Q1. Given a char[], calculate the no. of pairs [i, j], such that i < j, && s[i] == 'a', && s[j] == 'g', all characters are lowercase alphabets [a b c d —].

eg ⇒ arr ⇒ b a a g d c a g

0 1 2 3 4 5 6 7

pair : <1, 3> <2, 3> <6, 7>

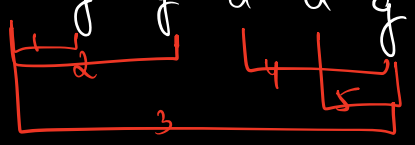
<1, 7> <2, 7>

~~<3, 6>~~ → ~~not valid~~.

g a

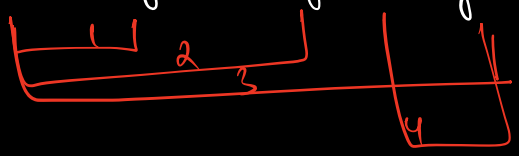
ap = 5.

eg ⇒ arr ⇒ b c a g g a a g



ap = 5.

eg ⇒ arr ⇒ a c g d g a g



ap = 4.

Brute force

** check for all possible pairs :-

```
c=0
for(i=0; i<N; i++){
    for(j=i+1; j<N; j++){
        if(arr[i]=='a' &&
           arr[j]=='g')
            c++
    }
}
print(c);
```

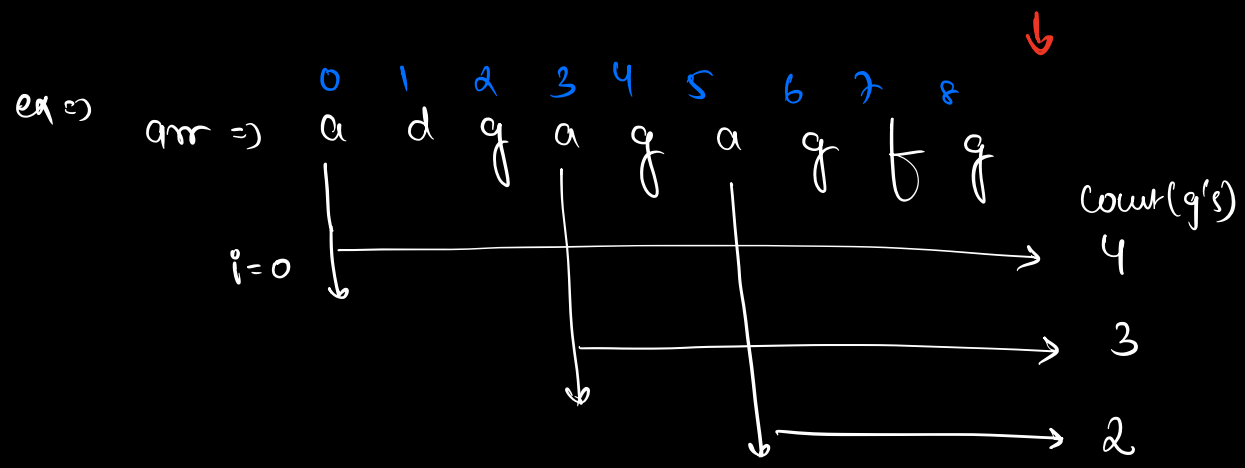
TC ⇒ O(N²)
SC ⇒ O(1)

** If S[i] != 'a', then second loop is not required :-

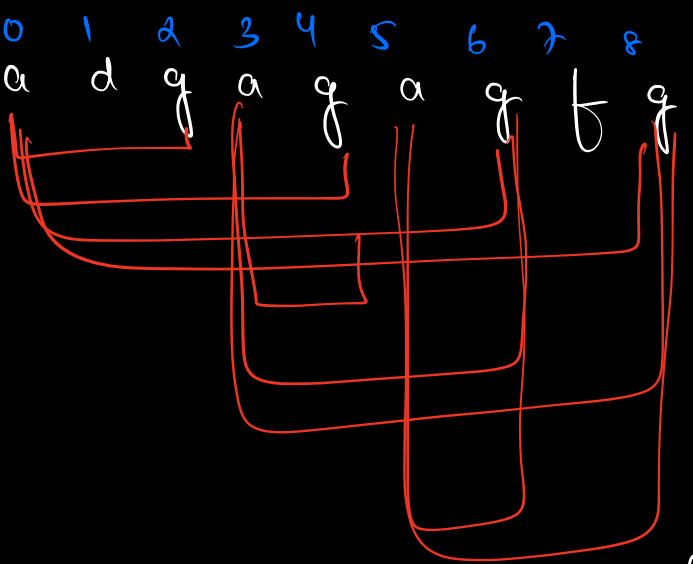
```
c=0
for(i=0; i<N; i++){
    if(arr[i]=='a'){
        for(j=i+1; j<N; j++){
            if(arr[j]=='g')
                c++
        }
    }
}
print(c);
```

