Q → You have a binary 2d array. You can perform one operations on the rows or the columns. Operation: flip the values i.e. make 0 → 1 or 1 → 0

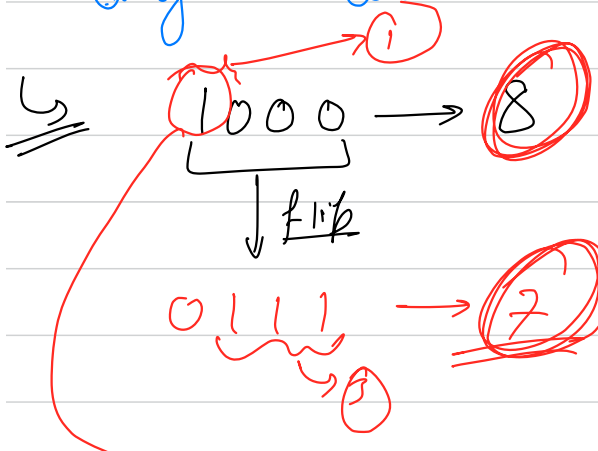
You can perform the abv operation multiple times. At the last, treat each row as a complete binary no. convert it to decimal & sum it up. find the max sum possible.

$1 \le n, m \le 60$

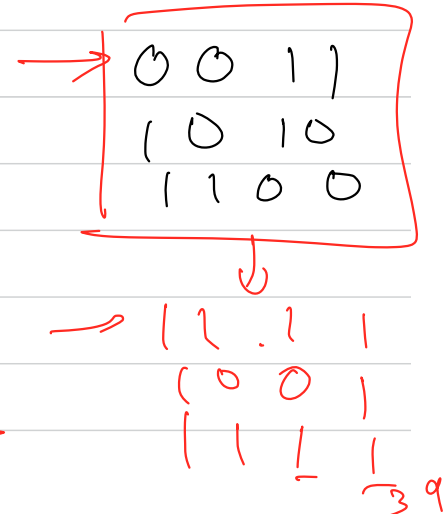
$$\begin{bmatrix} 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \end{bmatrix}_{1 \times m} \longrightarrow \begin{matrix} 1\ 1\ 1\ 1 \rightarrow 15 \\ 1\ 0\ 0\ 1 \rightarrow 9 \\ 1\ 1\ 1\ 1 \rightarrow 15 \end{matrix}$$

ans → 39

⇒ 01101 (only zeroes and 1s) → make each row to contribute as much as possible. We don't have any restriction on the no. of operations. (Brute force)



8+7=?



