SCALER

Array - Carry forward & Subarrays

TABLE OF CONTENTS

Hello Everyone

Notes

Contest Information:

Date : 20th June (Tentative)
9.00 PM IST (1.5 hrs)

Discussion — 10.30 PM IST
Passing marks - 60%
Total 3 Questions

* Absolutely necessary to appear for the contest.

Q. What if I fail?

Appear for the reattempts.

Total 3 reattempts.

Our goal should be to clear the live contest itself.

Reattempts is only for exceptional cases.

Q. How to prepare?

Solve assignment questions regularly and do quick revision before contest.

Keep PSP above 90%.

Count 'a-g' pairs

< *Question* > : 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'.

String s = "a b e g a g"
0 1 2 3 4 5

0,3
0,5
4,5

ans = 3

Quiz 1: String s = "a c g d g a g"
0 1 2 3 4 5 6

a	0	0	0	5
g	2	4	6	6

ans = 4

ans = ~~0 1 2 3 4~~

Quiz 2:

String s = "b c a g g a a g"

indices: 0 1 2 3 4 5 6 7

i

a	2	2	2	5	6
g	3	4	7	7	7

ans = 5

```
int countOfAb(String s)
{
    int ans = 0;
    for(i=0; i<s.length(); i++)        // based on len.
    {
        if (s[i] == 'a')
        {
            for (j=i+1; j<s.length(); j++)
            {
                if (s[j] == 'g')
                {
                    ans++;
                }
            }
        }
    }
    return ans;
}
```

TC: $O(N^2)$

SC: $O(1)$

		0	1	2	3	4	5	6	7	8
S =	"	a	c	b	a	g	k	a	g	g "

index	0	1	2	3	4	5	6	7	8
Count-a	~~0~~ 1	1	1	2	2	2	3	3	3
ans =	0	0	0	0	2	2	2	5	8

< / > Code

```
int countAGPair(String s)
{
    int ans = 0;
    int countA = 0;

    for(i=0; i< s.length(); i++)
    {
        if(s[i] == 'G')
        {
            countA++;
        }
        else if( s[i] == 'g')
        {
            ans = ans + countA;
        }
    }
    return ans;
}
```

TC: O(N)

SC: O(1)

Subarrays: A subarray is a continous part of an array. It is form by selecting one or more elements from the array.

0	1	2	3	4	5	6	7	8
4	1	2	3	-1	6	9	8	12

Array → {4, 1, 2, 3, -1, 6, 9, 8, 12}

1. 2, 3, -1, 6 ✓
2. 9 ✓
3. 4, 1, 2, 3, -1, 6, 9, 8, 12 ✓
4. 4 12 ✗
5. 1 2 6 ✗
6. 3, 2, 1, 4 ✗

Example: arr[ ] → [2 4 1 6 -3 7 8 4]

a. [1, 6, 8] ✗

b. [1, 4] ✗

c. [6, 1, 4, 2] ✗

d. [7, 8, 4,] ✓

Representation of Subarray:

Two ways:

① Start index and end index.

② Start index & length of subarray.

0 1 2 3 4 5 6 7 8

Array → { 4, 1, 2, 3, -1, 6, 9, 8, 12 }

SI → 2, EI → 5 { 2, 3, -1, 6 }

SI → 2, length → 4 { 2, 3, -1, 6 }

Quiz 4:

0	1	2	3	4	5	6
4	2	10	3	12	-12	15

SI = 0

SI	EI
0	0
0	1
0	2
0	3
0	4
0	5
0	6

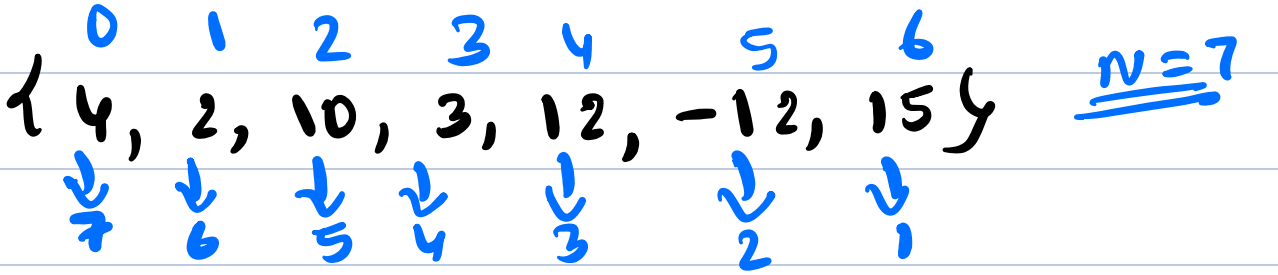
Ans = 7

Quiz 5:

0	1	2	3	4	5	6
4	2	10	3	12	-2	15
	SI					

SI	EI	
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]

Total number of subarrays



Starting index	Subarrays
0	n
1	n-1
2	n-2
3	n-3
⋮	⋮
n-1	1

$= 1 + 2 + \dots + n-3 + n-2 + n-1 + n$

= sum of first n natural no.

