Welcome स्वागतम्

- PROBLEM SOLVING TECHNIQUE – CARRY FORWARD on ARRAY (Data Structure)
  - Problems → optimising

count

Q. Given a character s[N] array, we need to calculate no of pairs (i,j) such that $i < j$ and $\underbrace{s[i] == \text{'a'}}$ && $\underbrace{s[j] == \text{'g'}}$ ← condition

All characters are in lowercase.

Ex

	0	1	2	3	4	5	6	7
S[N] =	b	a	a	g	d	c	a	g

Pairs of Indices

i<j	a(i)=='a', a(j)=='g'	
(1,3)	a g	
(1,7)	a g	
(2,3)	a g ✓	2<3
(2,7)	a g ✓	2<7
(6,7)	a g	

5 Pairs

g a
~~(3,6)~~ i<j

Ex

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

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

4 Pairs

Algo-1

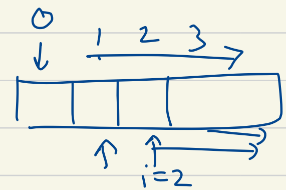
i<j

```
count = 0;
for ( i=0 ; i<= N-2 ; i++ ){

    for (j=i+1 ; j<= N-1 ; j++ ) {
        if( arr[i] == 'a' && arr[j] == 'g' ){
            count ++;
        }
    }
}

print(count);
```

$O(N^2)$



(0,1)	(1,2)	(2,3)
(0,2)	(1,3)	
(0,3)		
~~(0,4)~~		

$N = \boxed{4}$

$1+2+3+ \cdots (N-1)$

$= \propto N^2 = O(N^2)$

$3+2+1$

Generate all Pairs & check → $O(N^2)$ Time
→ $O(1)$ Space.

Algo-2

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

n-2

Search for g only if

arr(i) == 'a'

0 → (1, 2, 3, 4, 5, 6)
1 → x
2 → x
3 → x
4 → x
5 → (6)

```
count = 0
for (i=0 ; i<= n-2; i++) {

    if (arr[i] == 'a') {
        // Search for g    i+1 ---- n-1
        for (j = i+1 ; j<= n-1 ; j++) {
            if (a[j] == 'g') {
                count++;
            }
        }
    }
}
```

True for every idx then

Worst Case still remains same

a, a, a, a, a, a, a, g

$O(N^2)$ time

$O(1)$ space

Algorithm-3

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

r = 0

Step-1

=> Counting the number of 'g' to right of every 'a'

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

??

→ Prefix

suffix ←

g Count → count of 'g' at 'i' index

```
gcount[N-1] = arr[N-1]=='g' ? 1 : 0;
for (i=N-2; i>=0; i--){
    if (arr[i] != g)
        gcount[i] = gcount[i+1];      // prev val
    else
        gcount[i] = gcount[i+1]+1;    // prev val + 1
}
```

arr(i) = arr(i+1)

N-1   N

out of bounds exception

i=2 i=3

g	d
3	2

gCount[i] = g Count[i+1] +1;

g[2] = g[3] +1

Step-2

Prefix Sum

Store count of g

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

0, 2
0, 4 } 3
0, 6
+
5, 6 } 1

4

```
count = 0
for(i=0;  i<= n-1;  i++){
    if( arr(i) == 'a' ){
        Count = count + g count[i],
```

3

3

Time O(N+N)
= O(N)

Space = O(N)

0 + 3 + 1
= 4

Time & Space

0^{th} idx

a,	c,	g,	d,	g,	a,	g
3	3	3	2	2	1	1

Step-2

```
count = 0
for(i=0 , i<= n-1; i++ ){
    if (arr(i) == 'a') {
        count = count + gCount[i];
```

Array we built in step-1

3

$\boxed{0+3}$

$\boxed{3+1} = 4$

$O(N^2)$ Time → $O(N)$, Time → $O(N)$

$O(1)$ Space $O(N)$ space   ?

$\boxed{gCount = 0, \quad count = 0}$

Example —

a	'd'	g	a	g	'a'	g	'f'	g
$gc=4$	$gc=4$	$gc=4$	$gc=3$	$gc=gc+1$	$gc=2$	$gc=2$	$gc=1$	$gc=gc+1$
$c=5+4$	$c=5$	$c=5$	$c=5$	$gc=3$	$c=2$	$c=0$	$c=0$	$=1$
$=9$				$c=2$	$c=c+gc$			$count=0$

if you see a → $c = c + gc$

if you see g → $gc = gc + 1$

Pairs → Count

Carry forward

```
gc = 0
 c = 0

for ( i=N-1; i>=0 , i--) {

    if (arr(i) == 'g') {
            gc = gc+1;
    }

    else if (arr(i) == 'a') {
            c = c + gc;
    }

}

print (count)
```

Time → O(N)
Space → O(1)

a g g g

No of g to right of A.