If we want to send our rows to glue max contribution

the MSB should be 1 → if the msb of any

row is not only → flip the row

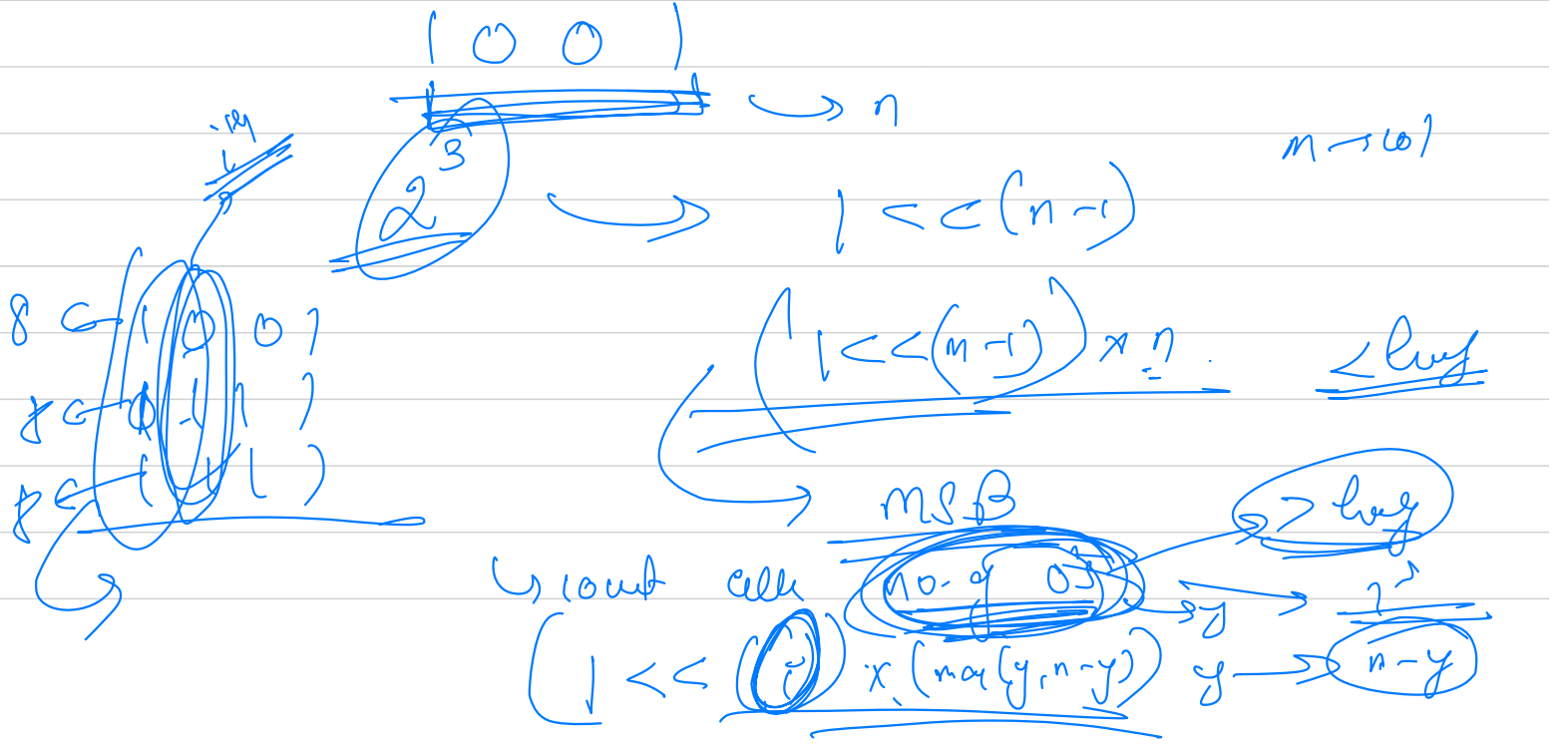
↳ from right to left the more ones we can get the

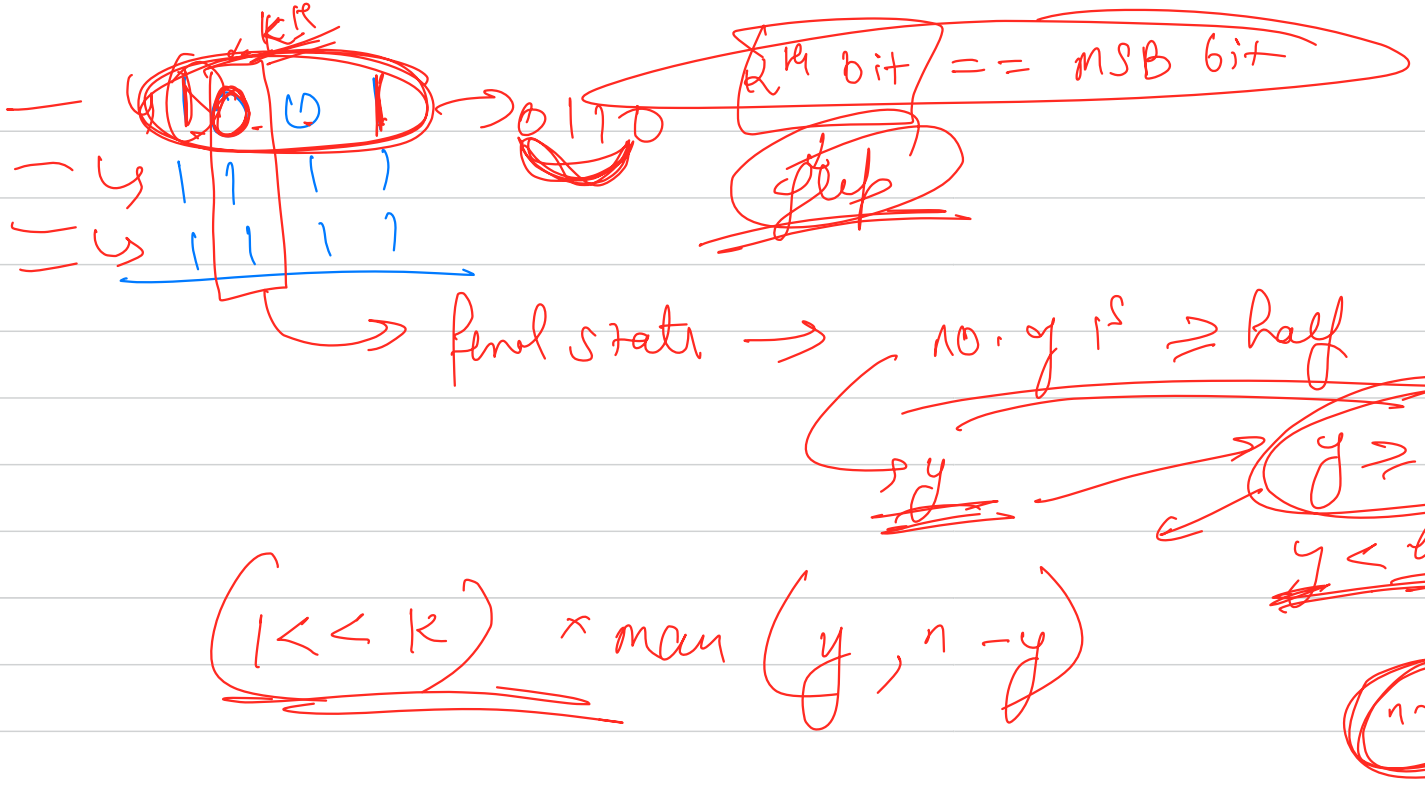
layer value we will encode.

