Q1. Given a char[], calculate 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

valid pairs :- <1, 3> <2, 3> <6, 7>

<1, 3> → a g

<1, 7> <2, 7>

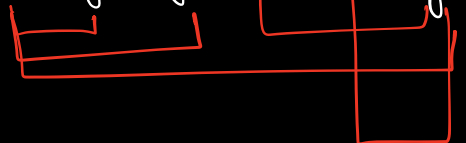
<3, 6> → g a

<6, 3> → a g → i < j

ans = 5.

eg ⇒ arr ⇒

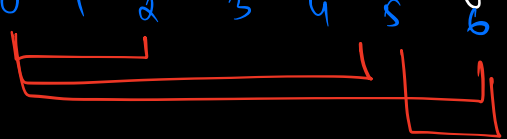
0	1	2	3	4	5	6	7
b	c	a	g	g	a	a	g



ans = 5.

eg ⇒ arr ⇒

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



ans = 4.

Brute force

Check all possible pairs of (i, j).

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

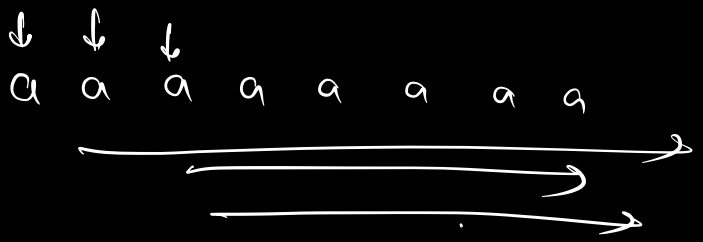
TC ⇒ O(N²)
SC ⇒ O(1)

* if s[i] != 'a', then second loop is not required.

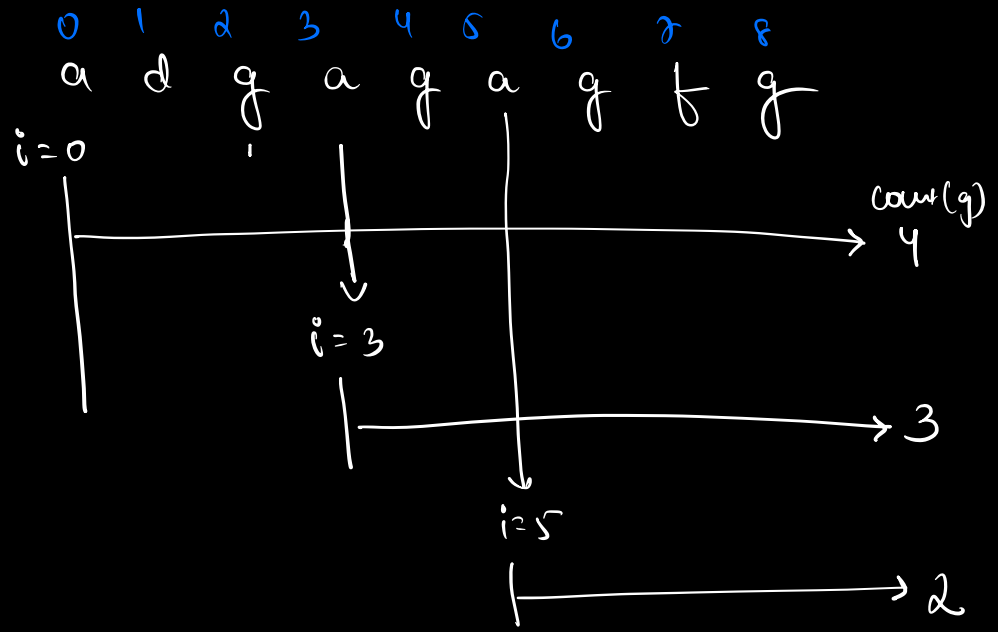
```
for(i=0; i<N; i++){
    if(s[i]=='a'){
        for(j=i+1; j<N; j++){
            if(s[j]=='g')
                c++
        }
    }
}
print(c)
```

Still, TC ⇒ O(N²)
SC ⇒ O(1)

worst case



ex =>



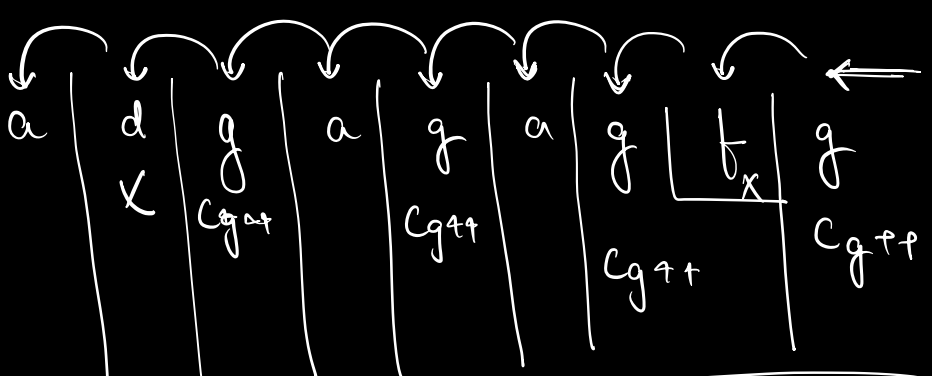
ans = 9.