TC ⇒ O(N²)
SC ⇒ O(1)

eg =>

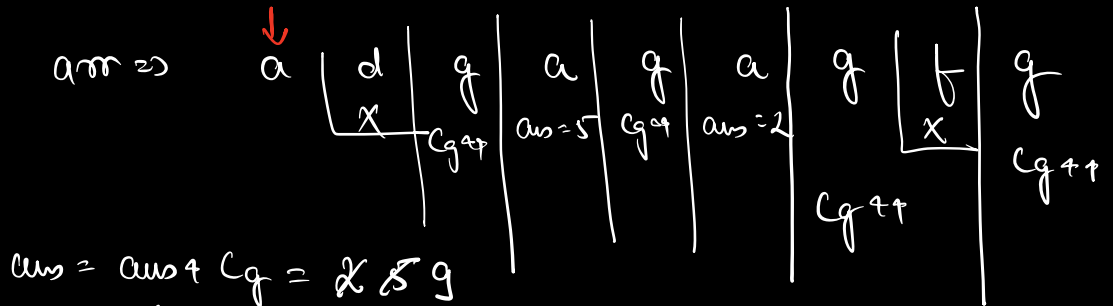


Op = g (4+3+2)



=> new approach :-

ans = 0
c_g = 0.



ans = ans + c_g = x x g
c_g = ø x x x 4

=> Op = g

TC ⇒ O(N)
SC ⇒ O(1).

⇒ Carry forward technique

pseudo

```
ans = 0   cg = 0.
for(i = N-1; i >= 0; i--){
    if(arr[i] == 'g'){
        cg++;
    }
    else if(arr[i] == 'a'){
        ans = ans + cg.
    }
}
print(ans).
```

soln-2 For every 'g' calculate 'a' to the left of 'g'.

Carry → C_a, ans

(left to right)

i = 0 → i = N-1.

Q2. Leaders in an array :

Given an arr[N], you have to find all leaders in arr, an element is known as a leader, if it is strictly greater than all elements on its right side.

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

ex ⇒ arr[] ⇒ 15 -1 7 2 5 4 2 3

✓ ✗ ✓ ✗ ✓ ✓ ✗ ✓

Ans = 5

ex ⇒ arr[] ⇒ 8 -2 4 7 6 5 1

✓ ✗ ✗ ✓ ✓ ✓ ✓

Ans = 5.

ex ⇒ arr[] ⇒ 10 7 9 3 2 4 5

✓ ✗ ✓ ✗ ✗ ✗ ✓

Ans = 3.

ex ⇒ arr[] ⇒ 10 8 8

✓ ✗ ✓

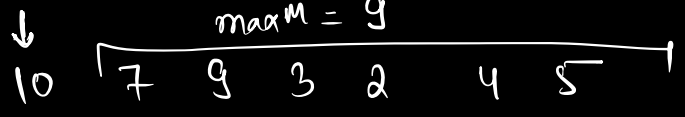
Ans ⇒ 2

Bruteforce ⇒ take each element, and iterate till end, and check if its greater than all elements.

TC ⇒ $O(N^2)$ How ? ⇒ pseudo code for bruteforce.

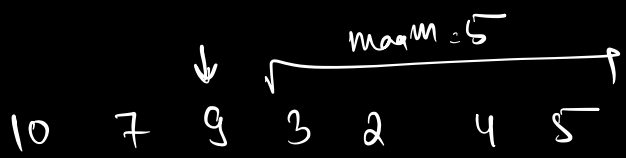
SC ⇒ ?

Otherwise



$10 > 9 \rightarrow$ 10 is leader, ans++.

$7 < 9 \rightarrow$ 7 is not a leader.



$9 > 5 \rightarrow$ 9 is a leader, ans++.

Obs:- If for index i, max is the largest element on right of i [i+1 to N-1], and if

arr[i] > max ⇒ ans++

leader.

ans = 2 3
max = ~~9~~ 10

10 7 9 3 2 4 5

Ans = 3

pseudo

```
c = 1, max = arr[N-1]
for( i = N-2; i >= 0; i-- ){
    if( arr[i] > max ){
        c++;
        max = arr[i]
    }
}
print(c).
```

TC ⇒ O(N)
SC ⇒ O(1)

: Subarrays :- [Basics]

** Continous part of an array is called subarray.

i) a single element is a subarray
ii) entire array is a subarray
iii) empty can be a subarray

eg ⇒ arr[9] ⇒

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

idx ⇒ [2 3 4 5] ⇒ Yes → [2-5]
idx ⇒ [3 4 6 7 8] ⇒ No → 5 is skipped

idq ⇒ [1 -3] ⇒ Yes.

idq ⇒ [4] ⇒ Yes

idq ⇒ [0 -8] ⇒ Yes.

