Given a string **s** of lowercase characters, return the **count of pairs (i,j)** such that **i < j** and **s[ i ] is 'a'** and **s[ j ] is 'g'**.

0 1 2 3 4 5

String s = 'abegag" Ans ⇒ 3.

String s = 'acgdgag' Ans ⇒ 4.

String s = 'bcaggaag' Ans → 5.

1) Brute force :-

0 1 2 3 4 5

String s = 'abegag"

n=6.

s = a a a a a

T.C → O(n²)

S.C → O(1)

```
int ans=0;
         i=5.
for (i=0; i<n; i++) {
    if (s[i]== 'a') {
        for (j=i+1; j<n; j++) {
            if ( s[j]=='g') {
                ans++;
            }
        }
    }
}
return result;
```

## Optimized Solution :-

String s = "$\overset{0}{a}\,\overset{1}{b}\,\overset{2}{e}\,\overset{3}{g}\,\overset{4}{a}\,\overset{5}{g}$"

### Observations :-

- For every **'g'**, we need to know the count of **'a'** on left side of **'g'**.
- We will store the count of **'a'** and whenever **'g'** is encountered, we will add the count of **'a'** to the result.

### Example :-

| | a | c | b | a | g | k | a | g | g |
|---|---|---|---|---|---|---|---|---|---|
| Count a=0 | 1 | 1 | 1 | 2 | 2 | 2 | 3 | 3 | 3 |
| Ans =0 | 0 | 0 | 0 | 0 | 2 | 2 | 2 | 5 | 8 |

```
count a = 0,   ans = 0;

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

return ans;
```

T.C → O(n)

S.C → O(1).

Carry forward :-

# ← Subarrays →

A subarray is a contiguous part of an array. It is formed by selecting a range of elements from the array. A subarray can have one or more elements and must be a contiguous part of the original array.

arr[] → 4, 1, 2, 3, -1, 6, 9, 8, 12

(2, 3, -1, 6) ✓

(9) ✓

4, 1, 2, 3, -1, 6, 9, 8, 12 ✓

(4, 12) ×

(1, 2, 6) ×

(3, 2, 1, 4) ×

## Quiz

**A[] = { 2, 4, 1, 6, -3, 7, 8, 4}**

Representation of a Subarray :-

→ Two ways

1) Start & end idx
2) Start Idx & len

0 1 2 3 4 5 6 7 8
arr[] → 4, 1, 2, 3, -1, 6, 9, 8, 12

sidx eidx

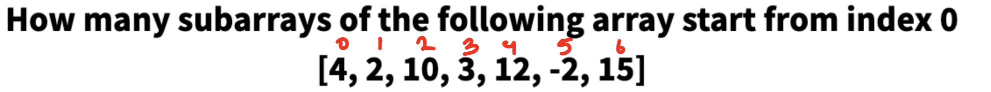
(2, 3, -1, 6) → 1st way (2 to 5)

→ 2nd way (2 & 4)

sidx len

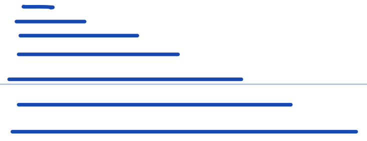
(9) → 1st way ( 6 to 6 )

→ 2nd way ( 6, 1 ).



How many subarrays of the following array start from index 0

0 1 2 3 4 5 6
[4, 2, 10, 3, 12, -2, 15]



|  | idx , | # subarrays |
|---|---|---|
| (4) | (0 – 0) |  |
| (4, 2) | (0 - 1) |  |
| (4, 2, 10) | (0 - 2) |  |
| (4, 2, 10, 3) | (0 - 3) |  |
| (4, 2, 10, 3, 12) | (0 – 4) |  |
| (4, 2, 10, 3, 12, -2) | (0 – 5) |  |
| (4, 2, 10, 3, 12, -2, 15) | (0 – 6) |  |

**How many subarrays of the following array start from index 1**

$[\overset{0}{4}, \overset{1}{2}, \overset{2}{10}, \overset{3}{3}, \overset{4}{12}, \overset{5}{-2}, \overset{6}{15}]$

Subarray
(s,e)

Ans => 6 .

(1,1) → 2

(1,2) → 2, 10

(1,3) → 2, 10, 3

(1,4) → 2, 10, 3, 12

(1,5) → 2, 10, 3, 12, -2

(1,6) → 2, 10, 3, 12, -2, 15

arr [n]

0 1 2 3 4 . . . . n-1

| Subarrays starting with 0 | starting with 1 | starting with 2 |
|---|---|---|
| (0-0) | (1-1) | (2-2) |
| (0-1) | (1-2) | (2-3) |
| (0-2) | (1-3) | (2-4) |
| ⋮ | ⋮ | ⋮ |
| (0-(n-1)) | (1-(n-1)) | (2-(n-1)) |
| n | n-1 | n-2 |

