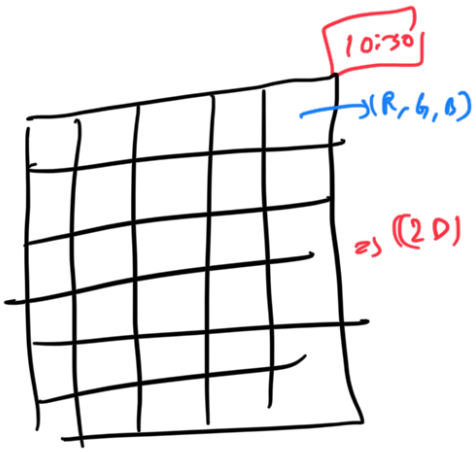
Multi - Dimensional Arrays



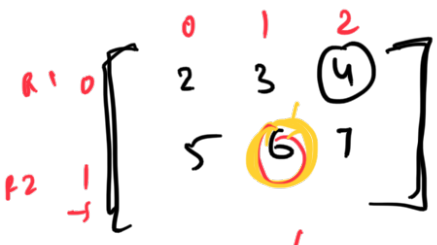
graphs => 2D Array
Adjacency List

Real life examples

1) Image
2) Chess Board
   ↓
   8 x 8
3) Sudoku (4x4)
4) Snake
5) Tic tac toe
6) Candy Crush

2D Arrays => Matrix

arr[n][m]
=> 2D Array with 'n' rows, 'm' columns



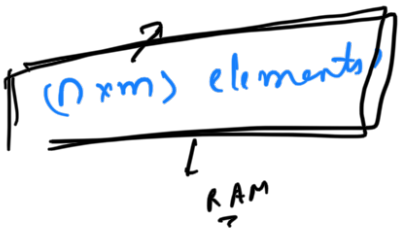
2 x 3

1D Array => arr[i]

6 => (1, 1)
4 => (0, 2)

(i, j) => 2D Index
2D Index ← Row No. Column No.

Integer - 4 bytes



n x m

Integer Array => (n x m) x 4 = 4.n.m

[ ][ ]nxm ... Array stored in Memory ?

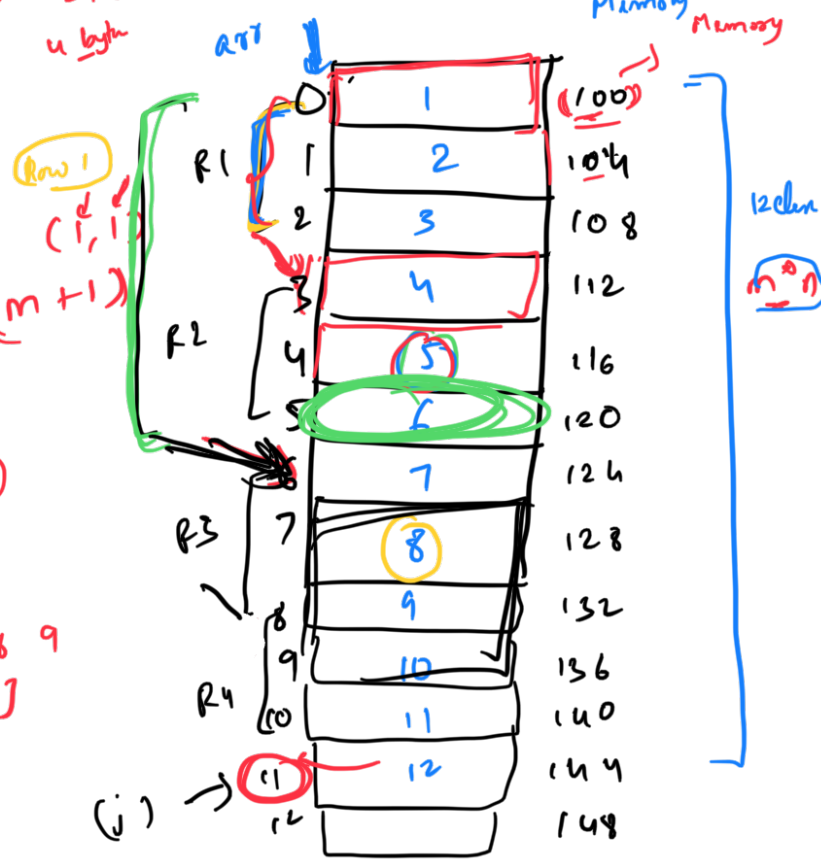
How is 2D Array stored in Memory (C++)

2 x 3 => (2 x m)

Row Major => store Row by Row

(i*m+j) (i*m+j-1)

(i*m+j) (i*m+j)



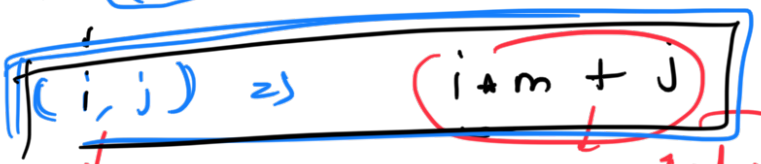
2D → 1D Matrix

[1 2 3 4 5 6 7 8 9 10 11 12]