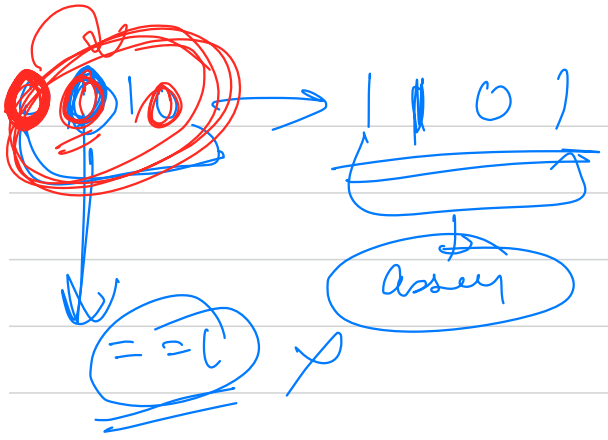
Row operation → should only change MSB to 1.

while flipping any column for internal bit, if we have more than half bits as 1, then don't flip.

Instead of actually flipping the bits, we can calculate them final contribution directly. (Bitmanipulation)




k^th
0 1 0
0 1 0
k^th bit == MSB bit
flip
final state →
10 · y is ≥ half
y
y ≥ half
y < half
(1 << k) × max (y, n - y)
n-y



no. of ans → y

no repeat → n - y

$\binom{y}{n}$ → many

```cpp
int maxSum(vector<vector<int> > &arr) {
    int sum = 0;
    int n = arr.size();// no of rows
    int m = arr[0].size();// no of columns

    sum += (1 << (m-1)) * n;
    for(int j = 1; j < m; j++) {
        int sameBit = 0;
        for(int i = 0; i < n; i++) {
            if(arr[i][j] == arr[i][0]) sameBit++;
        }
        sum += (1 << (m - j - 1)) * max(sameBit, n - sameBit);
    }
    return sum;
}
```

