Todays Content:

8:41 → 8:50

- Max and pair ✓
- Sum of all subarray OR ✓
- Subsets vs Subsequences
- Subset with given sum == k { if Time permits }

Given ar[N], find max ar[i] & ar[j] which can be obtained?

↳ i != j

Eg1: ar[3] = { 27 18 20 } ans = 18 (indices 0 1 2)

27 : 1 1 0 1 1 — 27 & 18 = 18

18 : 1 0 0 1 0 — 27 & 20 = 16

20 : 1 0 1 0 0 — 18 & 20 = 16

Eg2: ar[5] = { 21 18 24 17 16 } : ans = 17 (indices 0 1 2 3 4)

21 : 1 0 1 0 1 — 21 & 17 = 17

18 : 1 0 0 1 0 — 21 & 24 = 16

24 : 1 1 0 0 0 — 24 & 18 = 16

17 : 1 0 0 0 1

16 : 1 0 0 0 0

ideas:

→ Calculate & for all pairs & get max

TC: O(N²) SC: O(1)

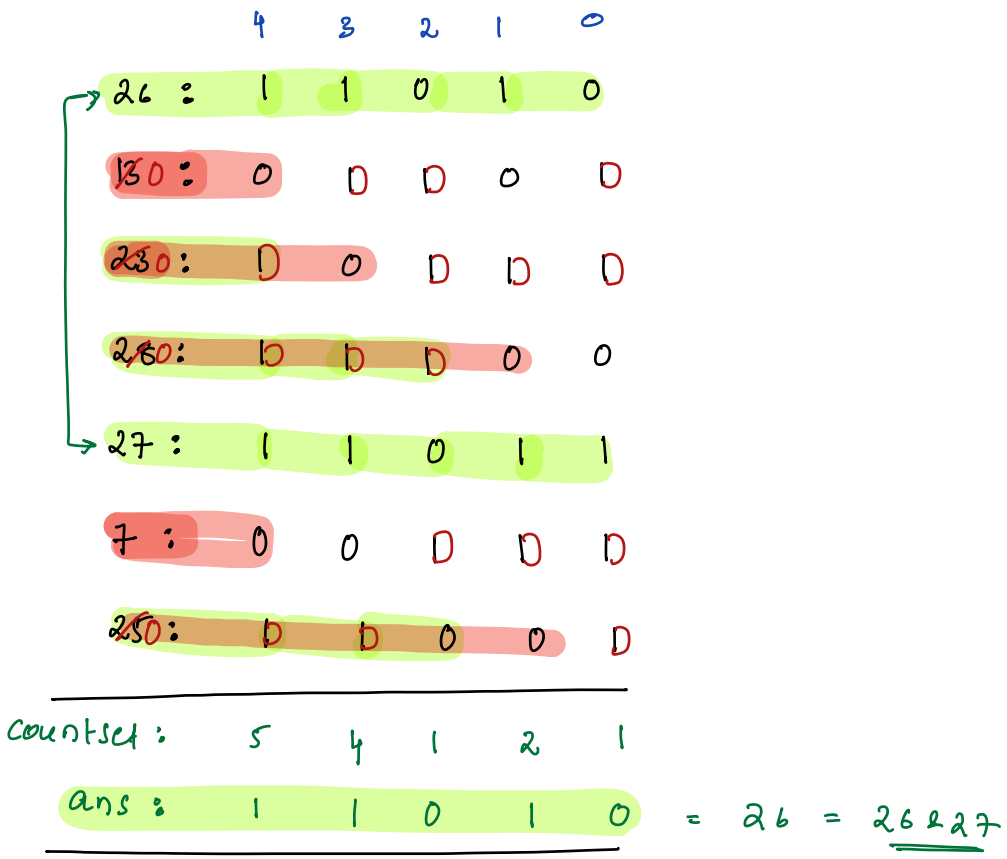
→ Sort ar[] : ar[5] = { 16 17 18 21 24 } (17 & 21)

- → & with 1st & 2nd max : *
- → one of the element is max : *
- → Get adjacent elements & *

