Content

Insert into non overlapping intervals.

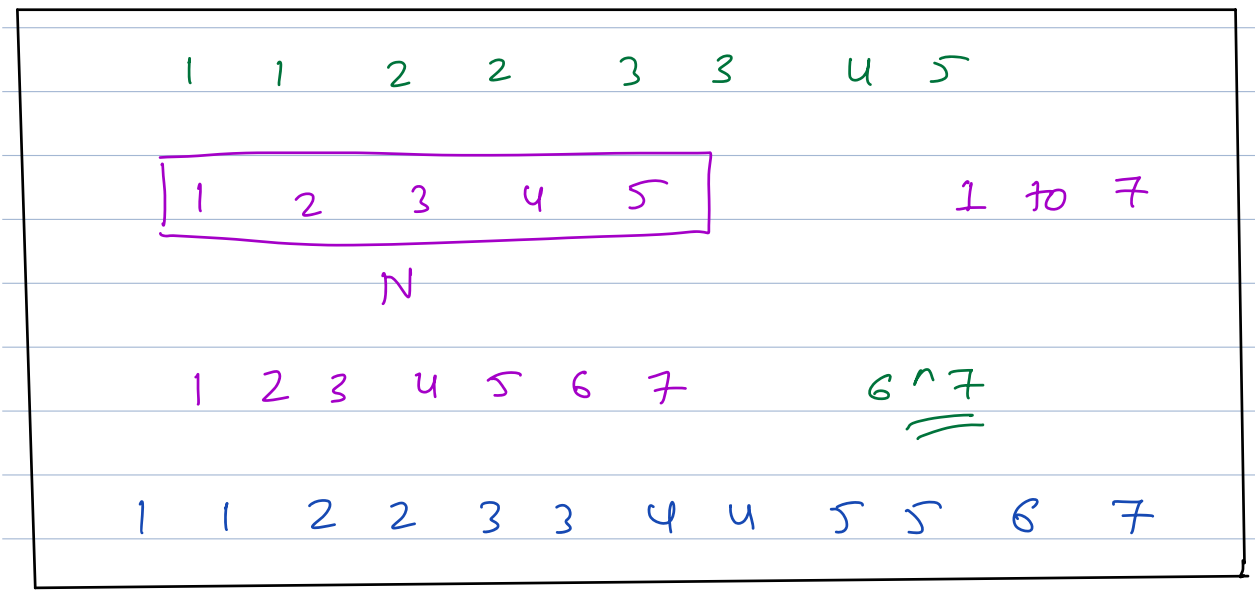
```java
public class Solution {
    public ArrayList<Interval> insert(ArrayList<Interval> intervals, Interval newInterval) {
        int L = newInterval.start;
        int R = newInterval.end;

        ArrayList<Interval> ans = new ArrayList<>();

        for(int i = 0; i < intervals.size(); i++){
            int endi = intervals.get(i).end;
            int starti = intervals.get(i).start;

            if(endi < L){
                // print(starti, endi)
                ans.add(new Interval(starti, endi));
            }
            else if(R < starti){
                // print(L, R)
                ans.add(new Interval(L, R));
                // Add the rest of the intervals remaining
                for(int j = i; j < intervals.size(); j++){
                    ans.add(intervals.get(j));
                }
                return ans;
            }
            else{ // Merge the overlapping intervals
                L = Math.min(L, starti);
                R = Math.max(R, endi);
            }
        }
        ans.add(new Interval(L, R));
        return ans;
    }
}
```

merge overlapping intervals

```java
public class Solution {
    public ArrayList<Interval> merge(ArrayList<Interval> intervals) {
        intervals.sort((a, b) -> {
            return a.start - b.start;
        });

        int S = intervals.get(0).start;
        int E = intervals.get(0).end;

        ArrayList<Interval> ans = new ArrayList<>();

        for(int i = 1; i < intervals.size(); i++){
            int starti = intervals.get(i).start;
            int endi = intervals.get(i).end;

            if(E < starti){
                // print S, E
                ans.add(new Interval(S, E));
                S = starti;
                E = endi;
            }
            else{
                E = Math.max(E, endi);
            }
        }

        ans.add(new Interval(S, E));

        return ans;
    }
}
```

```
module.exports = {
    /**
     * Interval: [start, end]
     *
     * param A: intervals, a list of Intervals
     * return :a list of Intervals
     */
    merge : function(A){
        A.sort((a, b) => a[0] - b[0]);

        let S = A[0][0];
        let E = A[0][1];

        let ans = [];

        for(let i = 1; i < A.length; i++){
            let starti = A[i][0];
            let endi = A[i][1];

            if(E < starti){
                // print(S, E)
                ans.push([S,E]);
                S = starti;
                E = endi;
            }
            else{
                E = Math.max(E, endi);
            }
        }
        ans.push([S, E]);
        return ans;
    }
};
```

