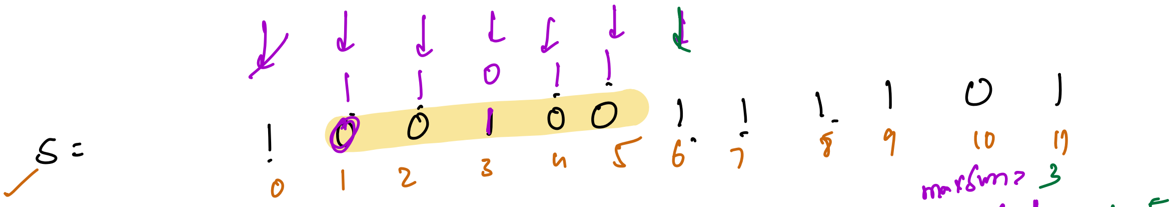
Agenda

1) Flip
2) Search for elements in row-wise & col-wise sorted matrix
3) Merge Overlapping Intervals
4) Kadane's AlgoC Max sum subarray

Question: Flip

Given a binary string (0's, 1's) , maximise the count of 1's in the string by flipping any substring.



maxSum = 3

Start = ~~0~~ 1 , end = 5

sum = ~~0~~ ~~1~~ ~~2~~ ~~1~~ ~~2~~ 3

[4, 5] = 9

[1, 5] = 10 ✓

[1, 10] =

S = 1 1 1 1 1 1
0 1 2 3 4 5

=> $\boxed{\text{Ans} = 6}$

Brute Force

Consider all the substrings : $\frac{N(N+1)}{2}$ ✓

we Flip the substring & count #1s : O(N)

$(i, j) \to \boxed{flip}$ $\underbrace{(i, j)}$

Total T.C : $O(N^3)$

S.C: $O(1)$

orig_ones = 5
max_ones = 5

Approach 2:

5-1+0 = 4

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

i=0, j=1

count0 = 0
count1 = 0

i=0, j = 0,1,2.... 7

[0,0]
[0,1]

Flip
i j
| 1 0 1 1 0 0 1 1 |

#ones = 5 (p)
#zeros = 3 (q)

x = total #1s in the string

$\boxed{x - p + q}$

#ones in orig substring

$\boxed{p, q}$

```
// ones_orig
ans·

for(i=0; i<N; i++) {
    count0 = 0, count1 = 0;
    for(j=i; j<N; j++) {        (i, j)
        if( s[j] == '0') count0++;
        else count1++;
        ans = max(ans, ones_orig - count1 + count0);
    }
}
return ans;
```

#ons + $\boxed{\text{count0-count1}}$

T.C: $O(N^2)$

S.C: $O(1)$

Approach 3:

We want a substring with   1 | 0000 1 | 0000 | 1

a) More no. of 0's (#zeros) - (+1)

b) less no. of 1's (#ones) (-1)

maximum (#zeros - #ones) ✓

[1 0 0 1 0 1 0 0]

diff = (-1) + 1 + 1 (-1) +1 -1 +1 +1

= $\boxed{2}$ → Maximized

Convert 0's ⟹ +1

1's ⟹ -1

S = 1 0 0 1 0 0 1 1 1 1 0 1

S' = [-1 1 1 -1 1 1 -1 -1 -1 -1 1 -1] ✓

sum = 3

⟹ Find the maximum sum subarray from this

Kadane's Algo: $\boxed{O(N)}$

S.C: O(1)

Ans = $\boxed{\text{maxSum}}$ + # orig_ones

Question: Given a row-wise & col-wise sorted matrix, search for a given element

[K = 18]

$$A = \left[\begin{array}{ccccc} \log N & \log N & \log N & \log N & \\ 5 & 10 & 15 & 20 & \rightarrow \log m \\ 6 & 12 & 20 & 23 & \rightarrow \log m \\ 7 & 14 & 21 & 30 & \rightarrow \log m \\ 17 & 26 & 33 & 48 & \rightarrow \log M \end{array}\right]$$

4 x 4

N x M

K = 14 → True

K = 22 → False

Brute Force:

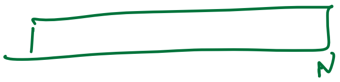
Iterate over entire matrix.

```
for (i=0 → N) {
    for (j=0 → M) {
        if(A[i][j] == K)
            True
    }
}
False
```

T.C: O(N × M)

S.C: O(1)

✓ **Approach 2:** Binary Search on Rows



⇒ O(log N)

Total T.C: O(N log M)