→ idea: 1 0 0 0 0 vs 0 1 1 1 1

obs: left side bits have higher significance

Eg: ar[7] = { 26 13 23 28 27 7 25 }



Pseudocode :

TC: $32 * [N + N] \rightarrow 64N$

SC: $O(1) \rightarrow O(N)$

```
int maxAnd(int ar[], int n){
    int ans = 0;
    for(i = 31; i >= 0; i--){
        // no. of elements with ith bit set
        int c = 0
        for(j = 0; j < n; j++) {
            if(checkbit(ar[j], i)) {c = c+1}
        }
        if(c >= 2) {
            // we can set ith bit in our ans = 1
            ans = ans + (1 << i);
            // iterate in ar[] & discard elements with ith bit = 0
            for(j = 0; j < n; j++){
                if(checkbit(ar[j], i) == false)
                    ar[j] = 0
            }
        }
    }
    return ans;
}
```

obs: Once entire ar[] done, pair is non-zero elements in ar[]

Q) get max and value achieved ? → if(c >= 4)

↳ Google / Hard

obs: In general bit manipulations can be solved by bit by bit

20) Given a binary ar[], calculate no: of subarrays whose OR = 0

(binary: 1/0)

Ex: ar[5] : 1 0 1 0 0 (indices 0 1 2 3 4)

subarrays: ans = 4

[1 1] [3 3]

[3 4] [4 4]

Obs: Subarray can contain only 0s

Note: With N elements, no of subarrays are $\frac{N*(N+1)}{2}$

Ex: ar[10] : 1 1 0 0 0 1 0 1 0 0 (indices 0 1 2 3 4 5 6 7 8 9)

= = 3 ele * 1 ele * 2 ele

Total Sub

6 Sub ⟷ 1 Sub ⟷ 3 Sub ⟶ 10

Ex: ar[10] : 0 0 0 1 0 0 1 1 0 0 (indices 0 1 2 3 4 5 6 7 8 9)

c = 0 : 1 2 3 0 1 2 0 0 1 2

After end : we need update ans

ans = ans + 6 (c = 0) ans = ans + 3 (c = 0) ans = ans + 3 = 12

int countSeqOR(int ar[], int n){ TC: O(N) SC: O(1)

```
int ans = 0, c = 0
for(i=0; i<n; i++){
    if(ar[i]==0){c=c+1}
    else{ans = ans + c*(c+1)/2, c=0}
}
ans = ans + (c*(c+1))/2
return ans;
```

Subarray with OR = 1 : Total subarrays - Subarray with OR = 0

return N*(N+1)/2 - ans

3(g) Given ar[N] calculate sum of all subarray OR's

Ex: ar[3] = { 7 4 8 } (0 1 2) : ans = 53

Subarrays

[0 - 0] = {7} = 7

[0 - 1] = {7|4} = 7

[0 - 2] = {7|4|8} = 15

[1 - 1] = {4} = 4

[1 - 2] = {4|8} = 12

[2 - 2] = {8} = 8

idea: For every subarray calculate its OR & add in final ans

TC: O(N²) * O(N) → O(N³)

↓

optimize to O(N²) TODO

↓

idea2: Apply Contribution technique bit by bit?

Ex: ar[6] = { 5 8 26 13 5 21 } → ans = 490

| | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|
| 5 = | 0 | 0 | 1 | 0 | 1 |
| 8 = | 0 | 1 | 0 | 0 | 0 |
| 26 = | 1 | 1 | 0 | 1 | 0 |
| 13 = | 0 | 1 | 1 | 0 | 1 |
| 5 = | 0 | 0 | 1 | 0 | 1 |
| 21 = | 1 | 0 | 1 | 0 | 1 |