[] ⇒ No

eg ⇒ subarrays [3 - 7] ⇒ 3 4 5 6 7

7 - 3 + 1 = 5

5 elements

sub [s - e] ⇒ e - s + 1

predefined func:

→ min(a,b) → returns minm of a & b.

TC ⇒ O(1)

→ max(a,b) → returns maxm of a & b

TC ⇒ O(1)

→ sort(arr) → arrange the elements in asc order by default;

TC ⇒ O(NlogN)

: always maintain TC of inbuilt func. in your overall TC

Q 3. Closest min max

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

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

min = 1
max = 6

idx ⇒ [0 8] ⇒ len = 9
[3 8] ⇒ len = 6
[3 6] ⇒ len = 4. → minm.

Ans = 4

eg ⇒ arr[] ⇒

	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

idx ⇒ [1 7] ⇒ 7.
idx ⇒ [2 5] ⇒ 4
idx ⇒ [8 10] ⇒ 3 → minm

Ans = 3

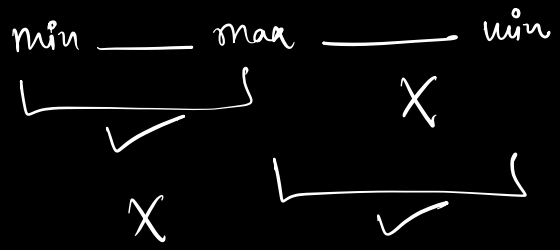
eg ⇒ arr ⇒ 8 8 8 8 8 8

min = 8
max = 8

sub ⇒ 1. (Ans)

Observation

I)



Shortest subarray will contain only 1 min & 1 max

II)

min & max would always be at boundaries.
[two ends of Sub array]

III) Combining (I) & (II):