add contribution of MSB

j^{th} col → $(m-j-1)^{th}$ bit

KMGit- $m-j-1$

1 2

0 0 1 → 1100

1 0 1 0

1 1 0 0

$(m-1)$

2 →

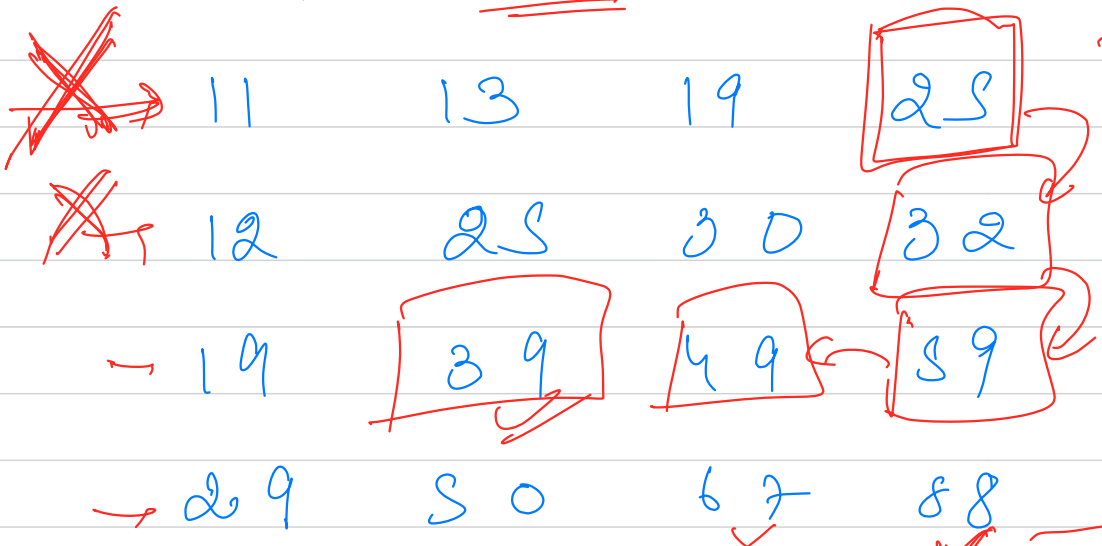
Sum = 0

n = 3

m = 4

$(1 << (m-1)) \times n$

Q) You have a row & column wise sorted matrix. Given an element x, find if it is present in the matrix or not.



O(nm)

Linear search for row

Binary search on col

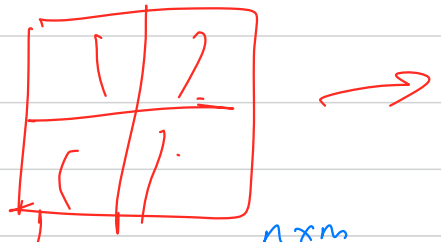
O(n x log m)

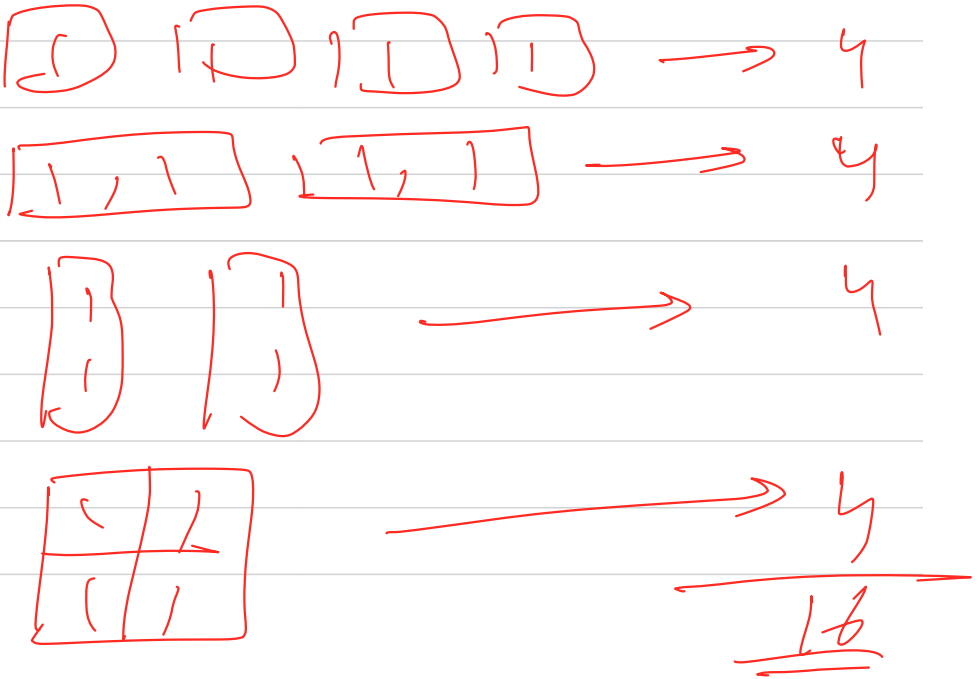
TC → O(n + m) → super slow

n ≤ 10^4
m ≤ 10^4

SC → O(1)

