Todays Content:

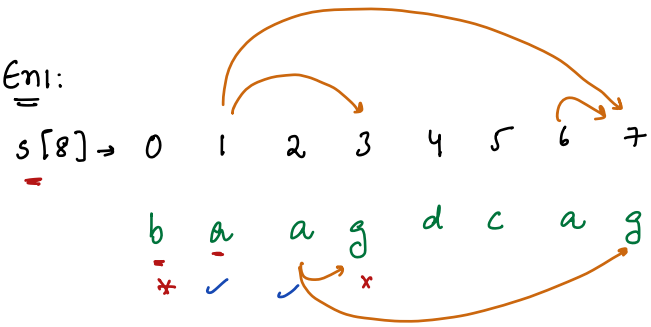
- Count pairs
- leaders in array
- Subarray basics → {concept}
- Smallest Subarray with Both min & max : {Asked in Interviews}

18) Count Pairs 'ag'

Given a char[], calculate no of pairs i, j such that

i < j && s[i] = 'a' && s[j] = 'g' All characters are lowercase 'a-z'

Eg1:

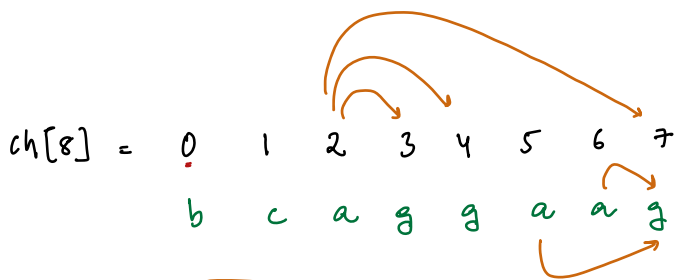


pairs: → 5 pairs

<1,3> <1,7>

<2,3> <2,7> <6,7>

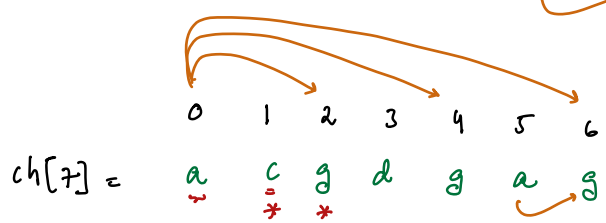
j i
<3 6 7> : i > j → Not valid
g a



Pairs:

<2,3> <2,4> <2,7>

<5,7> <6,7>



Pairs: <0,2> <0,4> <0,6>

<5,6>

Ideal: Check all pairs → optimization

```
int pairs(char s[]) {
    int n = s.length, c = 0
    i=0; i<n; i=i+1) {
        j = i+1; j<n; j=j+1) {
            if(s[i]=='a'
                && s[j]=='g') {
                c=c+1
            }
        }
    }
    return c;
}
```

i = N-1

j = N; j<N won't enter 2nd loop

TC → O(N²) SC → O(1)

```
int pairs(char s[]) {
    int n = s.length, c = 0
    i=0; i<n; i=i+1) {
        if(s[i]=='a') {
            j = i+1; j<n; j=j+1) {
                if(s[j]=='g') {
                    c=c+1
                }
            }
        }
    }
    return c;
}
```

TC: O(N²) SC → O(1)

Tracing — if s[i] == 'a', { iterate on right side to get no: of g's }

Ex2:

	0	1	2	3	4	5	6	7	8
ch[] =	a	d	g	a	g	a	g	f	g
	✓								

c = 0

+1 +1 +1 +1 ⇒ c = c+4.

* * ✓ +1 +1 +1 ⇒ c = c+3.

* ✓ +1 +1 ⇒ c = c+2.

c = 9

* * *

← Iterate once from Right - left & get no: of g's for every a

Tracing

c = 0, ans = 0

Ex3:

	0	1	2	3 ✓	4 ✓	5 ✓	6 ✓	7 ✓	8 ✓
ch[]:	a	d	g	a	g	a	g	f	g
	ans = ans+c; ans = 9 ↳ final ans		c=c+1; c=4	ans=ans+c; ans=5	c=c+1; c=3	ans=ans+c; ans=2	c=c+1, c=2		c=c+1, c=1

Pseudocode: → TC: O(N) SC: O(1)

```
int Pairs(char s[]){
    int n = s.length;
    int c=0, ans=0
    for(i=n-1; i>=0; i--){
        if(s[i]=='g'){c=c+1;}
        else if(s[i]=='a'){
            ans = ans+c
        }
    }
    return ans;
}
```

For every 'a' calculate no: of 'g' on right side

for every 'g' calculate no: of 'a' on leftside

Iterate from left → right → TODO

	0 ✓	1 *	2 ✓	3 ✓	4 ✓	5 ✓	6 ✓	7 x	8
Eg3:									
ch[i]:	a	d	g	a	g	a	g	f	g
c = 0	c=c+1		ans = ans+c	c=c+1	ans = ans+c	c=c+1	ans = ans+c		ans = ans+c
ans = 0	c = 1		ans = 1	c = 2	ans = 3	c = 3	ans = 6		ans = 9