1	2	3	4

Prefix Sum

1	3	6	10

Suffix sum

10	9	7	4

Idea

L → R

'a' 'a' 'a' g

Count of 'a' to the Left of g

```
for (i=0; i<=N-1, i++) {
    if ( __ a) { __ };
    if ( __ g) { __ };
}
```

Q. Leaders in a Array

Given an array[N] you have to find all leaders in the array, an element is leader if it strictly greater than all elements in it's right side or strictly greater than max element in the right. equivalent

arr[N-1] is always considered as a leader.

Ex-1

Indices	0	1	2	3	4	5	6	7
Array	15	-1	7	2	5	4	2	3

5 Leaders

Ex-2

0	1	2
10	8	8

2 leaders

Algo-1

0	1	2	3	4	5	6	7
15	-1	7	2	5	4	2	3

a(i)

	j		
i=0	(1—7)	7<15	15
i=1	(2—7)	7>-1	×
i=2	(3—7)	5<7	7
i=3	(4,7)		
i=4	(5,7)		
i=5	(6,7)		
i=6	(7,7)		
i=7		3 > -∞	3

Check if every is a leader

```
for(i=0 ; i<= n-1, i++) {
    max = -∞
    for(j=i+1, j<= n-1; j++){
        if ( a(j) > max) {
            max = a(j),
        }
    }
    if (a(i) > max){
        => a(i) is a leader , cnt ++
    }
}
```

→ finding out

i+1 —— n-1

$O(N^2)$ Time

$O(1)$ Space

Algo-2 → Carry forward the max from Right

i

0	1	2	3	4	5	6	7
15	-1	7	2	5	4	2	3

↑ Leader

max = $-\infty$

3

15 > 7   -1 > 7   7 > 5   2 > 5   5 > 4   4 > 3   2 > 3   3 > max

Leaders = 5   Leaders = 4   Leader = 3   Leader = 2   Leader = 1

max 15   Max = 7   Max = 5   Max = 4   Max = 3

```
Max = -∞ , L = 0
for ( i = N-1 ; i >= 0 ; i-- ) {
        if ( a(i) > Max ) {
            ⇒ Max = a(i);
            ⇒ Leaders = Leader + 1;
        }
}
```

TC O(N)

Space O(1)

1	5	3	2	6

Subarrays

3 2 ✓

5 ✓

5 3 2 ✓

5 3 2 6 ✓

~~5 6~~ Not continuous

10.35

Closest Min Max

⇒ Given an array, find the length of the smallest **subarray** which contains both Min & Max of the array!

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

Duplicates

Min - 1

Max - 6

Length - 4

Ex-2 → 4 $\boxed{10 \quad 8 \quad 3 \quad 2}$ 7 6

No Duplicates

max - 10

Min - 2

Length - 4

Potential Approaches

Sure Thing

"Min" & "Max" would be present at Corners

Length of subarray

3, 4, 5, 6

6

MinIdx MaxIdx

maxIdx − MinIdx + 1

6 − 3 + 1 = 4 element

3, 4, 5, 6

Max Idx Min Indx

MinIdx − Max Idx + 1

6 − 3 + 1 = 4 elemen

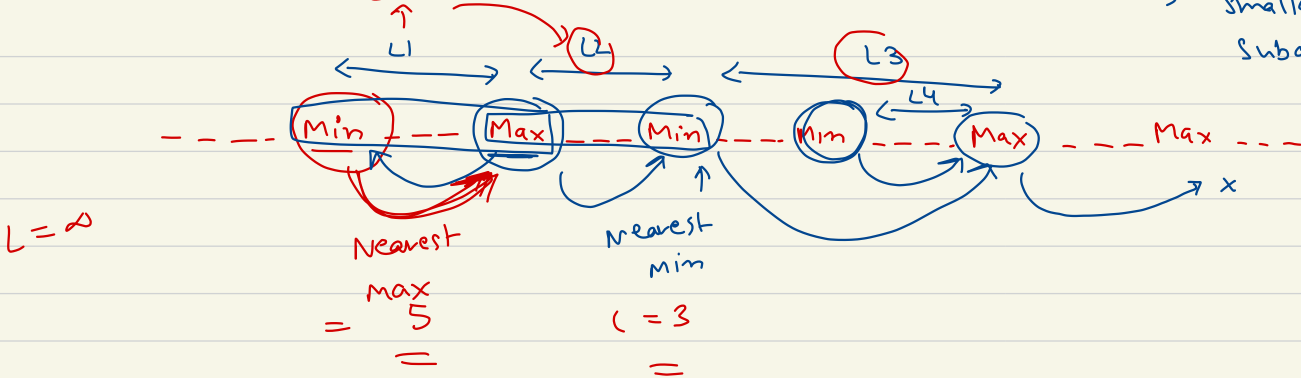
0	1	2	3	4	5
3	4	6	1	2	5

Corners of subarray

3 − 2 + 1

= $\boxed{2}$ ← Length

min → min(L1, L2, L3, L4) → Smallest Subarray.



