October 11, 2025

BIT MANIPULATION APPLICATIONS

Ques 1. Given an array. Find :

a. SUM of (AND of all subarrays) of the array.

b. SUM of (XOR of all subarrays) of the array.

[3, 2, 7, 1]

Example :

If you take subarray (2, 7, 1)
The ans of i) will be 1 & 2 & 7 = 0
For subarray (3, 2), ans i) = 3 & 2 = 2
Similarly we can continue finding the AND or XOR for all possible subarrays.
Then, the answer will be the sum of it all.

Hint : Use of contribution Technique

Atomic Item

Extending Ends
This we'll use.

Solution i)

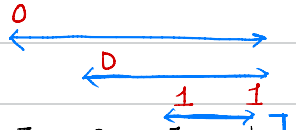
Arr[] :	3	2	7	1
Binary Format (MSB→)	0	0	1	0
	1	1	1	0
(LSB→)	1	0	1	1

Now, let's say our subarray is ending at the last value of array 1.

{3, 2, 7, 1}

Using Binary Format, let's calculate the AND values.

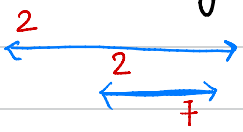
Value of AND at 1 (001) : 1
Value of AND of 1 & 7 : 1
Value of AND of 1 & 7 & 2 : 0 (Now, AND will only result in 0)
Value of AND of 1 & 7 & 2 & 3 : 0



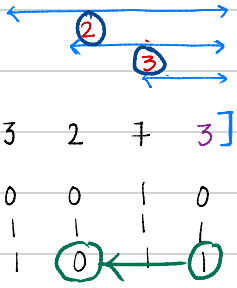
0	0	1	0
1	1	1	0
1	0	1	1

Here, we could only generate two values : 0 & 1.

Now, let's say our subarray is ending at 7.



0	0	1	0
1	1	1	0
1	0	1	1



← Why did this 3 became 2 here when doing AND?

The transition of 1 in LSB to 0 in LSB at 2 made our overall AND 2 from 3.

For every 1, the first 0 we see (tracing right → left) is where that 1 will first time turn 0 & then it will remain 0 for all future subarrays.

The possible number of bits in a number is 32 bits so we can say that only 32 distinct AND values are possible.

There are only logN different values.

```
    cin>>n;
    int arr[n];
    for(int i=0;i<n;i++)cin>>arr[i];

    map<int,int> andfreq;
    for(int i=0;i<n;i++){
        map<int,int> curandfreq;
        for(auto v:andfreq){
            curandfreq[v.first & arr[i]]+= v.second;
        }
        curandfreq[arr[i]]+=1;
        // we have freq of every and possible at arr[i];
        for(auto v:curandfreq){
            cout<<v.first<<" : "<<v.second<<endl;
        }
        cout<<endl;

        andfreq = curandfreq;
    }
}
```

Solution i>

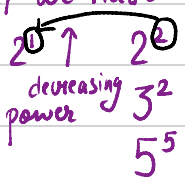
```
void solve(){
    int n;
    cin>>n;
    int arr[n];
    for(int i=0;i<n;i++)cin>>arr[i];

    int ans = 0;
    map<int,int> andfreq;
    for(int i=0;i<n;i++){
        map<int,int> curandfreq;
        for(auto v:andfreq){
            curandfreq[v.first & arr[i]]+= v.second;
        }
        curandfreq[arr[i]]+=1;
        // we have freq of every and possible at arr[i];
        for(auto v:curandfreq){
            ans += v.first * v.second;
        }

        andfreq = curandfreq;
    }
    cout<<ans<<endl;
}
```

→ Same can be done for GCD.

Let's say we have



The only way to get a new GCD is when the power will decrease.

Solution ii> # of subarr with XOR=1 (odd # of 1's)

[0, 0 | 1, 0]

how many Subarray ends here which has odd no. of 1's

Odd = 0

Even = 2

[0, 0, 1 | 0]

odd += 2

even += 0

odd += 1

If we maintain, at a point how many odd 1's subarrays are ending and the even 1's then we can use it Solve the problem.

```cpp
int get_odd_1_subarrays(int n,int arr[]){
    int ans = 0;

    int lodd = 0;
    int leven = 0;
    for(int i=0;i<n;i++){
        int codd = 0;
        int ceven = 0;
        if(arr[i]==1){
            codd = leven+1;
            ceven = lodd;
        }else{
            codd = lodd;
            ceven = leven+1;
        }
        // how many subarrays end at i, has odd 1's and even 1's
        ans += codd;

        lodd = codd;
        leven = ceven;
    }
    return ans;
}
```

```cpp
void solve(){
    int n;
    cin>>n;
    int arr[n];
    for(int i=0;i<n;i++)cin>>arr[i];

    int ans = 0;
    for(int pos=0;pos<30;pos++){
        int temp[n];
        for(int i=0;i<n;i++){
            if(arr[i]&(1<<pos)){
                temp[i]=1;
            }else{
                temp[i]=0;
            }
        }
        int cnt = get_odd_1_subarrays(n,temp);
        ans += cnt * (1<<pos);
    }
    cout<<ans<<endl;

}
```

$\overline{1}$ $\overline{3}$

0 1

1 1

Just 1	Just 3	XOR
0	1	1
1	1	0

When we add them,

1 + 3 + 2 = 6

Instead of this, count number of 1's in the generat subarray at a particular position.

$O(N^2)$

Just 1	Just 3	XOR	
0	1	1	$= 2 \times 2^1$
1	1	0	$= 2 \times 2^0$
			6

$O(N)$

[1, 3, 2]

0 1 1 $3 \times 2^1 = 9$

→ [1 1 0] 3×2^0

3 2 1

→ [1 1 0] 3 subarrays which has odd no of 1's.

AlgoZenith