28) leaders in a Array

Given an ar[N], you have to find leaders in ar[]

An ele is leader:

- if it is strictly greater than max of all elements in leftside
- Note: ar[0] is already considered as leader

Ex1: ar[8]:

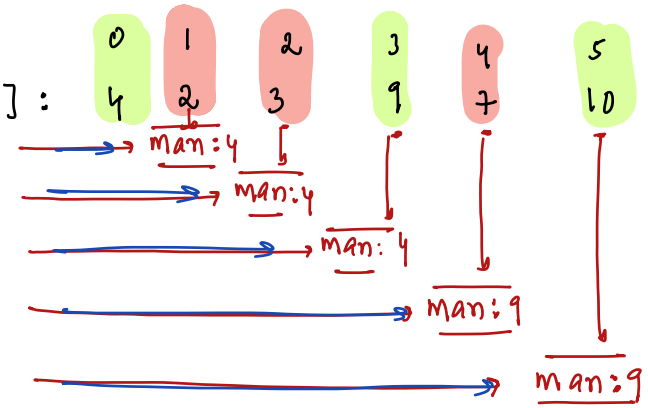
0	1	2	3	4	5	6	7
3	2	4	5	2	7	-1	15

c=5, leaders in array

Ex2: ar[6]:

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

c=3 leaders in array



Idea:

for every element ar[i],
get max from of array
[0, i-1] & compare with < ar[i]

```
int leaders (int ar[]) {
    int n = ar.length;
    int c = 1;
    for(i = 1; i < N; i++) {
        // check ar[i] leader
        // iterate from [0 i-1]
           & get max
        if(max < ar[i]) {
            c = c+1
        }
    }
    return c;
}
```

TC: O(N²) SC: O(1)

Eg1: ar[8]:

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

c = 1
max = 3

max < ar[2] → c = c+1 : 2, max = 4

max < ar[3] → c = c+1 : 3, max = 5

max < ar[5] → c = c+1 : 4, max = 7

max < ar[7] → c = c+1 : 5, max = 15

TODO → TC: O(N) SC: O(1)

8:33 pm

```
Int leaders (Int ar[]) {
    int n = ar.length;
    int c = 1;
    int m = ar[0]
    i=1; i<N; i++) {
        //ar[i] is leader
        if( ar[i] > m) {
            c = c+1
            m = ar[i]
        }
    }
    return c;
```

TC: O(N) SC: O(1)

```
}
```

Subarray: Continous part of an array is called Subarray

a) Complete array is also Subarray

b) Single element is also Subarray

c) [ ] is not considered as Subarray

	0	1	2	3	4	5	6	7	8	9
Eg: ar[10] :	2	4	3	7	9	6	8	7	9	2

[2-4] + [6-7] + [9-9]

Is Subarray?

[2 5] : Yes

[1-4] + [6-8] : No

[7-9] : Yes

[1-1] : Yes

[0-9] : Yes

[2-4] + [6-7] + [9-9] : No

length of Subarray: How many elements in Subarray

[3 7] → 3 4 5 6 7 → 5 → 7-3+1 → 5

[2 4] → 2 3 4 → 3 → 4-2+1 → 3

[5 8] → 5 6 7 8 → 4 → 8-5+1 → 4

[i j] → [j-i+1]

i: start index of subarray

j: end index of subarray

38] Closest Min Max

Given N array elements, find length of Smallest Subarray which contains Both Min & Max of array

Ex1: ar[10] :

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

min = 1, max = 6

Subarray : len

[s e] : e-s+1

[3 6] : 6-3+1 = 4

ans = 4

Ex2: ar[10] :

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

subarray : len

[s e] : e-s+1

[5 8] : 8-5+1 = 4

[8 10] : 10-8+1 = 3

ans = 3

Ex3: ar[5] :

0	1	2	3	4
8	8	8	8	8

min = 8, max = 8

[s e] : e-s+1

0-0 : 0-0+1 = 1

2-2 : 2-2+1 = 1

ans = 1

observations:

→ Min & Max should be present in corners in final ans subarray

Min . . . Max . . . . . .

. . . Max . . . Min . . . .