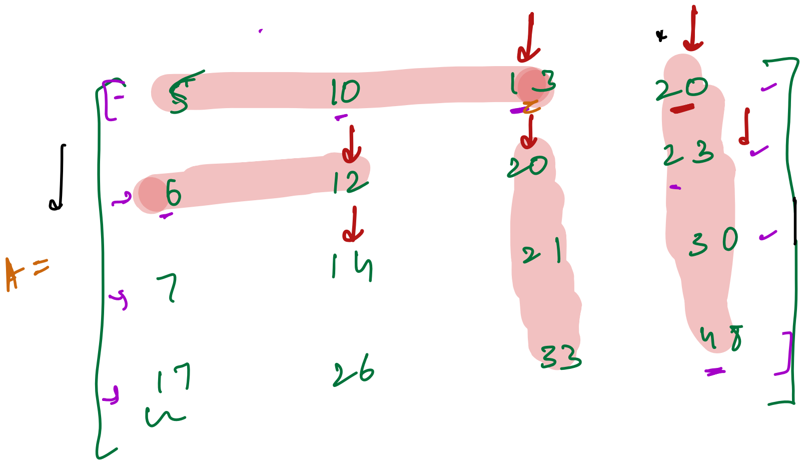
T.C: O(N. log M)

✓ **Approach 3:** Binary search on col

O(M. log N)

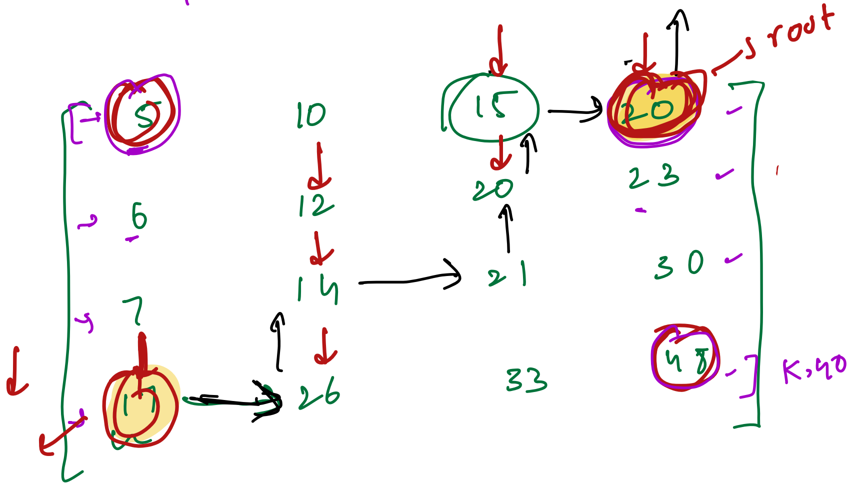
Approach 4:

K = 14



$$A = \begin{bmatrix} 5 & 10 & 13 & 20 \\ 6 & 12 & 20 & 23 \\ 7 & 14 & 21 & 30 \\ 17 & 26 & 33 & 48 \end{bmatrix}$$

K = 10



$$\begin{bmatrix} 5 & 10 & 15 & 20 \\ 6 & 12 & 20 & 23 \\ 7 & 14 & 21 & 30 \\ 17 & 26 & 33 & 48 \end{bmatrix}$$

K = 16

K = 18

i < 0 || j ≥ N

T.C: O(M+N)

(0, M-1) → (N-1, 0)

N+M cells

$\boxed{B.S.T}$

Timings

⇓

9:30 AM IST

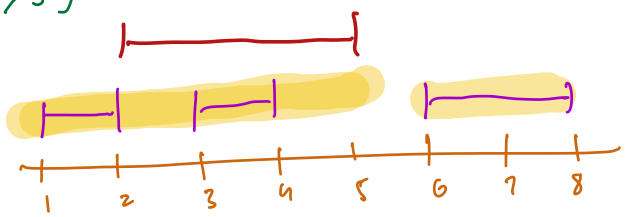
Question: Merge Overlapping Intervals

→ Given non-overlapping interval in a sorted order

→ Given a new interval. Add this new interval to the existing ones & return the new set of intervals

A: [1, 2] [3, 4] [6, 8]

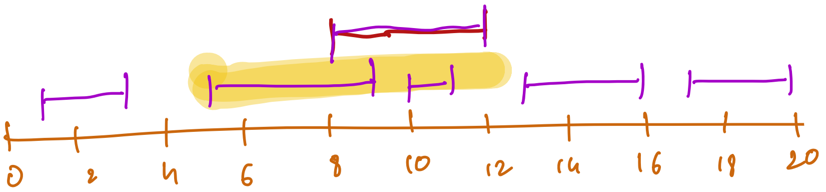
I = [2, 5]