Arr[] = [_ _ _ _ _ _] m*n

5 => arr[1][1]

8 => (2, 1) => (2*m) + 1



(i, j) => 1D

(3, 2) => 3 x 3 + 2 = 11

i j

2D (i, j) => (i * m + j) => (i*m+j) [ ]n x m

No. of column

n → Rows

m → Column

6 => (1, 2) => i * m + j

i j

(1 x m + j)

(1 x 3 + 2) = 5 => 1D Index

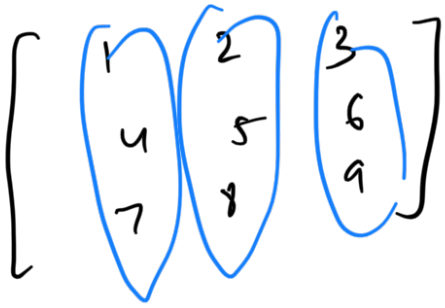
Addr(i, j) => base_address + ... (size)

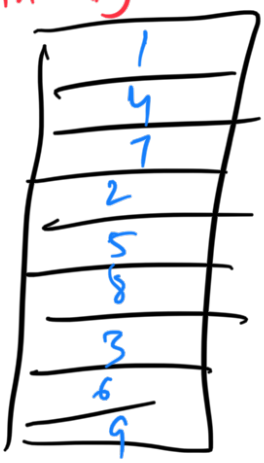
(i * m + j) * (size)

(INT: 4)

Column Major : Column by Column M

(Fortran, MatLab)



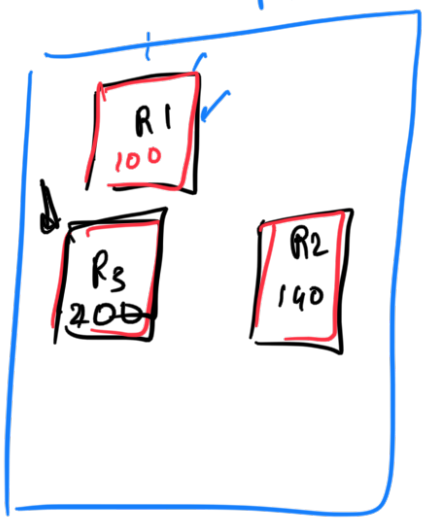


Array of Arrays :

(Java, JS, Python)

(Jagged Array)

Vector <Vector>



addr(R1) addr(R2) addr(R3)

arr = [100 , 140 , 200]

(1D Array) → Storing References of these 1D arrays in a different 1D Array

arr[2][1]

arr[i][j]

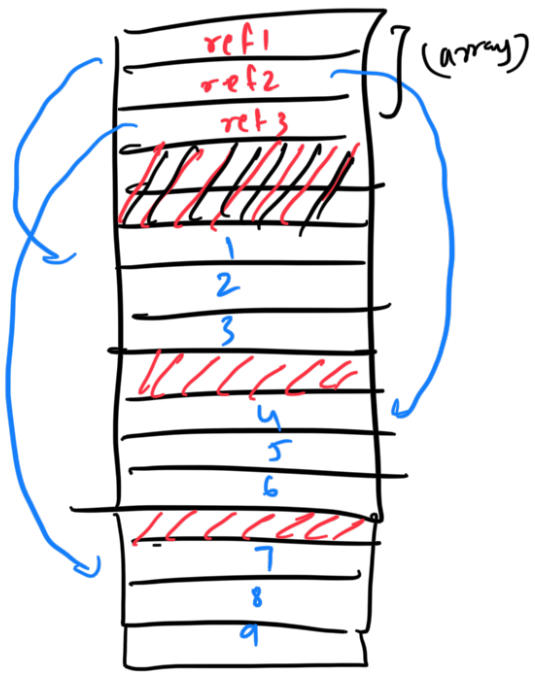
→ First look at 1D array of reference ⇒ O(1)

→ Using 'j' - find the element of O(1)

That 1D

=> All arrays are not in contiguous blocks

RAM



ref1 [1 2 3]
ref2 [4 5 6]
ref3 [7 8 9]
(3 Rows)
3x3

(9 elements)

1 D array 2) base address, size

{ ref1 [1 2 3 4 5 6]
ref2 [7 8]
ref3 [9 10 11] } (Jagged)

[ ]
int [][]

1 D array => Base Address + Size

(Dynamic Array)

Rows or Columns?

[ ]
[ ] . . . .

(i, j)

(2:30 30 mm)
mat[n][m]

Rows or Columns?