Given an A[] where all elements are distinct and in range 1 to N+2
two no. from range [1 to N+2] are missing
find the two missing no.



Search in row wise col wise matrix

```java
public class Solution {
    public int solve(int[][] A, int B) {
        int R = A.length;
        int C = A[0].length;

        int r = 0;
        int c = C - 1;

        /* B 3
        [[3, 3, 3]
        [4, 5, 6]
        [7, 8, 9]]
        */

        // return minimum value of r*1009 + c
        int ans = Integer.MAX_VALUE;

        while(r < R && c >= 0){
            if(A[r][c] == B){ // Found the ans
                ans = Math.min(ans, (r+1)*1009 + (c+1));
                c -= 1;
            }
            else if (A[r][c] < B) {
                r += 1;
            }
            else{
                c -= 1;
            }
        }

        if(ans == Integer.MAX_VALUE) return -1;

        return ans;
    }
}
```

Subarray OR

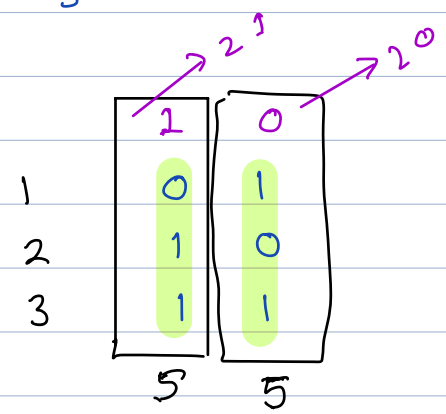
Given A[N]

V(subarray) ⟶ OR of all values in it.

Sum of all V(subarray) for all posible subarrays.

A = 1 2 3

1			0	1	1	
1	2		1	1	3	gm = 15
1	2	3	1	1	3	
2			1	0	2	
2	3		1	1	3	
3			1	1	3	



$2*5+5 = 15$

$2^1*5 + 2^0*5 = 15$

Sum of all subarray OR of A = [1 0 1]

1	⟶	1
1 0	⟶	1
1 0 1	⟶	1

gm = 5

subarray with OR == 1

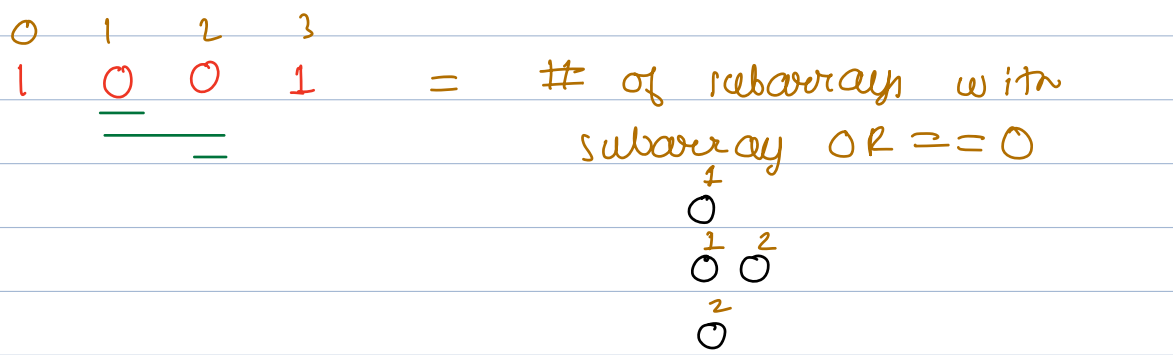
If my subarray contains atleast one '1'
⟶ entire OR = 1.