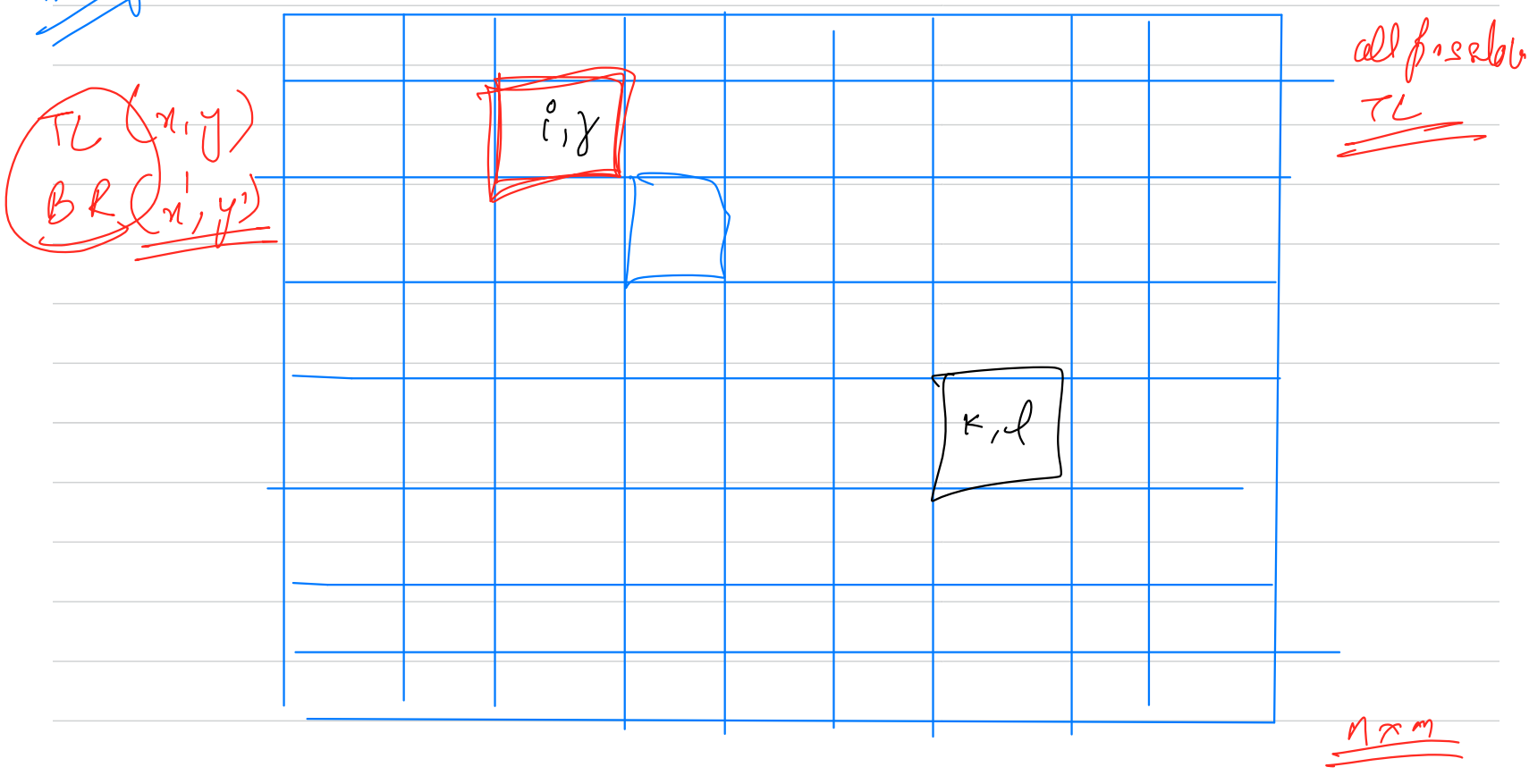
Q) You have 2d array of integers. Calculate the total sum of all sub matrices for the given matrix.