(Java)

(MLE)
(10^9)

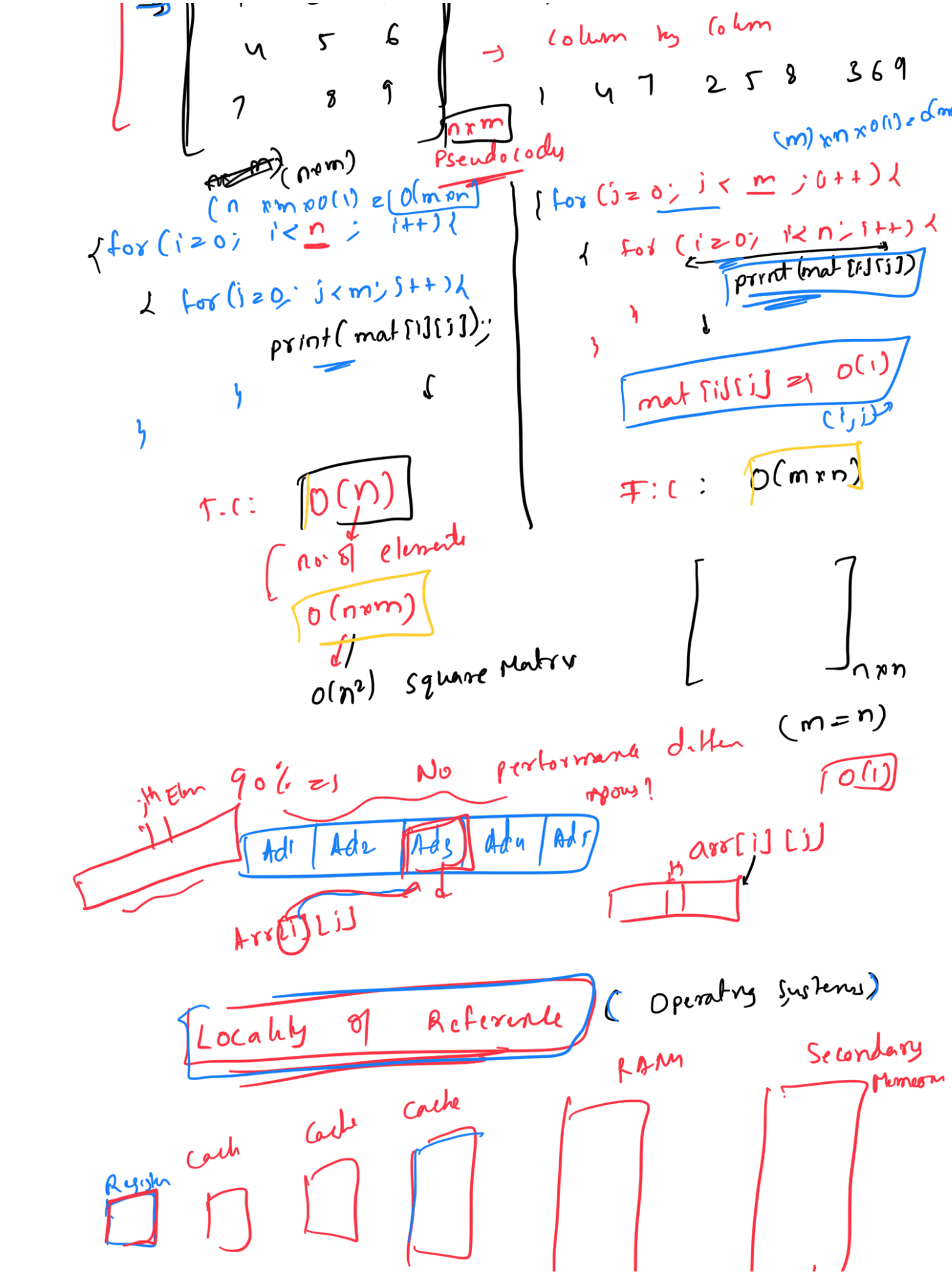
mat[][]

2) Iterating over 2D Matrix

→ Row by Row

[1 2 3]

1 2 3 4 5 6 7 8 9


4 5 6
7 8 9
nxm
Column by column
1 4 7 2 5 8 3 6 9
(n x m) x O(1) = O(mxn)
Pseudocode
for(i=0; i<n; i++){
for(j=0; j<m; j++){
print(mat[i][j]);
for(i=0; i<n; i++){
for(j=0; j<m; j++){
print(mat[i][j])
mat[i][j] => O(1)
(i,j)
T.C: O(n)
no. of elements
O(nxm)
O(n²) square matrix
S.C: O(mxn)
(m=n)
O(1)
jth Elm
90% =>
No performance difference now!
Adr1 Adr2 Adr3 Adr4 Adr5
Arr[i][j]
arr[i][j]
Locality of Reference
(Operating Systems)
Register
Cache
Cache
Cache
RAM
Secondary Memory

Column by Colum

Faster as cache

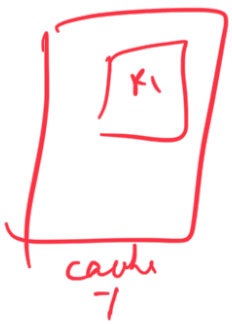
Least Acces

Row by Row as Faster Acces time.

