Q → Given a character array / string, count the pairs (i, j) s.t. $i < j$, $A[i] = 'a'$ & $A[j] = 'g'$.

0	1	2	3	4	5
a	b	e	g	a	g

Pairs → (0,3) (0,5) (4,5) Ans = 3

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

Pairs → (0,2) (0,6) (0,4) (5,6) Ans = 4

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

Pairs → (2,3) (5,7) (2,4) (6,7) (2,7) Ans = 5

Bruteforce → $i < j$, $A[i] = 'a'$ & $A[j] = 'g'$.

```
cnt = 0
for i → 0 to (N-1) {
    for j → 0 to (N-1) {
        if ( i<j && A[i] == 'a' && A[j] == 'g')
            cnt++
    }
} return cnt
```

TC = $O(N^2)$ SC = $O(1)$

```
cnt = 0
for i → 0 to (N-1) {
    if (A[i] == 'a') {
        for j → (i+1) to (N-1) {
            if (A[j] == 'g')
                cnt++
        }
    }
}
return cnt
```

→ count #'g' from (i+1) to (N-1)

A → [a a a . . .]

TC = $O(N^2)$ SC = $O(1)$

	0	1	2	3	4	5	6	7	8	9	10
A = [	b	c	a	g	c	g	a	x	a	g	t]
#'g' →	3	3	3	3	2	2	1	1	1	1	0

cnt_g

```
if( A[i] != 'g' )   cnt_g[i] = cnt_g[i+1]
else                cnt_g[i] = cnt_g[i+1] + 1
```

```
if (A[N-1] == 'g')   cnt_g[N-1] = 1
else cnt_g[N-1] = 0
for i → (N-2) to 0 {
    if( A[i] != 'g' )   cnt_g[i] = cnt_g[i+1]
    else                cnt_g[i] = cnt_g[i+1] + 1
}
```

→ TC = O(N)

→ calculating & storing

```
ans = 0
for i → 0 to (N-1) {
    if (A[i] == 'a') {
        ans += cnt_g[i+1]
    }
```

→ using

→ TC = O(N)

~~cnt_g[i+1]~~ → cnt_g[i] ✓

```
} return ans
```

Overall TC = O(N+N) = O(N)

SC = O(N) → crt_g[]

```
ans = 0
if (A[N-1] == 'g')  crt_g = 1
else crt_g = 0
for i → (N-2) to 0 {
    if (A[i] == 'g')  crt_g = crt_g + 1   // calculating
    if (A[i] == 'a')  ans += crt_g        // using
}
return ans
```

carry forward

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

```
i
0  1  2  3  4  5  6  7  8  9  10
A = [ b  c  a  g  c  g  a  x  a  g  t ]
```

ans = 0 + 2 + 3 = 5 ✓

crt_g = 0 + 2 + 1

Sol 2 → Travel L → R & count 'a'

Subarrays → continuous part of the array.

```
0  1  2  3  4  5  6  7  8  9  10
A = [ b  c  a  g  c  g  a  x  a  g  t ]
      subarray
```

single element ✓

full array ✓

```
0  1  2  3  4  5  6  7
A = [ 2  4  1  6  -3  7  8  4 ]
                      L     R
```

subarray (5, 7)

0	1	2	3	4	5	6
A = [4	2	10	3	12	-2	5]

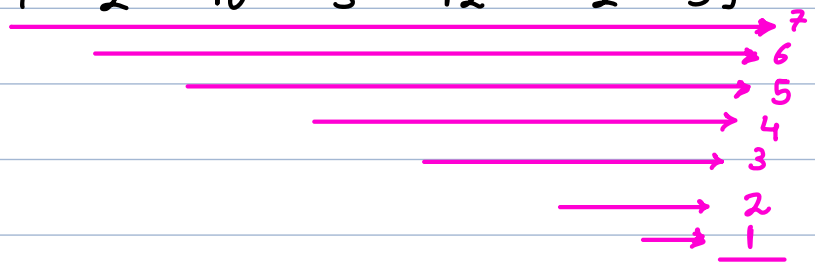
subarrays starting from index 0 → L = 0 Ans = 7

R = [0 6] → 7

subarray starting from index 1 → L = 1 Ans = 6

R = [1 6] → 6

0	1	2	3	4	5	6
A = [4	2	10	3	12	-2	5]



Total # subarrays = 1+2+3+...+N 1+2+...+7 = 28

$= \frac{N * (N+1)}{2}$

Q → Given an integer array & print subarray from index L to R.

			L		R	
0	1	2	3	4	5	6
A - [4	2	10	3	12	-2	5]

```
for i → L to R {
    print (A[i])
}
```

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

Q → Print all possible subarrays.