=> what is redundant?

4 + 3 + 2 = 9

Counting of g is redundant

Carry forward technique



ans = ans + cg = 9

cg = 4

ans = 9

TC => O(N)

SC => O(1)

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)
```

TC ⇒ O(N)
SC ⇒ O(1)

idea2

For every 'g' calculate no. of 'a' on the (left - right) side :-

what to carry :-

1) ans
2) count of a (Ca)

→ left to right

How → pseudo code

Q2. Leaders in an array :-

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

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

eg ⇒ arr[] ⇒ 15 -1 7 2 5 4 2 3

✓ ✗ ✓ ✗ ✓ ✓ ✗ ✓

Op = 5.

arr[] ⇒ 10 7 9 3 2 4 5

✓ ✗ ✓ ✗ ✗ ✗ ✓

Op = 3.

arr[] ⇒ 8 -2 4 7 6 5 1

✓ ✗ ✗ ✓ ✓ ✓ ✓

Op = 5.

arr[] ⇒ 10 8 8

✓ ✗ ✓

Op ⇒ 2.

Brute force

take each element, and compare to all elements toward right, and check if it's leader

TC ⇒ $O(N^2)$ SC ⇒ $O(1)$

Pseudo ⟶ How

Optimised

10 7 9 3 2 4 5

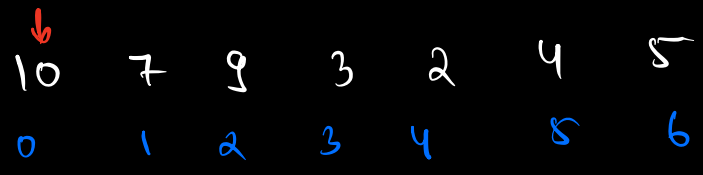
9 > 5 → 9 is a leader

3 < 5 → 3 is not a leader.

for any idx i, need to know the maxm from (i+1 to N-1)

```
if (arr[i] > maxm)
```

C++



count = ~~1~~ 2 3 ⟶ ans = 3

maxm = ~~5~~ 9 10

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

1) a single element is a subarray

eg ⇒ arr[] = [1 4 3 2 5]

[4], [3], [2] - - - subarrays

2) entire array is also a subarray

eg ⇒ arr[] = [1 4 3]

also a subarray.

3) 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

⇒ pdx ⇒ [2 3 4 5] ✓ ⇒ [2-5]
⇒ idx ⇒ [3 4 6 7 8] XX → 5 is skipped
⇒ idx ⇒ [1-3] → ✓
⇒ idx ⇒ [4] → ✓
⇒ pdx ⇒ [5-5] → ✓

[] → XXX.

subarray ⇒ [3-7] ⇒ 3 4 5 6 7 → 5 elements

↓

7-3+1 ⇒ elements.
5

subarray ⇒ [s-e] ⇒ length = e-s+1

⇒ Usage of predefined func:-

⇒ min(a,b) → returns the minm of a & b.
TC ⇒ O(1)

⇒ max(a,b) → returns the maxm of a & b
TC ⇒ O(1)

⇒ sort(arr) → arranges the array into asc order by default.
TC ⇒ O(NlogN)

** always consider TC of predefined func in overall TC.

→ find the minm no. of an array:-