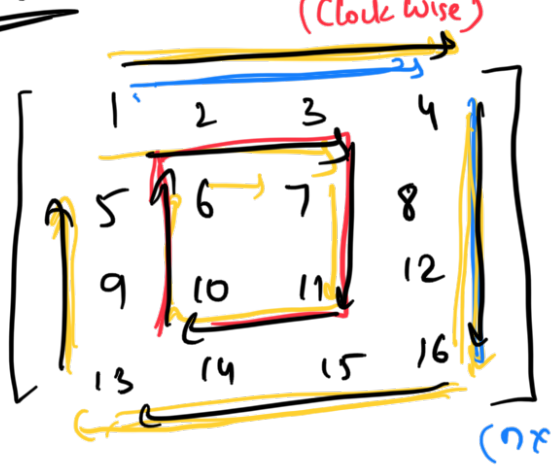
C++ :

Tagged Array

T.C: O(m×n)



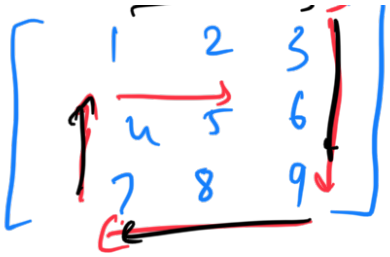
Question: Print Matrix in Spiral way



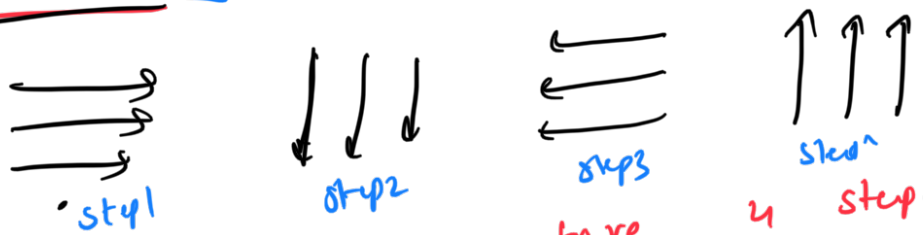
Ans: 1 2 3 4 8 12 16
15 14 13 9 5 6 7
11 10

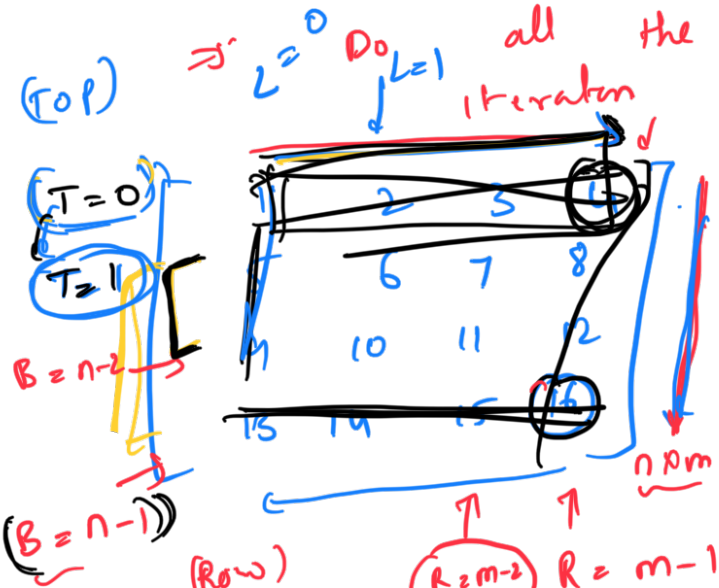
for(i=0; i<n; i++)
for(j=m-1; j>=0; j--

6 9 8 7 4 5



Writing code





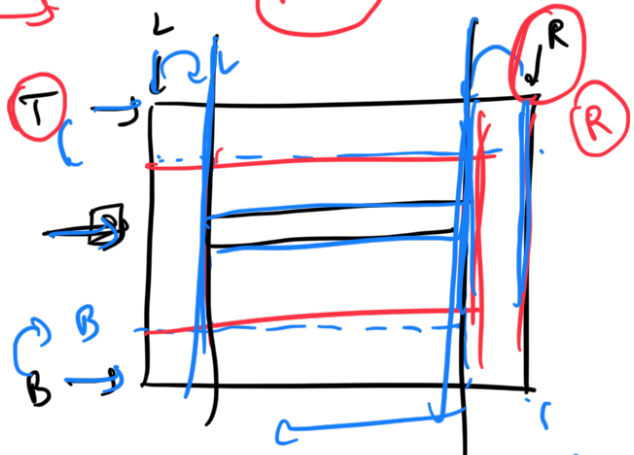
2^0 Do all the iteration

L=1

R = m-2

R = m-1

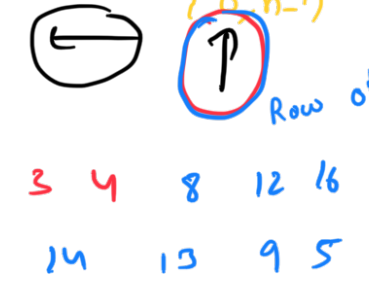
n rows



T = 0 , B = n-1 , L = 0 , R = m-1

above 4 steps in each m column (m-1)

T++;



Discarding $(n-1)^{th}$ row

$(n-2)^{th}$ row

n(1-1)

