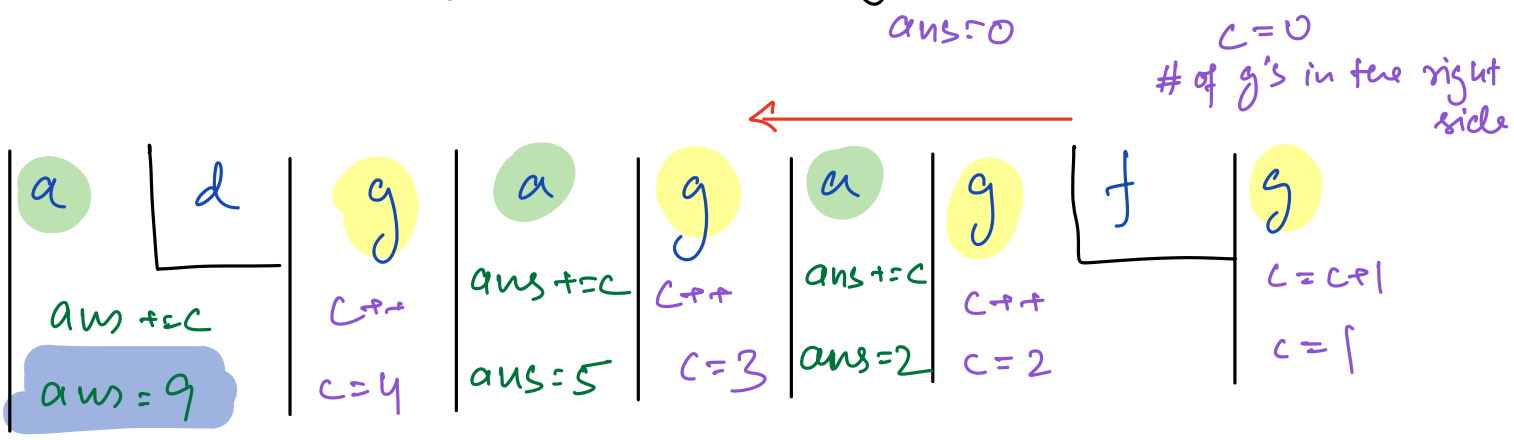
```
ans = 0
for (i=0; i<n; ++i) {
    if (s[i] == 'a') {
        for (j = i+1; j<n; ++j) {
            if (s[j] == 'g')
                ++ans
        }
    }
}
```