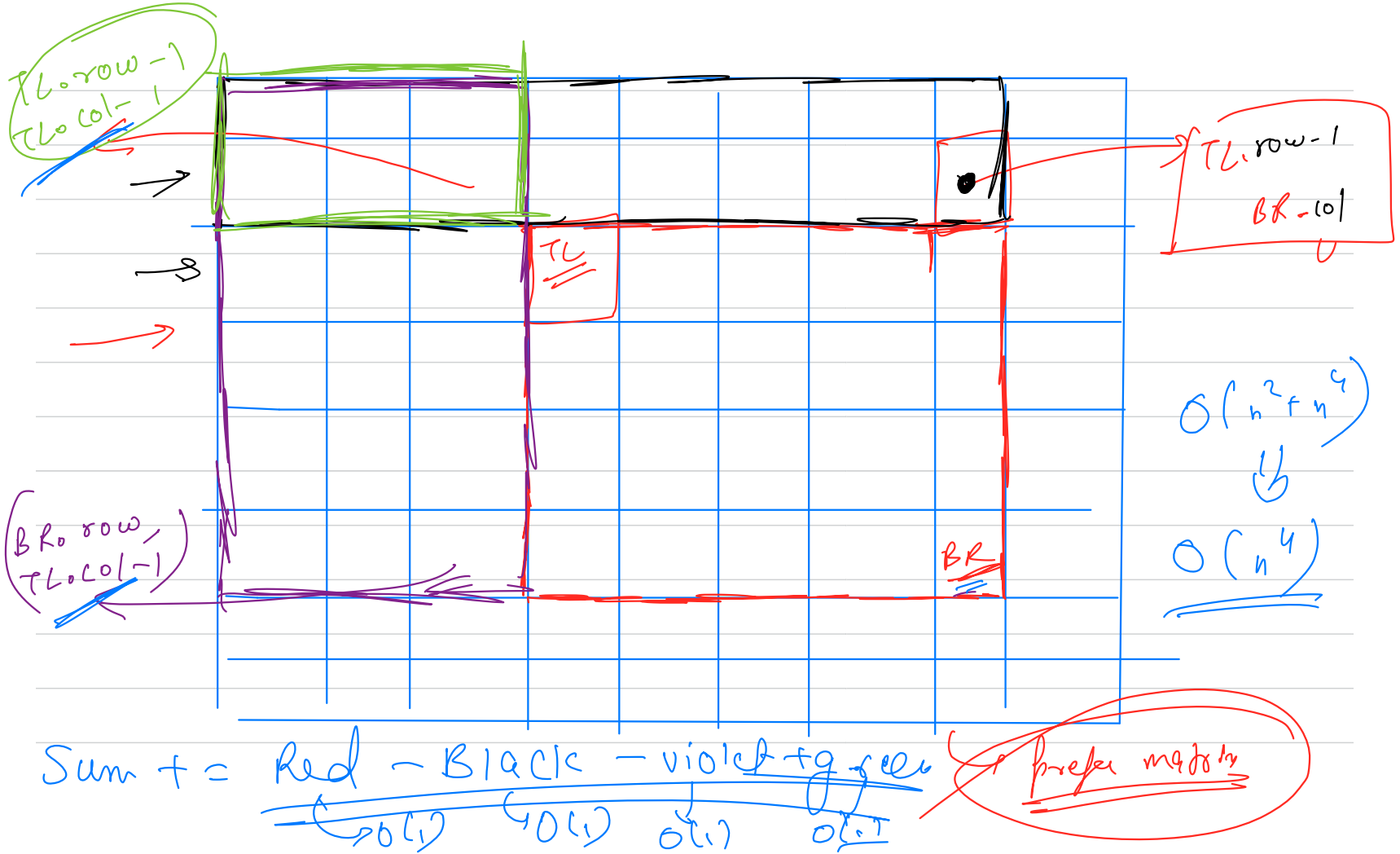
$n, m \leq 10^4$