Collapsing the matrix

Peeling layers of onion

L ↓ , → R

T ↓ , ← B

T ≤ B

L ≤ R

while ((T ≤ B) && (L ≤ R))

T ≤ B || L ≤ B

L = 5

R = 3

T, B

Top

Bottom

(T ≤ B)

for (k = L ; k ≤ R ; k++) {

print (mat[T][k]);

}T++;

for (k = top ; k ≤ b ; k++) {

print(mat[K] [K]);

```
R--;
for (k = R; k >= L; k--)
    print(mat[B][k]);
B--;
for(k = B; k >= T; k--){
    print(mat[k][L]);
L++;
}
```

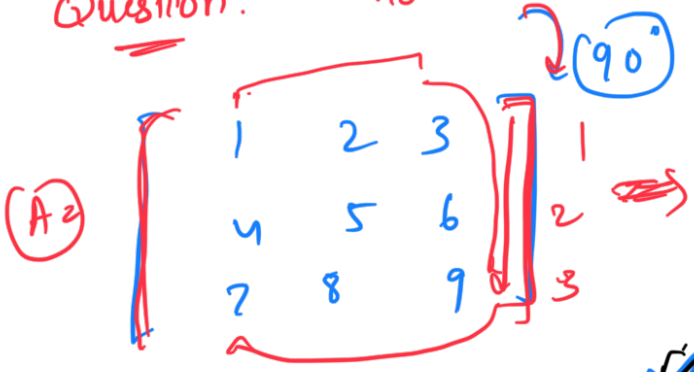
Way2: Count no. of printed elements (count)

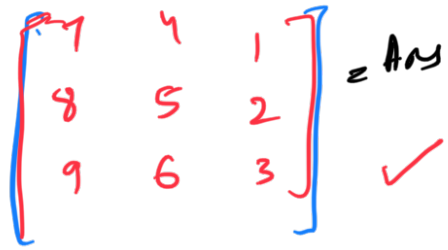
```
if (count == mn){
    break;
}
```

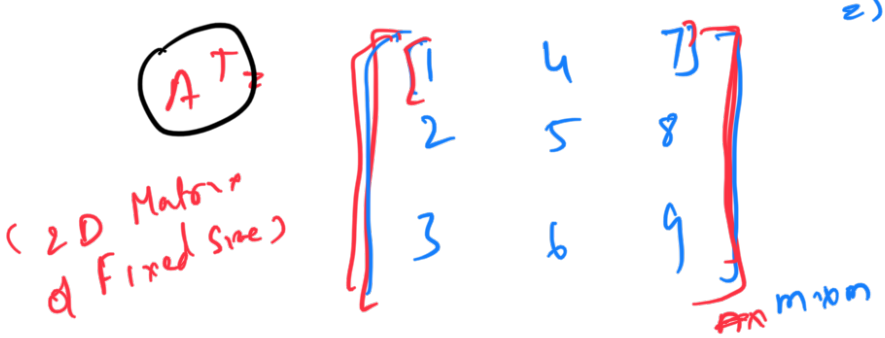
T.C: O(mn)

Question: Rotate a Matrix by (90°) (Amazon)

A2 = [ ]







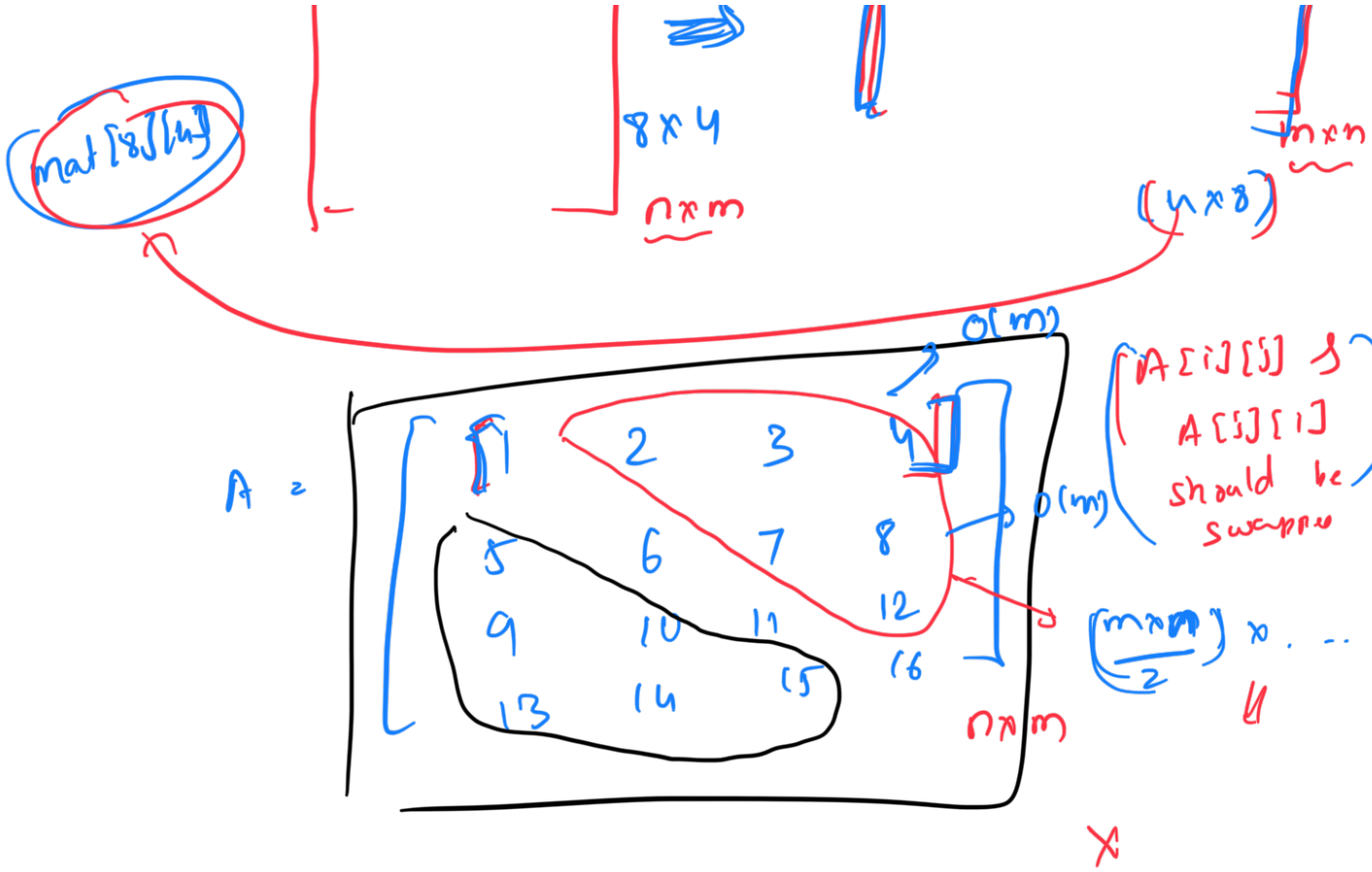
(2D Matrix of Fixed Size)