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

< *Question* > : Given an array, si and ei. Print from si to ei.

si ≤ ei

arr → [4 2 10 3 12 -2 15] si = 2, ei = 5

0 1 2 3 4 5 6

output : { 10, 3, 12, -2 }

- void printSubarray(arr, si, ei) {

```
for(i = si ; i<=ei ; i++)
{
    print(arr[i]);
}
```

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

}

print 1 subarray → T.C O(N)

< *Question* > : Print all the possible sub-arrays of the given array.

[5, 7, 3, 2]

0 1 2 3

O/P - [5]

[5, 7]

[5, 7, 3]

[5, 7, 3, 2]

[7]

[7, 3]

[7, 3, 2]

[3]

[3, 2]

[2]

Array = { 1, 2, 3 }
0 1 2

Ans: {1} {2} {3}
{1 2} {2 3}
{1 2 3}

```
void printAllSubarray(int[] arr)
{
    // Generate all subarrays.

    for(si=0; si<n; si++)
    {
        for(ei=si; ei<n; ei++)
        {
            for(i=si; i<=ei; i++)
            {
                print(A[i]);
            }
            // println() // Break line
        }
    }
}
```

TC: $O(N^3)$

SC: $O(1)$

Array = { 1, 2, 3 }
0 1 2

si=0 ei=0 i=0 1

↓

ei=1

↓

ei=2

si=1 ei=1 → ei=2

si=2 ei=2

Output:

1

1 2

1 2 3

2

2 3

3

Break: 10-32 pm

Min Max

< *Question* > : Given an array of N integers, return the length of smallest subarray which contains both maximum and minimum elements of the array.

$1 \le N \le 10^6$

arr[ ] → [2 2 6 4 5 1 5 2 6 4 1]

0 1 2 3 4 5 6 7 8 9 10

→ Ans = 3

Another eg.

A[] → {1, 2, 3, 1, 3, 4, 6, 4, 6, 3}

0 1 2 3 4 5 6 7 8 9

Ans = 4

Brute force:

→ Check all the subarray.

→ $O(N^3) * O(N) \Rightarrow O(N^3)$

[1, 2, 0, 4, 5, 5, 3]

Observation:

1) The answer subarray should contain only one instance of min and max element. (minimize the length)

2) The min and max value must be present at the corner of subarray.

3) so, we are looking for subarray that either
   a) start with max and end with min
   b) start with min and end with max.

Observation

1. There must be exactly one occurrence of min & max element.

   [min min - - - - max]

2. Min and max elements should be the end point of subarray.

3. *case-1*: [min - - - - max]

   case-2: [max - - - - min]

0 1 2 3 4 5 6 7 8 9 10

arr[ ] → [2 2 6 4 5 1 5 2 6 4 1]

i

smallest = 1

largest = 6

ans = ~~4~~ 3

latest_min_index = 5

latest_max_index = ~~2~~ 8

[8, 8, 8, 8, 8]

↳ Ans = 1

< / > Code

```
int minSubArray(int[] A)
{
    // minValue → min element from arr.
    // maxValue → max element from arr.
    int latest_min_index = -1
    int latest_max_index = -1
    int ans = //max value of int or N.
    for(i=0; i<N; i++)
    {
        if(A[i] == maxValue)
        {
            latest_max_index = i;
            if(latest_min_index != -1)
            {
                ans = min(ans, i - latest_min_index + 1)
            }
        }
    }
```

index latest/recent min value encountered

index of latest max value encountered

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

TC: O(N)

SC: O(1)

```
if(A[i] == minvalue)
{
    latest-min-index = i;
    if(latest-max-index != -1)
    {
        ans = min(ans, i - latest-max-index + 1);
    }
}
return ans;
}
```

curr length (i - latest-max-index)

curr

```
if (ans > length)
{
    ans = length
}
```

Doubt Session

```
for(i=0; i<n; i++)
{
    if (i%2==0)
    {
        print (even index)
    }
    print (Odd index);
}
```

i=0

even in
odd index