Step-1 find out the min / max element min → 1 , max → 6

Step-2 Min ↷ Nearest max on Right

Max ↷ Nearest Min on Right

inner loop

ind	0	1	2	3	4	5	6	7	8	9	10	11	12
arr	2	2	6	4	5	1	5	6	2	4	3	4	1
	x	x	✓	x	X	min	X	max	X	X	x	x	x
			Max			Min		Max					

$L = 5-2+1 = \boxed{4}$

Smallest = 4

$L = 7-5+1 = 3$

Smallest = 3

$L = 12 - 7 + 1 = 6$

= 3

Pseudo code

O(N) ←

MinE = ✓

MaxE = ✓

WOP → TODO

Step-1

Smallest = ~~∞~~ N

Step-2 → for (i=0 , i< N , i++) {

$O(N^2+N)$

$= O(N^2)$ time

Space

$= O(1)$

if (a(i) is MinE) {

finding nearest max for given Min →

for (j = i+1 ; j< N ; j++) (

if (a(j) is MaxE) { Math.

smallest = min(smallest , j-i+1);

break;

}

}

}

starting point of subarray

else if (a(i) is MaxE) {

find nearest min for given max →

for (j = i+1; j< N; j++) (

if (a(j) is MinE) {

Smallest = min(smallest, j-i+1);

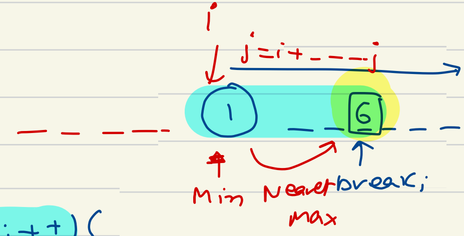
break;

) }

}

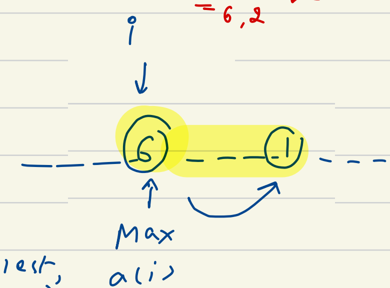
}

smallest = ~~∞~~ , 3



6 , 5-4+1

= 6 , 2



→ 1

MinF = ✓ 1

maxE = ✓ 6

N−1

1 -- 6 -- 6 -- 6 - Min

Min Max Max Max

R to L

Carry Forward Idea

maxIdx = −1; smallest = N, minIdx = −1

O(N) time
O(1) space

Min

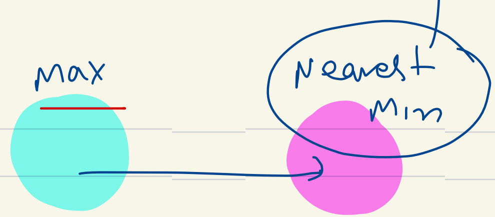
Nearest Max

```
for(i = N-1; i >= 0; i--) {
    if ( arr[i] == MinE) {
        minIdx = i;
    if(maxIdx != -1) {
        len = maxIdx - i + 1
        Smallest = min(Smallest, len);
    }
}
```