=) Transpose

→ Rows becomes Columns

→ Columns become Rows

Out of bound Error



Algo

1) Find Transpose ✓ O(mxn)
2) Reverse the row O(mxn)

Reversing n rows

O(m) x n = O(mxn)

rows

T.C: O(m.n)

[1 2 3 4 5 6]

Size = n

= O(n)

Code Complexity

O(n/2)

(n/2)

Question:

-> Given a matrix of 0's and 1's and if

-> All the rows are sorted!

-> with maximum

Task: Find the row with max no. of 1's [Increasing]

mat =

0	0	1	1	1
1	0	0	1	1
2	1	1	1	1
3	0	0	0	0

n x m

ans = 2

(ans = 2)

Approach 1:

For every row, count no. of 1's. Find the row with maximum 1's.

T.C: O(n x m)

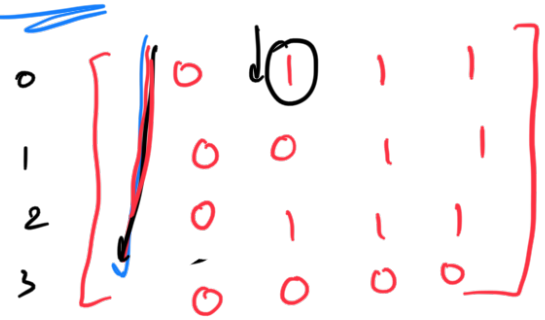
Approach 2:

For every row, find the sum. Find the row with maximum sum

T.C: O(m x n)

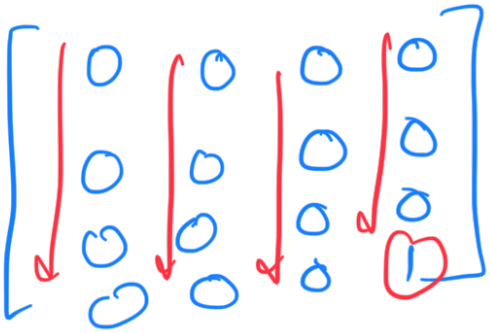
Approach 3:

Iterate Column by Column



Array is sorted

ans = 0



T.C: O(m x n)

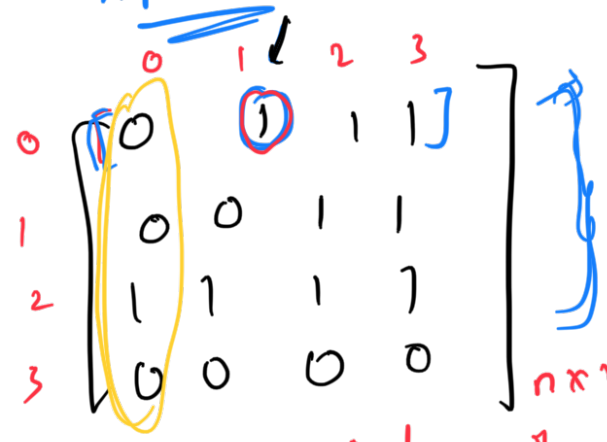
Best be O(1)

T.C: Worst Ca

B.S

Approach 4:

Binary Search



N = 4

(ind = 1)

((N - ind) = 3)

1) sorted
2) 0's & 1's

Find Index of first 1

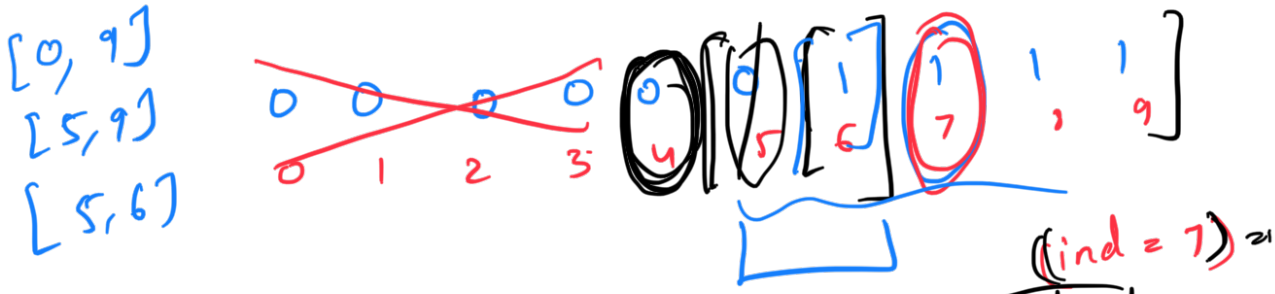
B.F :- T.C: Linear Search.

T.C. O(m)

(ans = 6)

(sorted)

→



(ind = 7) ⇒ ind = 6

$\left(\frac{0+9}{2}\right)$ ⇒

$\frac{n}{2} \rightarrow \frac{n}{4} \rightarrow \frac{n}{8} \cdots 1$

(logn)

int left;
int right

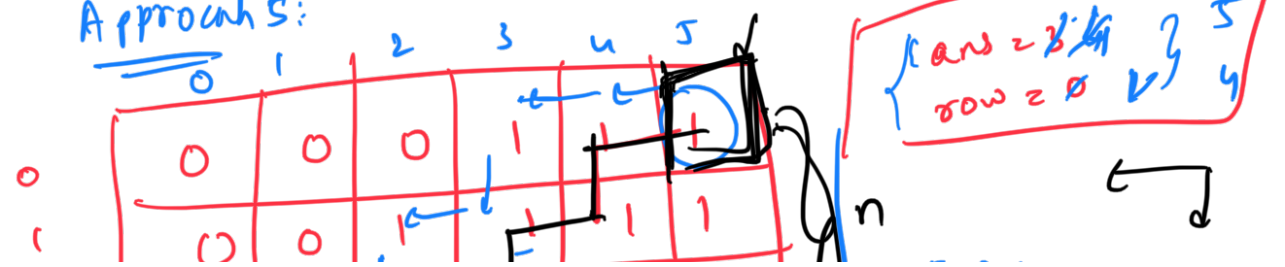
For 1 Row ⇒ (logm)

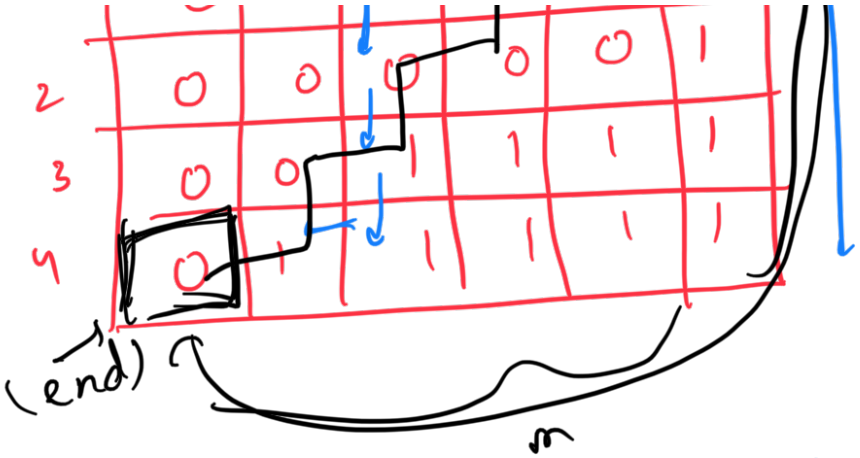
T.C: n x logm

⇒ O(nlogm)

Approach 5:

ans = 4





T.C :

1) Left
2) Bottom

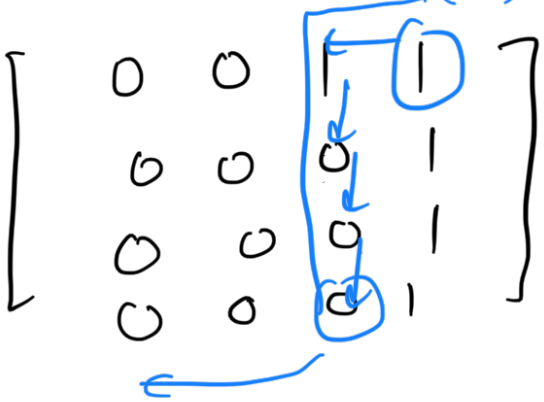
(m+n) steps

T.C : O(m+n)