```
min = arr[0].
for(i=0; i<N; i++) {
    min = min(arr[i], min)
}
}.
```

```
if(arr[i] < min) {
    min = arr[i].
}
```

Q 3. Closest Min Max

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

ex ⇒ arr ⇒

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

minm = 1, maxm = 6.

idx ⇒ [0 8] = 9 (len)

[3 8] = 6.

[3 6] = 4 ⟶ Shortest ⇒ ans = 4

ex ⇒ arr ⇒

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

min = 1

max = 6

idx		len
	[1 7] ⇒	7
	[2 5] ⇒	4
	[8 10] ⇒	(3) ⟶ ans = 3

ex ⇒ arr[] = 8 8 8 8 8 8 8

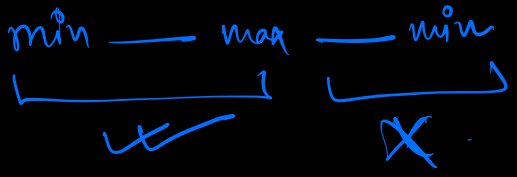
min = 8

max = 8

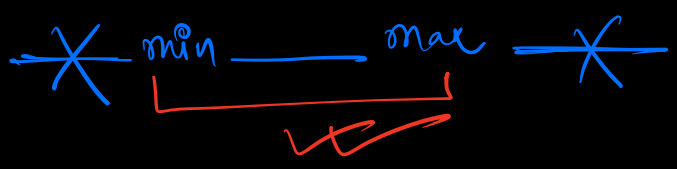
ans = 1.

Observations

1) Our final shortest subarray can only contain 1 min & 1 max



11) min & max should always be at boundaries.



111) from pt 1 & pt 2, there are only 2 possibilities

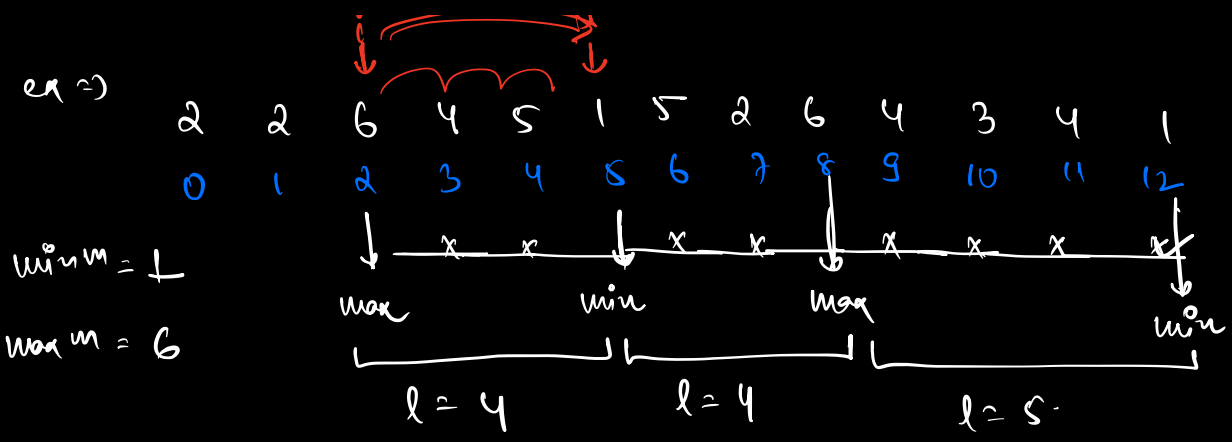
Case 1 → [min ——— max]

if standing at min, find the nearest max on right side.

Case 2 → [max ——— min]

if standing at max, find the nearest min on right side.

eg =>



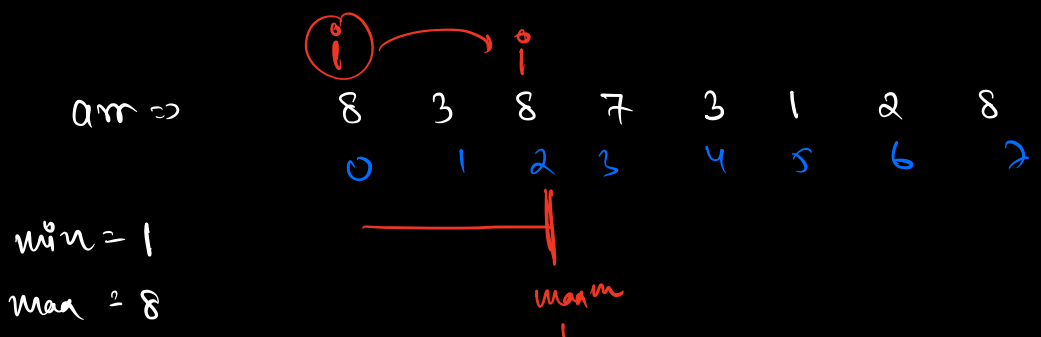
Op => min len = 4

pseudo

```
ans = N.
iterate and get maxm → max
iterate and get minm → min
for( i=0; i<N; i++){
    if( arr[i] == min){ // find nearest max
        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
    }
}
}
```

TC ⇒ $O(N^2)$
SC ⇒ $O(1)$



break and start again to find min, i = 2

update breakpoint pseudo code with this code

⇒ if starting from max to find min,
and we encounter a max → break & start again from new max

⇒ if starting from min to find max
and we encounter a min → break & start again from new min

TC ⇒ $O(N^2)$ → How TC ⇒ $O(N)$ → first next class

min = 1 , max = 6.

ans = N
minL = -1
maxL = -1

eg =>

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 = 0
maxL = 1
l = 2
ans = 2

maxL = 1
minL = 5
l = 5
ans = 3

maxL = 3
minL = 5
l = 3
ans = 3.

minL = 5
maxL = 8
l = 4, ans = 4

minL = 12
maxL = 8.
l = 5, ans = 5

minL = 12
maxL = -1

→ Op = 2

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

pseudo

i) iterate → get min & max

```
if ( min == max )
    return 1
```

ii)

```
ans = N
minL = -1
maxL = -1
for( i = N-1 ; i >= 0 ; i-- ) {
    if( arr[i] == min) {
        minL = i
        l = (maxL - minL) + 1
```

→ abs.

if (minL != -1 & maxL != -1)

```
            ans = min(ans, l)
        }
        else if(arr[i] == maa) {
            maaI = i
            l = (maaI - minI) + 1
            ans = min(ans, l)
        }
    }
    print(ans)
```

if(minI != -1 & maaI != -1)

also.

```
maa = arr[0], c = 0.
for(i=0; i<N; i++) {
    if(arr[i] == maa)
        c++;
    else if(arr[i] < maa) {
        maa = arr[i]
        c = 1
    }
}
```