TC : $O(N^2)$

SC : $O(1)$

HW : try using suffix array

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++
    else if (s[i] == 'a')
        ans += c
}
print(ans)
```

TC : O(N)

SC : O(1)

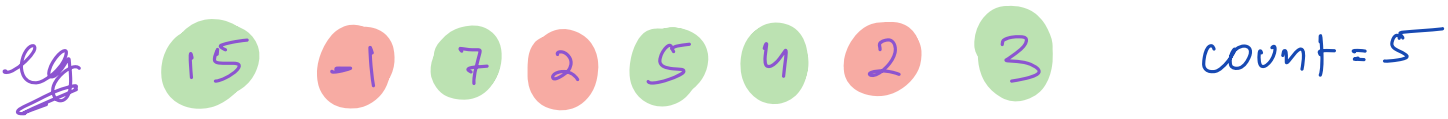
HW: Can you traverse from left to right?

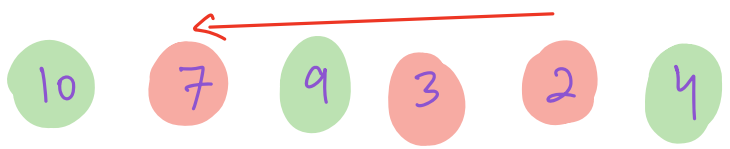
Question 2 Leaders in an Array

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.





Code

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

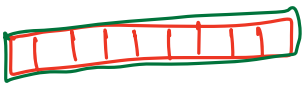
TC : O(N)

SC : O(1)

Subarray

Continous part of an array is called subarray.

 → single element is a subarray

 → full array is a subarray

→ empty array is not subarray

eg indices:

[2,3,4,5] ⇒ subarray ✓

[3,4,6,7,8] ⇒ subarray ✗

[1,2,3] ⇒ subarray ✓

[5] ⇒ subarray ✓

4 5 1 9 0 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 subarray as ⟶ [i,j] ✓

because all indices are included in [i,j]

length of a subarray [i,j] ⇒ j-i+1

[1,4] ⇒ 4-1+1 = 4

1,2,3,4

You can use these pre-defined functions

→ min(a,b) TC: O(1) SC: O(1)

→ max(a,b) TC: O(1) SC: O(1)

→ sort() array TC: O(NlogN) SC: O(N)

BREAK: 8:05 - 8:15

Question 3 Closest Min Max

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