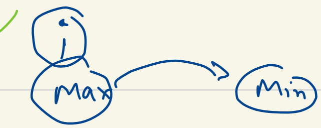
Carry Forward for nearest max

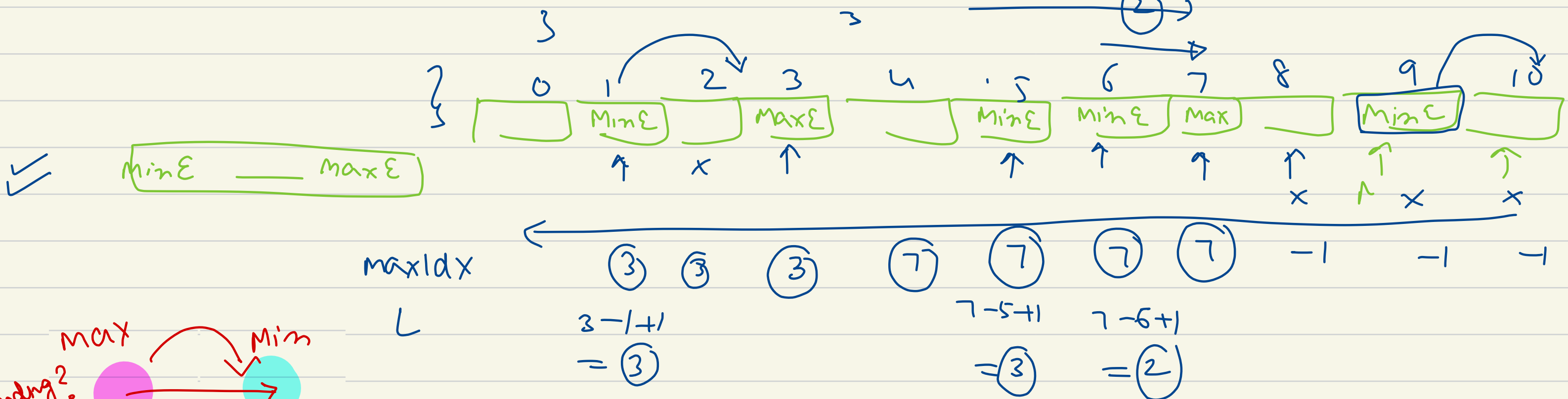
Helps me here

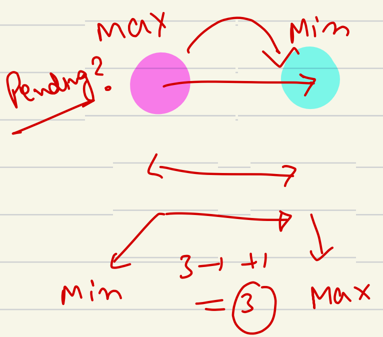
Carry Forward

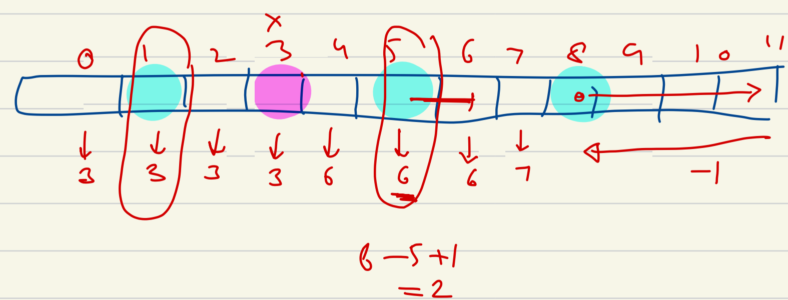


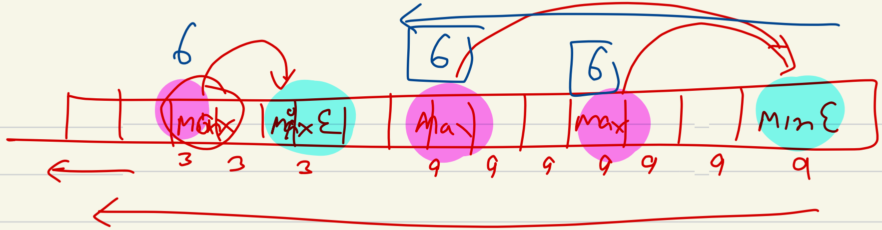

```
else if (arr(i) == maxE) {
    maxIdx = i;
    if(minIdx != -1) {
        len = minIdx - i + 1;
        smallest = min(smallest, len);
    }
}
```





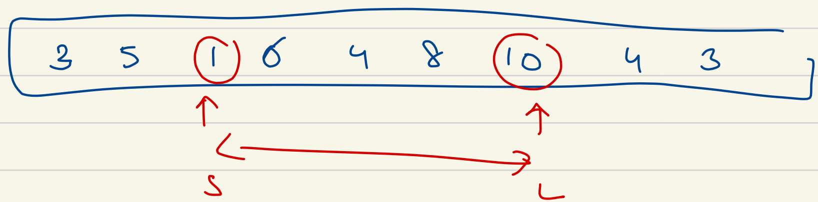






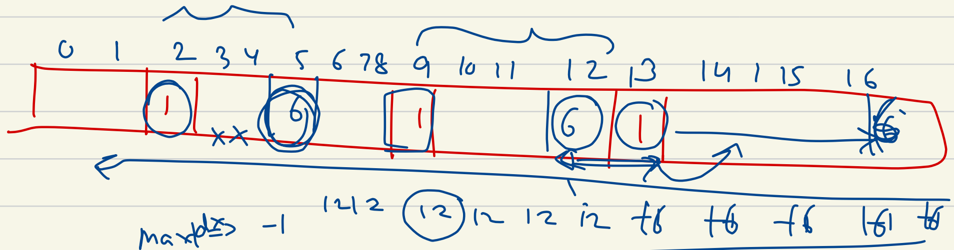
→ Maintain the min Idx

→ Update the length when max E is encounte..



Easy

No Duplicates. (L − S + 1)



if (———)

{

Carry Forward

1 ——— 6

12 − 9 + 1 = 4

5 − 1 + 1 = 4

3

6 ——— 1

max idx = ~~12~~ 5