In how many subarray OR 0th bit = 1 → 18 0th bit → 18 * $2^0$ = 18

bo[6] = { 1 0 0 1 1 1 }, subarrays with OR = 1

// Total Subarrays

// Subarrays with OR = 0

= 21 - 3 = 18

In how many subarray OR 1st bit = 1 → 12 1st bit → 12 * $2^1$ = 24

bo[6] = { 0 0 1 0 0 0 }

// Subarray with OR = 0

= 21 - 9 = 12

In how many subarray OR 2nd bit = 1 → 18 2nd bit → 18 * $2^2$ = 72

bo[6] = { 1 0 0 1 1 1 }

// Subarray with OR = 0

= 21 - 3 = 18

In how many subarray OR $3^{rd}$ bit = 1 → 17 $3^{rd}$ bit → $17 * 2^3 = 136$

bo[6] = { 0 1 1 1 0 0 }

↳// subarray with OR = 0

= 21 - 4 = 17

In how many subarray OR $4^{th}$ bit = 1 → 15 $4^{th}$ bit → $15 * 2^4 = 240$

bo[6] = { 0 0 1 0 0 1 }

↳// subarray with OR = 0

= 21 - 6 = 15

Pseudocode:

TC: 32 * [N + N] → O(N)

int subsumOR (int ar[], int n) { SC: O(N) → TODO O(1)

To store info about $i^{th}$ bit

```
int ans = 0
for(i = 0; i < 32; i++){
    // no: of subarray OR with ith bit set
    // Store all ith bit of ar[] element in b[]
    int b[n] // binay array
    for(j = 0; j < n; j++)
        b[j] = checkbit(ar[j], i)

    ans = ans + countSub1(b) * (1<<i)
}
return ans;
```

Given binay ar[], calculate & return no: of subarrays with OR = 1

countSub1(b) → Total Contribution of $i^{th}$ bit

$2^i$

}

Subarray : continous part of an array

Subsequence: A sequence obtained deleting none a more elements from array.

→ Data should be arranged based on increasing order index

→ Empty subsequence is valid

| | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| ar[6] = | 3 | 2 | 1 | 9 | 6 | 8 |

* * ✓ * ✓ ✓ → 1 6 8 ✓

✓ * * ✓ ✓ * → 9 6 3 * not ordered

↳ 3 9 6 ✓

* * ✓ ✓ ✓ ✓ → 1 9 6 8 ✓

x ✓ ✓ ✓ ✓ x → 2 1 9 6 ✓

x x x x x x → { } ✓

✓ ✓ ✓ ✓ ✓ ✓ → 3 2 1 9 6 8 ✓

obs:

→ Every subsequence is subarray : False

→ Every subarray is subsequence : True

ar[3] = { 3 1 8 } —— Sort arr[] ——→ ar[] = {1 3 8}

| Subsequence: | | Subsequence: |
|---|---|---|
| { } | | { } |
| {3} | | {3} |
| {1} | | {1} |
| {8} | | {8} |
| {3 1} | Not same, order of data is diff | {1 3} |
| {3 8} | | {3 8} |
| {1 8} | | {1 8} |
| {3 1 8} | Not same, order of data is diff | {1 3 8} |

Obs: Subsequence changes, if sort arr[]

↳ Take aways:

1) Order changes
2) Data remains same

Q] Sum of every subsequence
Max of every subsequence
Min of every subsequence

} Then will only depend on data, we can still sort since data remains same

**Subset:** Same as subsequence order doesn't matter.

ar[3] = { 3 1 8 } —Sort arr[]→ ar[] = { 1 3 8 }

2 * 2 * 2 = 8

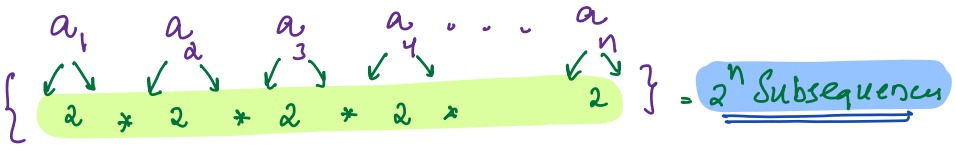
| Subset | Subset |
|---|---|
| { } | { } |
| {3} | {3} |
| {1} | {1} |
| {8} | {8} |
| {3 1} | {1 3} |
| {3 8} | {3 8} |
| {1 8} | {1 8} |
| {3 1 8} | {1 3 8} |

