Problem solving 6

Prepared by Ahmed Ibrahim Shaaban

Sum of All Subarrays

	0	1	2	3	4
arr[] :	5	-1	-1	3	6

0 → 0:5 ; 1:4 ; 2:3 ; 3:9 ; 4:15

1 → 1:-1 ; 2:-2 ; 3:2 ; 4:8

2 → 2:-1 ; 3:3 ; 4:9

3 → 3:9 ; 4:15

4 → 4:6

0-2 : 5 -1 -1 = 3

0-3 : 3 + 3 = 6

0-4 : 6 + 6 = 12

Sum of All Subarrays : $O(n^2)$

BF: For every subarray get sum

```
ans = 0
for i = 0 to (N-1) for j = i to N {
    for k = i to j {
        sum = sum + arr[k]
    }
    ans = ans + sum
}
```

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

This part we can optimize using pre[] using pre[] to get sum from i..j : $O(1)$

TC: $O(N^2)$ SC: $O(1)$/$O(N)$

depends on how you implement

Left Index Right Index

	0	1	2	3	4	5	6
arr[i] :	3	1	6	4	7	8	10

Q: In how many subarrays is index 1 present ?

arr[i] : 3 1 6 4 7 8 10 (0 1 2 3 4 5 6)

2 6

Total = 12

Q: Generalize: How many subarrays contain index i ? Formula: $(arr[i] - b - a)$

$a_0 \; a_1 \; a_2 \; a_3 \ldots a_i \; a_{i+1} \; a_{i+2} \ldots a_{N-1}$

Left = [0 .. i] = i+1 ; Right = [i .. N-1] = N-i

Total Subarrays : $(i+1)(N-i)$

How many = 3*5 = 15

Left: 3 Ways to left

Right: 5 Ways to right

Prefix Code

```
sum = 0
for i = 0 to (N-1) for j = i {
    left = i+1
    right = N-i
    count = (i+1)(N-i)
    sum = sum + count * arr[i]
}
return sum
```

TC: $O(n)$ SC: $O(1)$

Idea: Adding contribution of each array element

2,8

	0	1	2	3	4	5	6
arr[i] :	3	-1	6	4	2	8	10

Subsequences:

→ Pick any random in array

→ Order is maintained

Index :	0	2	4	5
arr[] :	3	6	2	8

→ More value 8

	0	1	2	3	4	5	6
arr[i] :	3	-1	6	4	2	8	10

In how many subsequence will arr[i] + 4 be max ?

Subsequences

{3, -1}

{3, -1, 6}

{3, -1, 6, 4}

{3, 6, 4}

{3, 6, 2}

= 2^4 Subsequence will contain 4 has maximum

1c choices {6}, {6}, {6}

	0	1	2	3	4	5	6
arr[i] :	3	-1	6	4	2	8	10

In how many subsequence will arr[i] + 4 be max ?

8

// Q5: Sum of Max of All Subsequence

Q5: Calculate max of those each element will become minimum in a subsequence? & calculate contribution of the element.

To get max of elements less than that, sort the arr in increasing order

Pseudo code:

```
sort the array : N log N
ans = 0
for i = 0; i < N; i++{
    count = i + 0
    mx = arr[i]
    con = 2^i * arr[i]
    ans = ans + con
}
return ans
```

O(N)

overflow

1<<i

1<<100?

overflow

overall TC : N log N

```
ans = 0
(pow = 1)   mod = 10^9 + 7
for i = 0; i < N; i++{
    con = (pow * arr[i]) % mod
    ans = (ans + con) % mod
    pow = (pow * 2) % mod
}
```

// Sum of Min of All Subsequences?

Sum of Max of all Subsequence - Sum of Min of each Subsequence

Sum of (Max - Min) of Each Subsequence

Hint: In how many subsequence an element will be minimum

{10: 20 ; 7} 8 min break

Q6 Given N array, calculate element is either 0 or 1

	0	1	2	3	4	5	6	7
arr[i] :	1	0	0	0	1	0	1	0

Formula

a	b
0	0
0	1
1	0
1	1

2 types Q:

← 1 ___ R :

How many 0 from 1 → R

1 3 7 : 3

1 1 5 : 4

Ans: Calculate pre[]

pre[i] : How many 0s are there from [0, i]

Type 1 - Easy done

2 L → R :

Either or operator

Calculate OR of All Elements from L → R

2 2 6 : 1

2 1 3 : 0

Hint: Given L → R calculate no. of 1s

Then range using pre[] : O(1)

```
if ( x = R - L + 1 ){
    // L - R all elements are 0s
    print (0)
}
else { // There is atleast a single 1
    print (1) }
```

Now L → R Elements from L-R, R-L+1

Q8 Given N arr[], each element is either 0 or 1

Calculate no. of subarrays where OR = 0

	0	1	2	3	4	5	6	7
arr[i] :	1	0	0	0	1	0	1	0

0s

[0 .. 3] : 1

[1 .. 3] : 0

[1 .. 5] : 1

Obs: Your subarray should only contain 0s

Obs: It should contain only contain 0s

Logic:

For every subarray {sum or & equals = 0}

```
for i = 0; i < N; i++{
    for j = i; j < N; j++{
        calculate OR [i, j]
        // equals to 0
    }
}
```

optimize this part → $O(N)$

TC: $O(N^2)$

Ex1

	0	1	2	3	4	5	6	7
arr[i] :	1	0	0	0	1	0	1	0

C = 0

0s = 3

C = (3*4)/2

C = 6

0s = 2

C = (2*3)/2

C = 3

Given N, No. of Subarrays = $\frac{(N)(N+1)}{2}$

Ex2

	0	1	2	3	4	5	6	7
arr[i] :	0	1	0	0	0	1	0	1

0s = 1

count = 1

0s = 4

count = count + 10

0s = 2

count = 3

Edge Case

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

0s = 2

count = 3

→ we are not adding this to final ans.

```
// ans = 0, c = 0
for 0; i < N; i++{
    if (arr[i] == 0){
        c++
    }
    else { // arr[i] == 1,
        How number of subarrays which can be expected by c terms
        ans = ans + (c)(c+1)/2
        c = 0
    }
}
ans = ans + (c)(c+1)/2 → Edge Case
return ans
```

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

→ Calculate how many subarrays with OR = 1 ,

→ Total Subarrays - Subarrays with OR = 0

→ $\frac{(N)(N+1)}{2}$ - ans