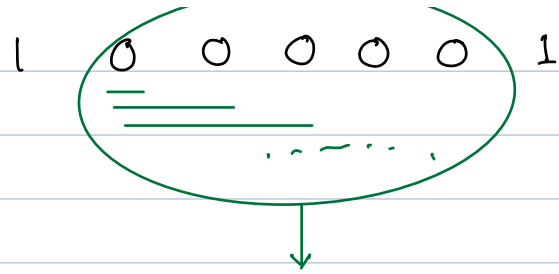
If my subarray contains only 0s
⟶ OR of subarray = 0

Total # of subarrays with OR == 1

Total subarray — # subarrays with only 0s in them.

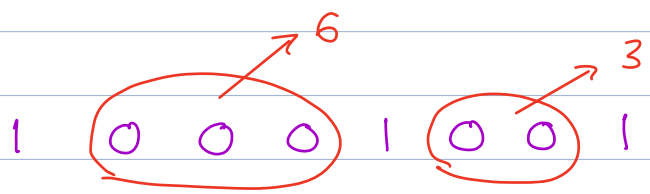
$$\frac{N*(N+1)}{2}$$





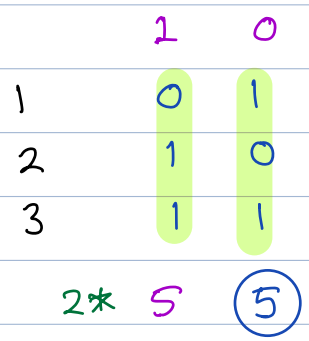
cnt2 = 5

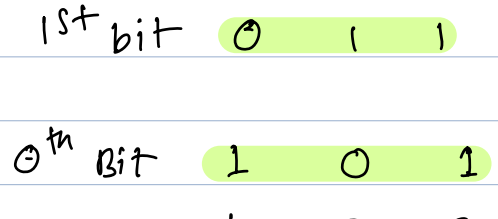
$\frac{5*6}{2} = 15$



$= 6+3 = 9$

subarray with OR = 0





subarray with OR = 1 $\Rightarrow$ Total subarrays — # subarray with OR=0

$= 6 - 1 = 5$

Pseudocode

```
any = 0

for bit ⟶ 0 to 31 {
    // To make implementation simpler first
       solve using bit array of each bit
       position, then remove extra space.
    binary = int[N]
    for i ⟶ 0 to N-1 {
        if (checkBit(A[i], bit)){
            binary[i] = 1
        }
    } else { binary[i] = 0 }

    // count of # subarrays with OR = 0
    zero = 0   // cnt #subarrays with OR = 0
    s = 0 // streak of continuous 0's

    for i ⟶ 0 to N-1 {
        if (binary[i] == 0) {
            s += 1
        }
        else {
            zero += s*(s+1)/2
            s = 0
        }
    }
```

```
    zero += s*(s+1)/2   // 000
    ones = n*(n+1)/2 - zero
    ans += (1 << bit) * ones
                2^bit
}   ans %= MOD   {10^9+7}.
```

print(ans)

TC : O(N)

SC : O(N)

2nd

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

ans 0 =

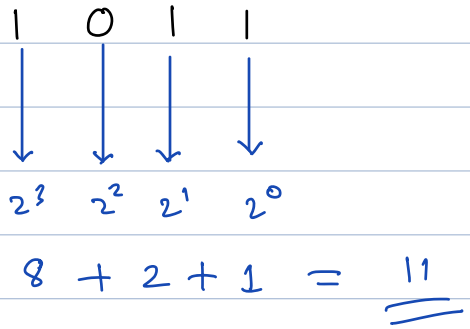
ans 1 =

4 ^ 2 = (1)(1) 0

Group 1

3 3 7 7 2

Group 0 5 5 4



Maximum AND Pair

```
module.exports = {
 //param A : array of integers
 //return an integer
    solve : function(A){
        let ans = 0;

        for(let bit = 31; bit >= 0; bit--){
            let count = 0;

            for(let i = 0; i < A.length; i++){
                let pow = BigInt((1 << bit));
                if((A[i] & pow) > 0){ // (A[i] >> bit) & 1
                    count += 1;
                }
            }

            if(count >= 2){
                ans |= (1 << bit);

                // reset the values to 0 which cannot be used
                for(let i = 0; i < A.length; i++){
                    let pow = BigInt((1 << bit));
                    if((A[i] & pow) == 0){
                        A[i] = BigInt(0);
                    }
                }
            }
        }

        return ans;

    }
};
```