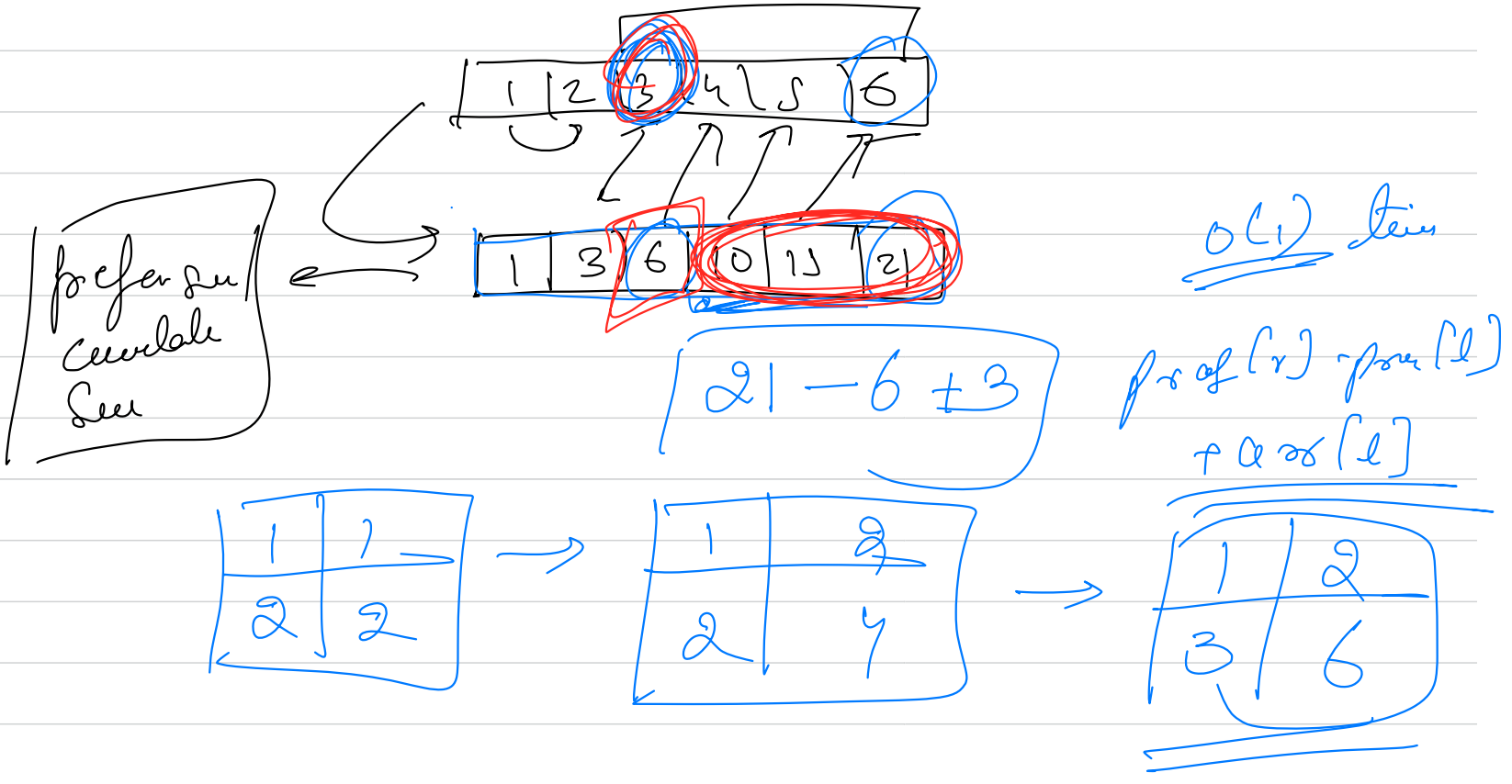
Brute force → take all submatrices → & store max of it