**Obs:** Sorting won't effect subsets of an array

| | 0 | 1 | 2 | 3 | 4 | 5 | | |
|---|---|---|---|---|---|---|---|---|
| ar[6] = | 3 | 2 | 1 | 9 | 6 | 8 | | |
| | * | * | ✓ | * | ✓ | ✓ | → | 1 6 8 : Subset |
| | ✓ | * | * | ✓ | ✓ | * | → | {9 6 3} |
| | | | | | | | | {3 6 9} |
| | | | | | | | | {3 9 6} |
| | * | * | * | * | * | * | → | { } : Subset ✓ |

Same, order won't matter

ar[N]: How many subsequence are present



Q) Given ar[N], check if there exists a subsequence with sum = k

ar[8] = { 6 8 3 2 4 1 9 10 }

k = 18 → { 8 1 9 }, { 8 10 } ...

k = 30 → { 6 8 2 4 10 } ...

Idea: For every subsequence, iterate & get sum == k

TC: $O(2^n * N) \Rightarrow O(2^N * N)$

Ex: ar[3] = { 9 2 7 }  (0 1 2)

N = 3 subsequence = 8 : [0 7]

N = 4 subsequence = 16 : [0 15]

N subsequence = $2^N$ : [0, $2^N - 1$]

| | 2 | 1 | 0 | |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | → { } |
| 1 | 0 | 0 | 1 | → {9} |
| 2 | 0 | 1 | 0 | → {2} |
| 3 | 0 | 1 | 1 | → {9 2} |
| 4 | 1 | 0 | 0 | → {7} |
| 5 | 1 | 0 | 1 | → {9 7} |
| 6 | 1 | 1 | 0 | → {2 7} |
| 7 | 1 | 1 | 1 | → {9 2 7} |

number → subsequence

Example

5 → 2 1 0 / 1 0 1

{ ar[2] * ar[0] }
{ 7 9 }

6 → 2 1 0 / 1 1 0

{ ar[2] ar[1] * }
{ 7 2 }

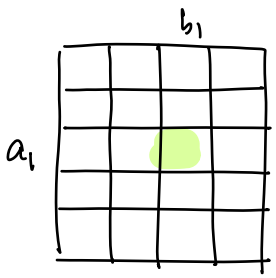
```
bool subsum(int ar[], int n, int k){ TC: O(2^N * N)
                                      SC: O(1)
    for(i=0; i<(1<<n); i++){
        // number i, map to subsequence
        // convert number i -> n bit number
        sum = 0
        for(j=0; j<n; j++){ // bit position i -> {subsequence}
            if(checkBit(i, j)){
                // In this subsequence take j th element
                sum = sum + ar[j]
            }
        }
        if(sum == k){ return True }
    }
    return false;
}
```

Dryrun: ar[3] = { 9 2 7 }

(0 1 2)

| | i: 2 1 0 | sum |
|---|---|---|
| 0 → | { 0 0 0 } | { } = 0 |
| 1 → | { 0 0 1 } | { 9 } = 9 |
| 2 → | 0 1 0 | { 2 } = 2 |
| 3 → | 0 1 1 | { 9 2 } = 11 |
| 4 → | 1 0 0 | { 7 } = 7 |
| 5 → | 1 0 1 | { 9 7 } = 16 |
| 6 → | 1 1 0 | { 2 7 } = 9 |
| 7 → | 1 1 1 | { 9 2 7 } = 18 |

Doubts




```
TL = (a1, b1)
a1 = 0; a1 < n; a1++){
    b1 = 0; b1 < m; b1++){
        TL = (a1, b1)
        a2 = a1; a2 < n; a2++){
            b2 = b1; b < m; b2++){
            TL = (a1, b1)  BR = (a2, b2)
            sum = 0
            i = a1; i <= a2; i++){
            |   j = b1; j <= b2; j++){
            }   sum = sum + mat[i][j]
            print(sum)
```