Case 1 → min ——— max

↓

if I am standing at min, find next max at right

Case 2 → max ——— min

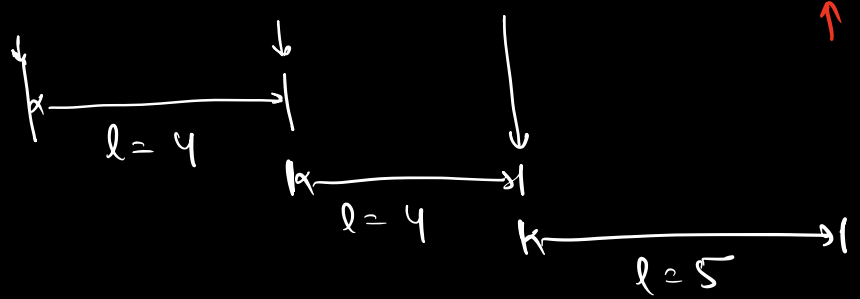
↓

if I am standing at max, find next min at right

eg ⇒

0	1	2	3	4	5	6	7	8	9	10	11	12
2	2	6	4	5	1	5	2	6	4	3	4	1

min = 1
max = 6

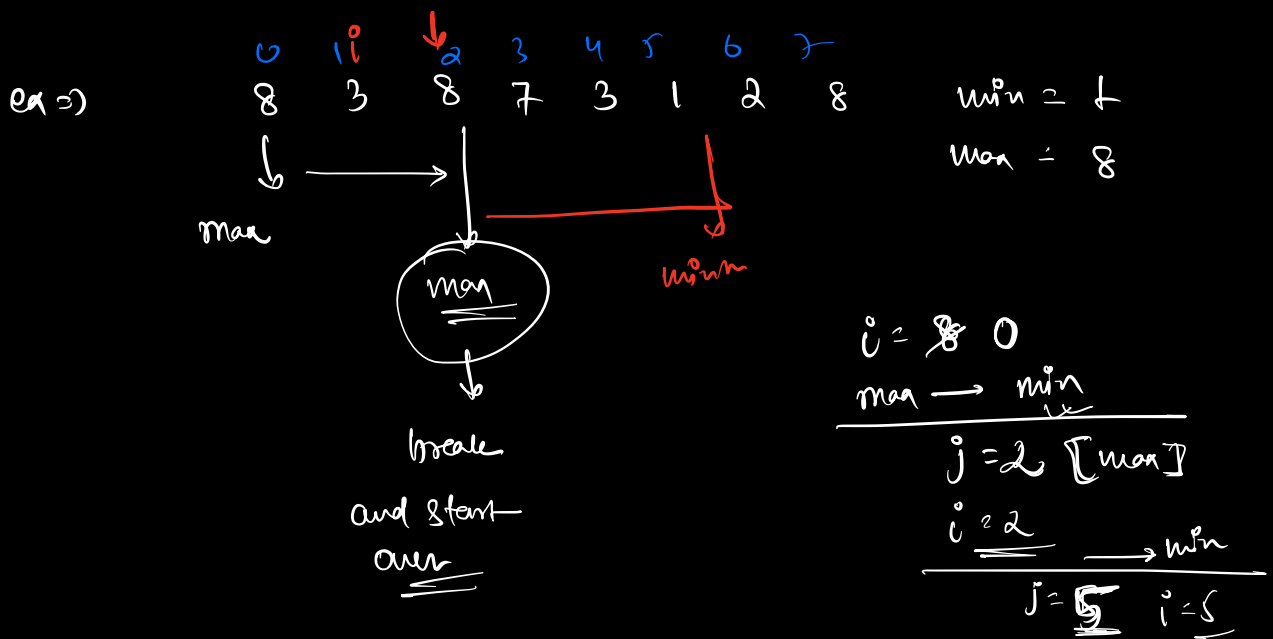


op : minm leng ⇒ 4.

Pseudo

ans = N.
iterate, get min & max] if(min == max) return 1.

```
for( i=0; i<N; i++){
    if(arr[i] == min){ // find nearest maxm
        for(j= i+1; j<N; j++){
            if(arr[j] == max){
                ans = min(ans, j-i+1);
                break;
            }
        }
}
else if(arr[i] == max){ // find nearest min
    for(j= i+1; j<N; j++){
        if(arr[j] == min){
            ans = min(ans, j-i+1);
            break;
        }
    }
}
```



So, if you are at max, finding min but got condition

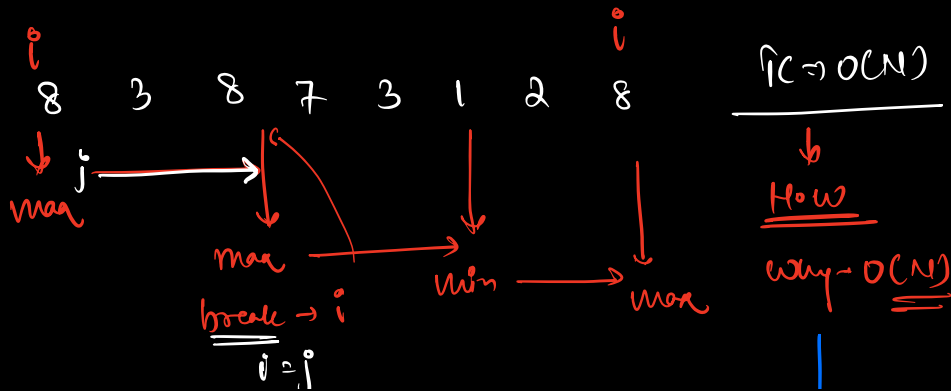
max ⇒ break

So, if you are at min, finding max but got condition

min ⇒ break

TC ⇒ $O(N^2)$

SC ⇒ $O(1)$



=> carry forward min = 1, max = 6

ans = N
minL = -1
maxL = -1

0	1	2	3	4	5	6	7	8	9	10	11	12
1	6	4	6	5	1	5	2	6	4	4	2	1
↑		X		X		X	X		X	X	X	minL = 12
minL = 0	minL = 5	maxL = 1	minL = 5	maxL = 3	minL = 5	maxL = 8		maxL = 8	minL = 12			maxL = -1
maxL = 1	maxL = 1		maxL = 3		maxL = 8			minL = 12				
L = 2	L = 5		L = 3		L = 4			L = 5				

ans = ~~13 5~~ ~~4~~ ~~3~~ 2.

ans = 2

TC => O(N)
SC => O(1)

pseudo

```
ans = N
minI = -1
maxI = -1

1) iterate ⇒ find min & max

for( i = N-1 ; i >= 0 ; i-- ) {
    if ( arr[i] == min ) {
        minI = i ;
        l = abs( minI - maxI ) + 1
        ans = min( ans, l )
    }
    else if( arr[i] == max ) {
        maxI = i ;
        l = abs( minI - maxI ) + 1
        ans = min( ans, l )
    }
}

print(ans)
```

check if maxI != -1

check if minI != -1