↳ Case-I: Min . . . . . Max : if ar[i] == Min :

get nearest max find in right side

Case-II: Max . . . . . Min : if ar[i] == Max:

get nearest min find in right side

Ex:

ar[13] ans = N = 13 min = 1 max = 6

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

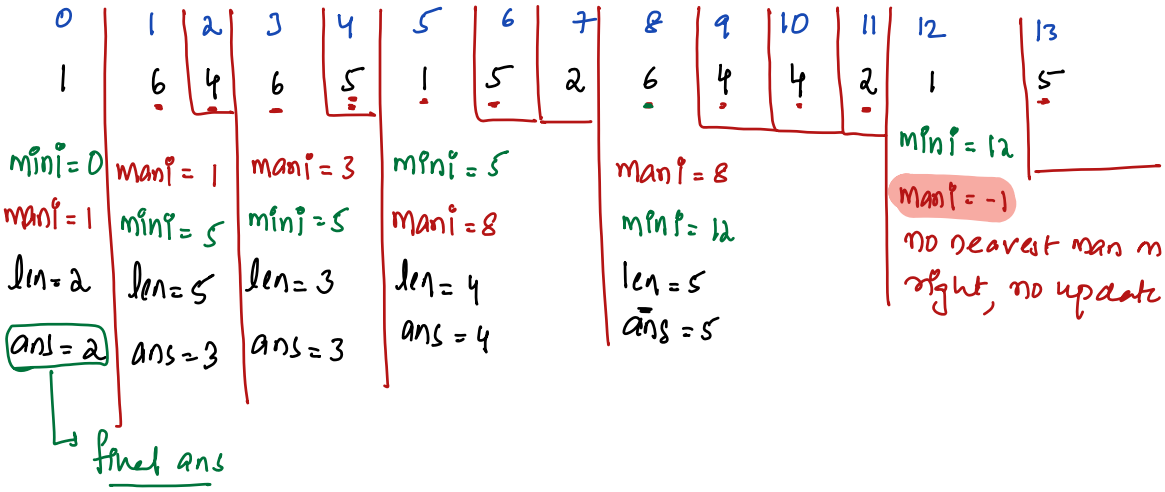
len = [2 5] = 4 len = [5 8] = 4 len = [8 10] = 3 → ans = 3

```
int smallength(int ar[]){ → TC: O(N + N^2)  SC: O(1)
    int max = (max of ar[])  } → O(N)   TC: O(N^2)   O(1)
    int min = (min of ar[])  }
    if(max == min) {return 1}
    int n = ar.length;
    int ans = n
    for(int i = 0; i < N; i++) { → O(N)
        if (ar[i] == min){
            // search for nearest max and n right
            for(j = i+1; j < N; j++) { → O(N)
                if (ar[j] == max) {
                    // [i j] len = j - i + 1
                    if( len < ans ) { ans = len }
                    break;
                }
            }
        }
        else if( ar[i] == max) {
            // search for nearest min and n right
            for(j = i+1; j < N; j++) { → O(N)
                if (ar[j] == min ) {
                    // [i j] len = j - i + 1
                    if( len < ans ) { ans = len }
                    break
                }
            }
        }
    }
    return ans;
}
```

idea2: Iterate from Right to left & get nearest min_ind & nearest max_index

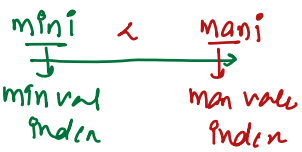
// To indicate, yet to found

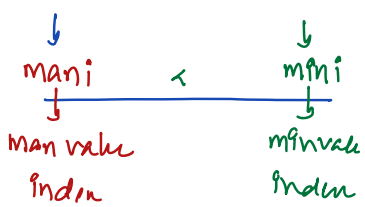
ar[14]: min = 1, max = 6, ans = 14, mini = -1, maxi = -1;



```
int minLength(int ar[]){   TC: O(N+N+N) ≈ O(N)
    int n = ar.length
    int ans = n
    int man = {man of arr}   int min = {min of arr}
    if(min==man){return 1}
    int mini = -1;   int mani = -1
    for(i = N-1; i>=0; i--){
        if(ar[i]==min){ //nearest man index on right
            mini = i
            if(mani != -1){
                len = mani - mini + 1
                if(len < ans){ans = len}
            }
        }
        else if(ar[i]==man){ //nearest min index on right
            mani = i
            if(mini != -1){
                len = mini - mani + 1
                if(len < ans){ans = len}
            }
        }
    }
    return ans;
}
```

O(N) – O(N) [we can calculate min & max in same loop]





TC: O(N) SC: O(1)

version II = TODO → left → right TC: O(N) SC: O(1)

Case-I: Min ..... Max : if ar[i] == Max:
get nearest min and in leftside

Case-II: Max ..... Min : if ar[i] == Min
get nearest max and in left side

Doubts:

	0	1	2	3	4	5	6	7	8	9	10	11
ar[ ] :	2	3	7	6	2	-1	2	3	4	1	7	-1
min =	2	2	2	2		-1					-1	
min i =	0	0	0	0		5					5	
max =	2	3	7	7		7					7	
max i =	0	1	2	2		2					2	

* If we get new max/min we update min & max & their index

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

len = [2 5] = 4 len = [5 8] = 4 len = [8 10] = 3 → ans = 3