A = [1 2 3]

1

1 2

1 2 3

2

2 3

3

```
for L → 0 to (N-1) {
    for R → L to (N-1) {        R >= L        L —— R
        for i → L to R {
            print (A[i])
        }
    }
}
```

TC = $O(N^3)$ SC = $O(1)$

Q → Given an integer array, find the length of smallest subarray which contains both min & max element of the array.

	0	1	2	3	4	5	6	7	8
A = [	5	8	1	3	8	10	6	1	2]

min = 1

max = 10 Ans = 3

	0	1	2	3	4	5	6	7	8	9	10
A = [	2	2	6	4	5	1	5	2	6	4	1]

min = 1

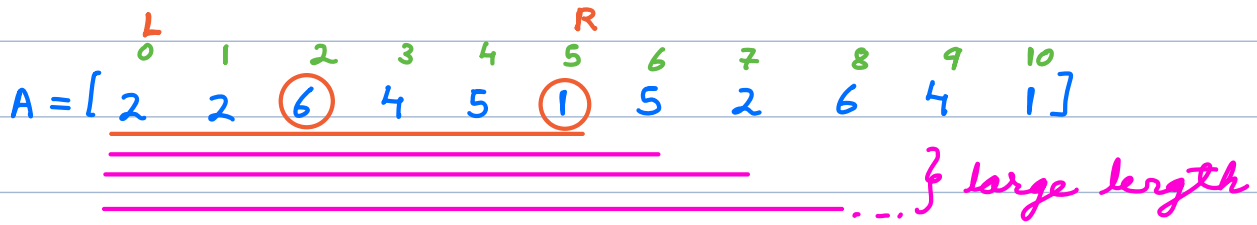
max = 6 Ans = 3

```
minA = A[0]        maxA = A[0]
for i → 1 to (N-1) {
    minA = min (minA, A[i])
    maxA = max (maxA, A[i])
```

```
}                                                    ✓

ans = N    // complete array
for L → 0 to (N-1) {
    for R → L to (N-1) {        L —— R
        vstMin = false   vstMax = false
        for i → L to R {
            if (A[i] == minA)   vstMin = true
            if (A[i] == maxA)   vstMax = true
        }
        if (vstMin && vstMax)
            ans = min(ans, R-L+1)
    }
}
return ans
```

TC = $O(N^3)$ SC = $O(1)$

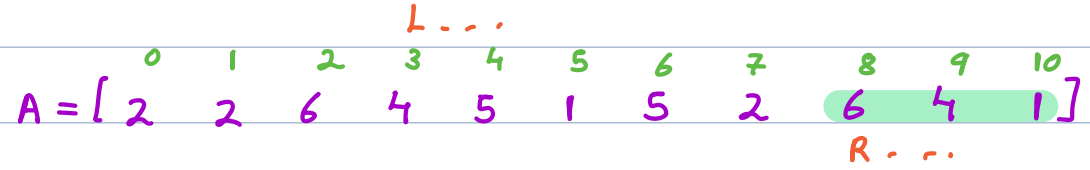



```
ans = N
for L → 0 to (N-1) {
    vstMin = false    vstMax = false
    for R → L to (N-1) {
        if (A[R] == minA)   vstMin = true
        if (A[R] == maxA)   vstMax = true
        if (vstMin && vstMax) {
            ans = min(ans, R-L+1)
            break
        }
    }
}
```

TC = $O(N^2)$

SC = $O(1)$

} return ans



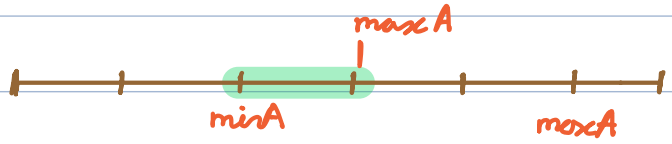
minA = 1 ~~false~~ true

maxA = 6 ~~false~~ true

ans = ~~4 6 5 4~~ 3

Observations → min length

minA & maxA



ans subarray → start at min & end at max OR

start at max & end at min

(A[i] == minA) → find next closest maxA index on right side ✓

(A[i] == maxA) → find next closest minA index

// minA maxA ✓

minInd = -1 maxInd = -1 ans = N

for i → (N-1) to 0 {

if (A[i] == minA) {

minInd = i // calculate

if (maxInd != -1) // minInd — maxInd

ans = min (ans, maxInd - minInd + 1) // use

```
    } if (A[i] == maxA) {
        maxInd = i          // calculate
        if (minInd != -1)   // maxInd — minInd
            ans = min(ans, minInd - maxInd + 1) // use
    }
} return ans
```

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

	0	1	2	3	4	5	6	7	8
A = [	5	8	1	3	8	10	6	1	2]

(... i at index 5; indices 5–7 highlighted)

minA = 1   minInd = ~~-1~~ 7   ans = ~~N~~ 3

maxA = 10   maxInd = ~~-1~~ 5