Subarrays starting with (n-1),

$((n-1) - (n-1))$

$1$

Total subarrays in an array

$$n + (n-1) + (n-2) + \dots\ 2 + 1 \Rightarrow \frac{n(n+1)}{2}$$

arr → (1, 2, 3) → $n=3$ $\quad \frac{3(3+1)}{2} \Rightarrow \underline{6}.$

Subarrays

(1)

(2)

(3)

(1,2)

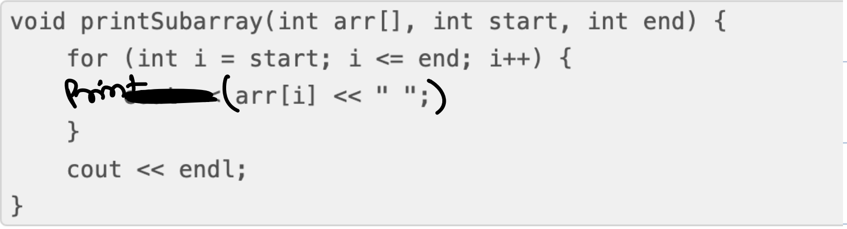
(2,3)

(1,2,3)

Given an array of integers and two indices, a start index and an end index, we need to print the subarrays of the array that starts from the start index and ends at the end index (both inclusive).

$$\begin{array}{ccccc} 0 & 1 & 2 & 3 & 4 \\ (10, & 20, & 30, & 40, & 50) \end{array}$$

s = 2, e = 4 . → (30, 40, 50) .



```
void printSubarray(int arr[], int start, int end) {
    for (int i = start; i <= end; i++) {
    Print[redacted](arr[i] << " ";)
    }
    cout << endl;
}
```

T.C → O(n)

S.C → O(1).

Ques Print all Subarrays of a given array.

$$arr = [\overset{0}{10}, \overset{1}{20}, \overset{2}{30}]$$

$$\frac{n(n+1)}{2} \Rightarrow \frac{3(3+1)}{2} \Rightarrow 6 .$$

(0-0) → 10

(0-1) → 10, 20

(0-2) → 10, 20, 30

(1-1) → 20

(1-2) → 20, 30

(2-2) = 30

arr = [$\overset{0}{10}$, $\overset{1}{20}$, $\overset{2}{30}$]

```
for( s=0; s<n; s++) {
    for(e=s; e<n; e++) {
            print (s + " " + e);
    }
}
```

| s | e |
|---|---|
| 0 | 0 |
| 0 | 1 |
| 0 | 2 |
| 1 | 1 |
| 1 | 2 |
| 2 | 2 |

(0 0) → 10

(0 1) → 10, 20

(0 2) → 10,20,30

(1, 1) → 20

(1, 2) → 20,30

(2, 2) → 30

```
for( s=0; s<n; s++) {
    for(e=s; e<n; e++) {
        for(i=s; i<=e; i++) {
                print(arr[i]);
        }
    }
}
```

T.C → $O(n^3)$

S.C → $O(1)$.

## Ques

**Given an array of N integers, return the length of smallest subarray which contains both maximum and minimum element of the array.**

0 1 2 3 4 5 6 7 8 9 10

**2 2 6 4 5 1 5 2 $\underline{6\ 4\ 1}$**

Maxm = 6

min = 1

Ans → $\underline{3}$.

## Brute force :-

Check all subarrays & find array.

T.C → $O(n^3)$

S.C → $\underline{O(1)}$.

0 1 2 3 4 5 6 7 8 9 10

**2 2 6 4 5 1 5 2 6 4 1**

Maxm = 6

min = 1

Ans → 3.

Observations :-

1) we need only 1 max & 1 min in that.

2) subarray ( no max & min will be there in b/w)

6
or
1 1 6  max ~~max~~ min

min max

1 6 5 ← min & max will be on the corners.

min max

max min

## Dry Run :-

last_min_idx = ~~-1~~ ~~8~~ 10
last_max_idx = ~~-1~~ ~~7~~ 8
minValue = 1
MaxValue = 6
ans = ~~+∞~~ ~~4~~ 3

| idx → | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | 2 | 2 | 6 | 4 | 5 | 1 | 5 | 2 | 6 | 4 | 1 |

if(Max==Min) → { return 1 }

```
int min = Todo
int max = Todo
int last_min_idx = -1, last_max_idx = -1;
ans = Integer.Max_Value;
for (i=0; i<n; i++) {
    if (arr[i]==min) {
        if (last_Max_Idx != -1) {
            ans = Min(ans, i-last_Max_Idx+1);
        }
        last_min_idx = i
    }
    else if (arr[i]==max) {
        if (last_min_Idx != -1) {
            ans = min (ans, i-last_min_Idx+1);
        }
        last_Max_idx = i ;
    }
```

T.C → O(N)
S.C → O(1)

| 3

return ans;

Edge Case :-

arr -> 8, 8, 8    len=1

max = 8
min = 8

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

minIdx maxIdx

||
2

| 10 | 20 | 30 | 40 | . . . |
|---|---|---|---|---|