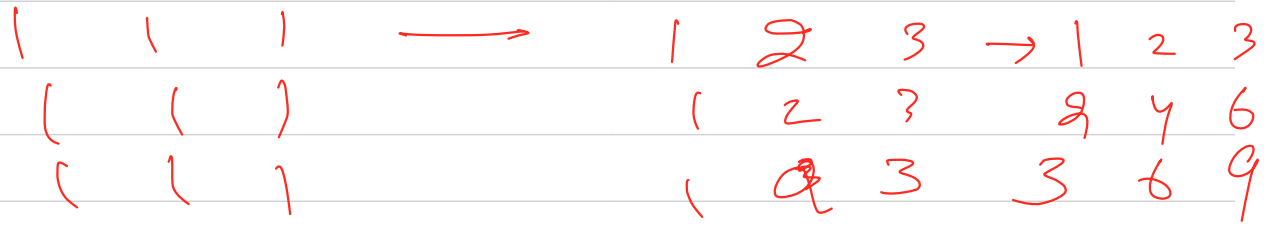


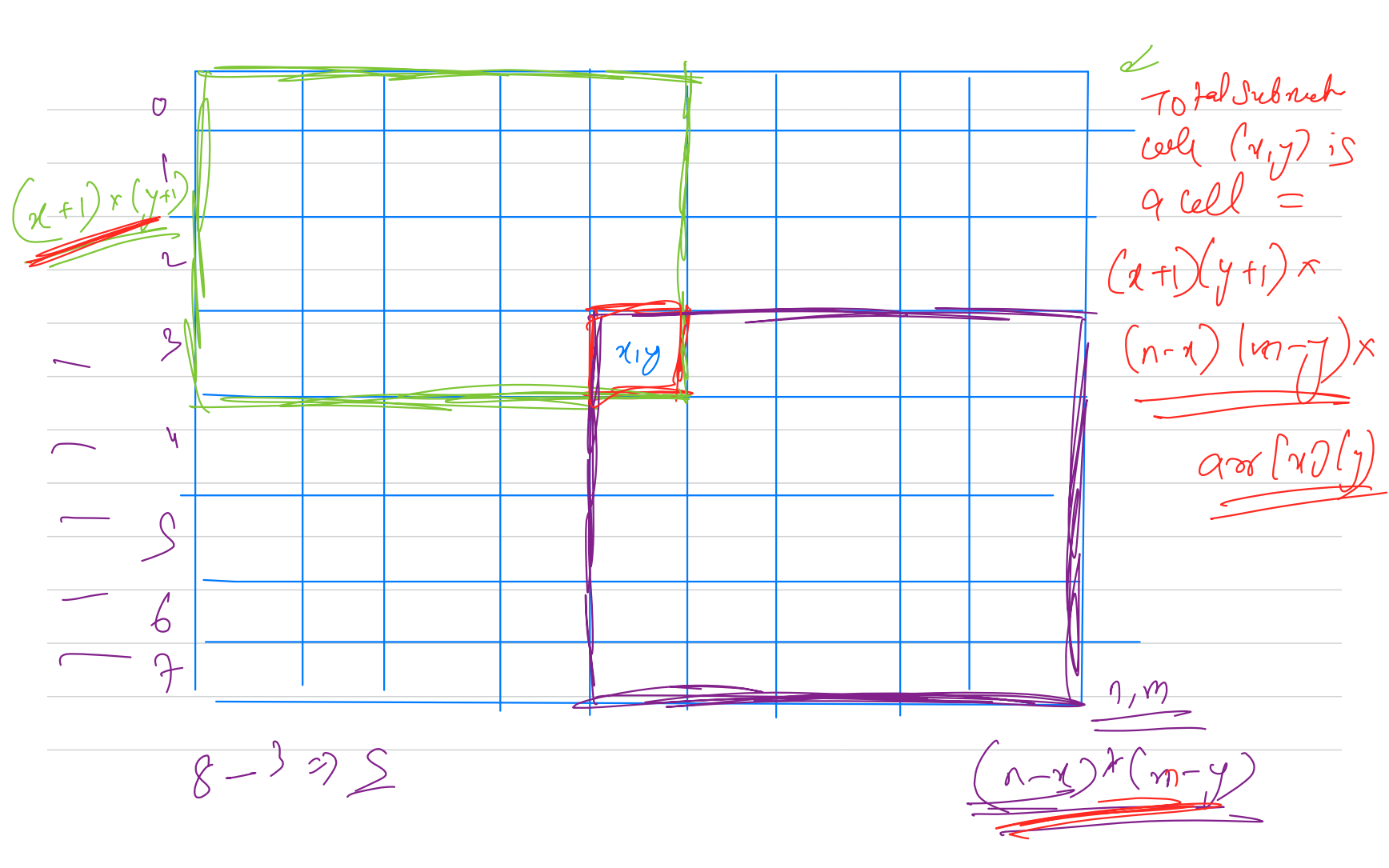
```
for(i=0; i<n; i++)
    for(j=0; j<m; j++) {
        (i, j) → TL
        for(k=i; k<n; k++) {
            for(l=j; l<m; l++) {
                (k, l) → BR
                for(x=i; x<=k; x++) {
                    for(y=j; y<=l; y++) {
                        sum += a[x][y]
                    }
                }
            }
        }
    }
```

$O(n^6)$


TL.row-1
TL.col-1
TL
TL.row-1
BR.col
BR.row,
TL.col-1
BR
$O(n^2 + n^4)$
$O(n^4)$
Sum += Red - Black - violet + green
O(1) O(1) O(1) O(1)
prefix matrix


1 2 3 4 5 6
1 3 6 10 15 21
preferences
O(1) time
21 − 6 + 3
pref[r] − pref[l] + arr[l]
1 1
2 2
1 2
2 4
1 2
3 6




0
1
2
3
4
5
6
7
(x+1) x (y+1)
x,y
n,m
(n-x)*(m-y)
8 -> 5
Total Subrect
cell (x,y) is
a cell =
(x+1)(y+1) x
(n-x)(m-y) x
arr[x][y]

2 reasons → You go to all submatrix →

→ You are calculate overlapping part of your & again.

→ Instead of calculaly overlapping cell again & again can we instead calculate the no. of submatrices the cell is a part of ??

(cache) → arr[x][y] × (count of submatrix)

O(1) → O(1) → O(1) → O(n³×n²)

Q: Given a 2D matrix of integers, and you have q queries, where in each query you get 4 integers x_1, y_1, x_2, y_2 → denoting a submatrix. For each query return the sum of submatrix.

$n, m \leq 10^4$

$q \leq 10^6$

→ prefix sum approach

$O(n*m)$ $q \to O(1)$

↳ $O(n*m + q)$