Ans = [[1, 5], [6, 8]]

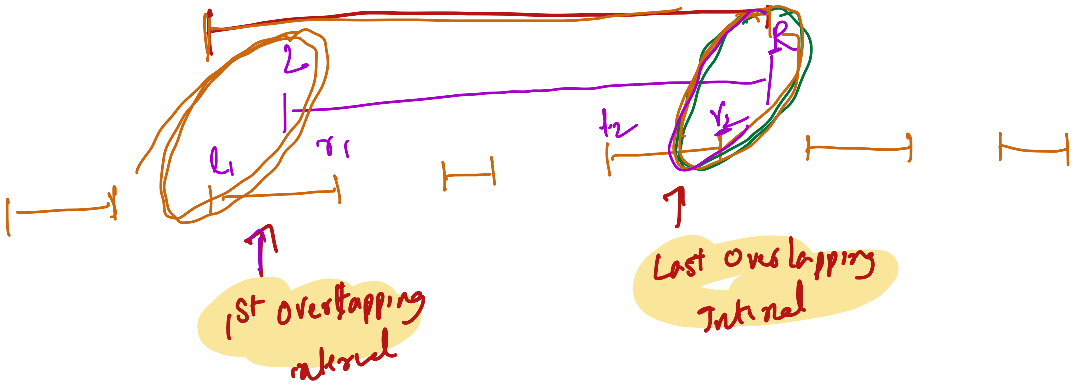
A = [1, 3] [5, 9] [10, 11] [13, 16] [17, 20]

I = [8, 12]



Ans = [[1, 3] [5, 12] [13, 16] [17, 20]]

Solution:



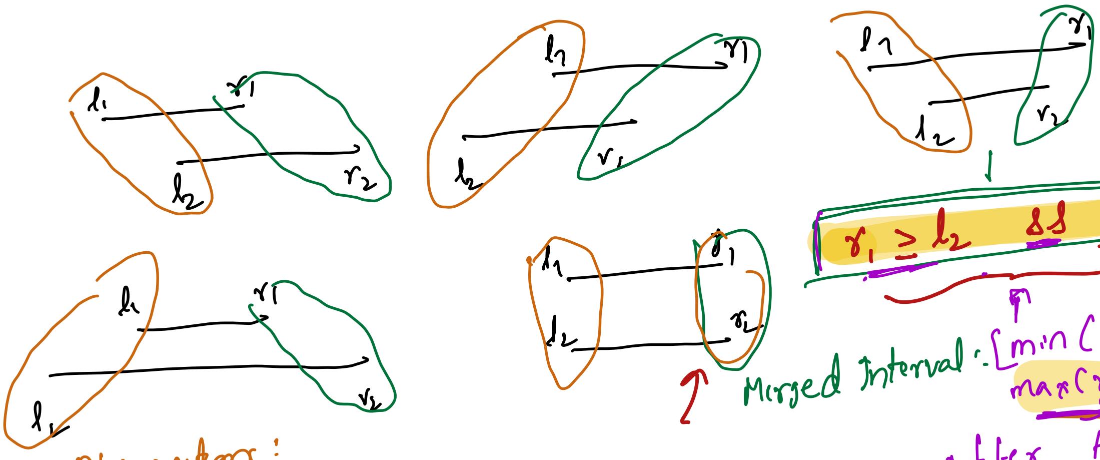
→ Find 1st overlapping interval ✓

→ Find last overlapping interval ✓

→ Merge the Interval

2 interval

(l_1, r_1) (l_2, r_2) $\{r_1 \geq l_1$ && $r_2 \geq l_2\}$



$r_1 \geq l_2$ && $r_2 \geq l_1$

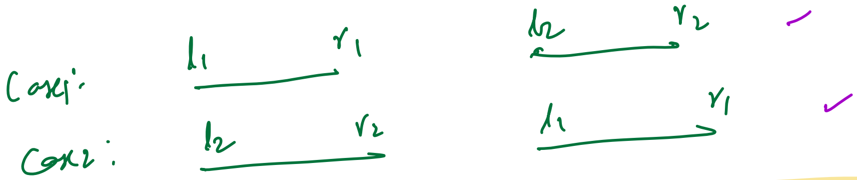
Merged Interval: $[\min(l_1, l_2), \max(r_1, r_2)]$

Observations:

→ 1st Interval has to stop after the 2nd interval starts $r_1 \geq l_2$

→ 2nd Interval has to stop after the 1st interval starts $r_2 \geq l_1$

No Overlap (l_1, r_1) (l_2, r_2)



if($l_2 > r_1$ || $l_1 > r_2$)

→
```
for(i=0;  i<N;  i++) {
    if( isOverlap( Interval[i],  I ) ) {
        first_overlap = i; }
        break;
    }                                   } O(N)
}

for(i=N-1;  i>=0;  i--) {
    if( isOverlap( Interval[i],  I ) ) {
        last_overlap = i; }
        break;
    }                                   } O(N)
}
```

First_Overlap = (l_1, r_1)

Last_Overlap = (l_2, r_2)

Interval = (L, R)

Merged Interval: → O(1)

$[\min(l_1, L) , \max(r_2, R)]$

⇒ [ ]

A = [[1,3] [5,9] [10,11] [13,16] [17,20]] → O(N)

I = [8,12]

A' = [[1,3] [5,12] [13,16] [17,20]]

output

S·C: O(1)

T·C: O(N)
S·C: O(1)

Kadane's Algo)

→ Given an array, find the max sum subarray

Ex1: A = 1 2 3 4 −10

[1]
[1, 2]
[1,2,3]
[1,2,3,4]

Ans = 10

Ex4: −2 1 −3 4 −10 2 1 −5 4 => [4]

Ex2: −2 1 −3 4 −1 2 1 −5 4

[4, −1, 2, 1] => [4] => 4
[4, −1] => 3
[4, −1, 2] => 5
[4, −1, 2, 1] => 6

Ans = 6

Ex3: −7 −10 −3 −5

Ans = −3

Brute Force :

Consider all subarrays : $\frac{N(N+1)}{2}$.

+

Carry forward Approach

T.C : $O(N^2)$

```
maxSum = -INF;
for(i=0; i<N; i++){
    sum=0
    for(j=i; j<N; j++){
        // (i,j) represent subarray
        sum += A[j];
        maxSum = max(maxSum, sum)
    }
}
return maxSum
```

T.C : $O(N^2)$

S.C : $O(1)$

Approach2: Kadane's Algo

[1, 2, 3, 4, 5] =>

[1] => 1

[1, 2] => 3

[1, 2, 3] => 6

[1, 2, 3, 4] => 10

[1, 2, 3, 4, 5] => 15

Prefix

Lets assume A[i. . . . . j] is the max sum subarray.

Claim: No prefix of the subarray A[i....j] would have a negative sum

Proof:
Lets assume, there is a prefix of A[i....j] which has a negative sum

$$\text{sum}(A[i.....j]) = \text{sum}(\underbrace{A[i.....k]}_{\text{prefix}}) + \text{sum}(A[k+1....j])$$

$$\text{sum}(A[i....j]) = \boxed{-10} + \underbrace{\text{sum}(A[k+1....j])}$$

$$\underbrace{\text{sum}(A[k+1....j])} = \boxed{\text{sum}(\boxed{A[i.....j]})} + 10$$

max

A =	1	-2	1	-3	4	-1	2	1	-5	4 10
	0	1	2	3	4	5	6	7	8	9 10

sum = ~~0~~ ~~1~~ ~~-1~~ ~~0~~ ~~1~~ ~~-2~~ ~~0~~ ~~4~~ ~~3~~ ~~5~~ ~~6~~ ~~1~~ ~~8~~ 15

maxSum = ~~1~~ ~~4~~ ~~5~~ ~~6~~ 15

[ ]

[1, -2]

[4, -1]

A = 5 6 7 9 1 4

sum = ~~0~~ ~~5~~ ~~11~~ ~~18~~ ~~27~~ ~~28~~ 32

A = -7 -4 -2 -6 -5

sum = ~~0~~ ~~-7~~ ~~0~~ ~~-4~~ ~~0~~ ~~-2~~ ~~0~~ ~~-6~~ ~~0~~ -5

max Sum = ~~-INF~~ ~~-7~~ ~~-4~~ -2

```
maxSum = -INF;
 sum = 0;
for(i=0; i<N; i++){
        sum += A[i];
        maxSum = max(maxSum, sum)   (store index
                                      (start,end))
        if(sum < 0){
            sum = 0;
            start = i+1
        }
}
return maxSum;
```

T.C: O(N)

S.C: O(1)

+1 => 0

-1 => 1

[1 0 0 0 1] => sum = 2

[0 1 1 0 0 0 1 1 0 0]

swap

1 0 0 1 1 1 0 1 1

Jun22 Advanced

1/2 => [illegible] months