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

[3,6] → smallest subarray

min = 1
max = 6

length = 6 - 3 + 1 = 4

2 2 6 4 5 1 5 2 6 4 1

min = 1
max = 6

length = 3

8 8 8 8

min = 8
max = 8

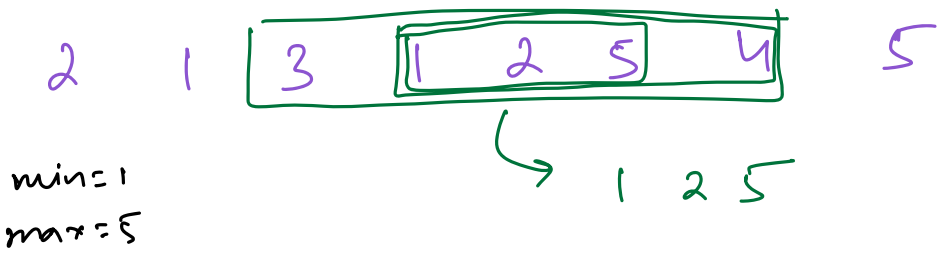
length = 1

Observation

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

If you have more than 1 Min or 1 Max value in the answer, then you can always shrink the subarray.

2. Min & Max are always present at corners.

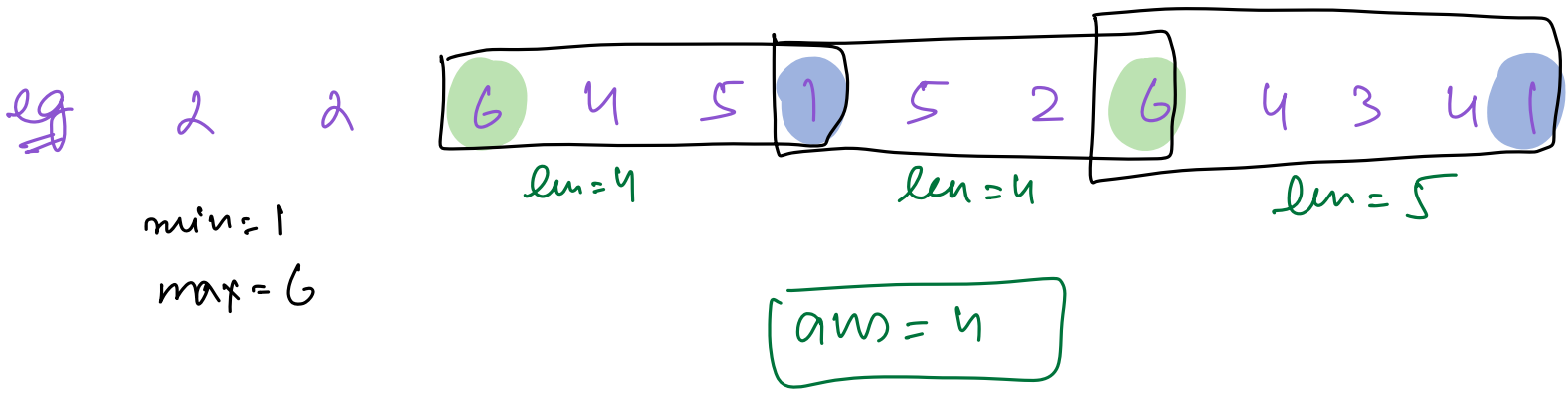


You can shrink your subarray until max & min are not at corners.

3. 2 cases:

→ [min ..... Max] → for every min value, find the closest max value in the right.

→ [Max ..... Min]



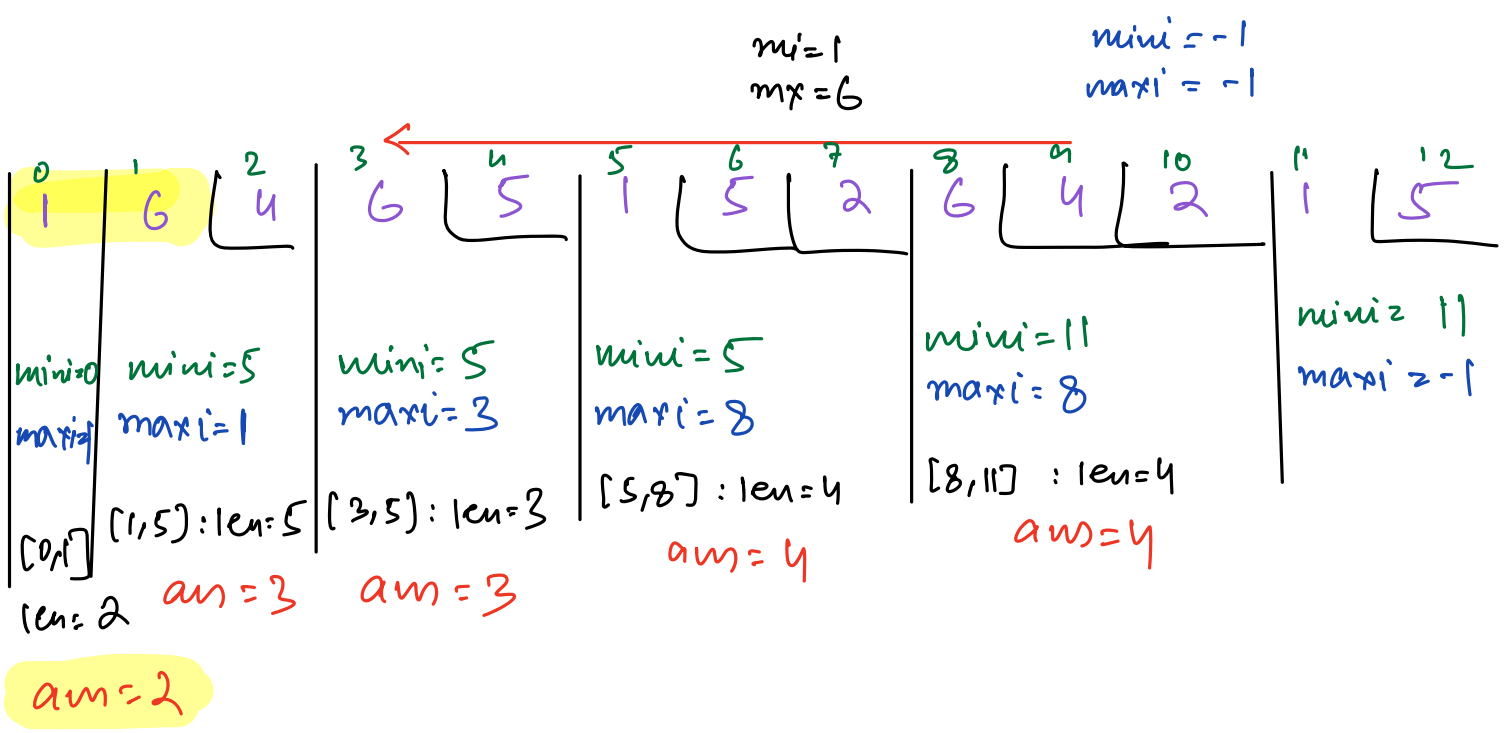
Bruteforce

```
ans = N
// iterate & get min & max  -> TODO
if ( mi == mx )
    return 1

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

mini=-1, maxi=-1, ans=N

for(i=n-1; i>=0; --i){
    if(a[i]==mi){
        mini=i
        if(maxi != -1)
            ans = min(ans, maxi-mini+1)
                      abs(maxi-mini)+1
    }
    else if(a[i]==mx){
        maxi=i
        if(mini != -1)
            ans = min(ans, mini-maxi+1)
                      abs(maxi-mini)+1
    }
}
```

TC:O(N)
SC:O(1)

```
}
return ans
```