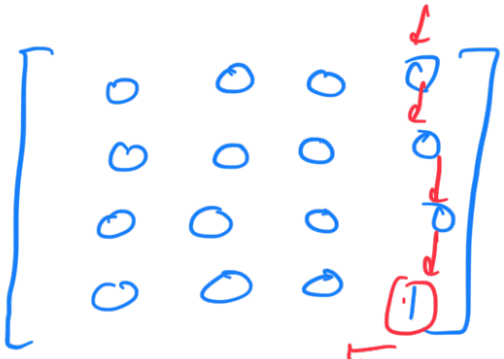
0^{th} row → n^{th} Row (n steps)

$(m-1)^{th}$ colm → 0^{th} colm (m steps)

m+n steps



(1+3) rowsh



O(m) operation

Question:

Search in a sorted Row-wise & column-wise Matrix.

T = 14 = True

T = 35 = False

mat

Brute force Approach

Iterate the whole matrix

T.C: O(n*m)

Approach 2: B.S on Rows

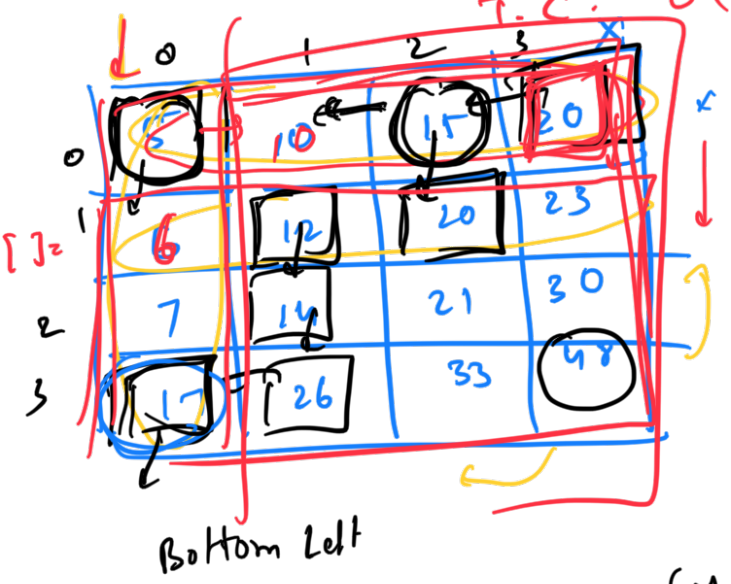
(logm)

T.C: n * logm

Approach:3 B.S on columns

logn → 1 Colum

T.C: O(m * logn)



(T = 17)

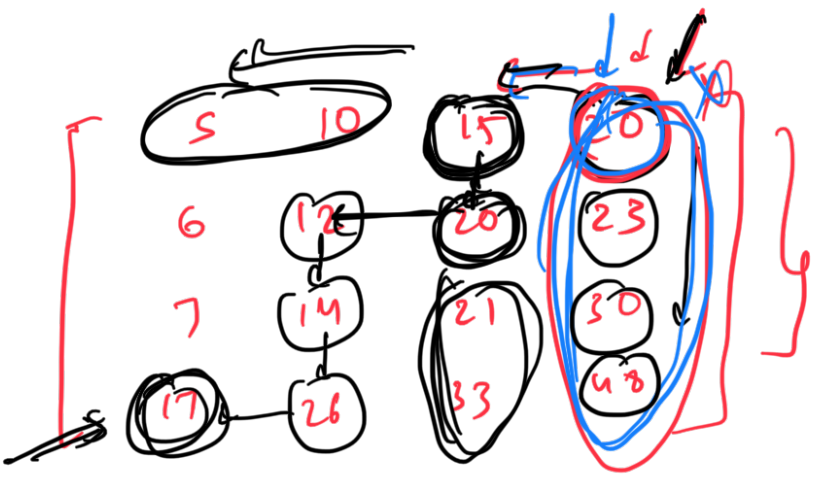
T.C: O(m+n)

Binary Search Tree

⇒ BST

(Min Heap / Priority_queue)

Colums as node

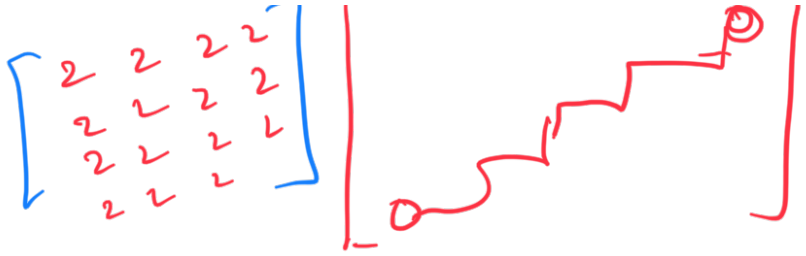


(T = 17)

n steps down

m steps Left

~~Left~~



Down
(m+n) steps.

(6 - 1 test case)

Corner case

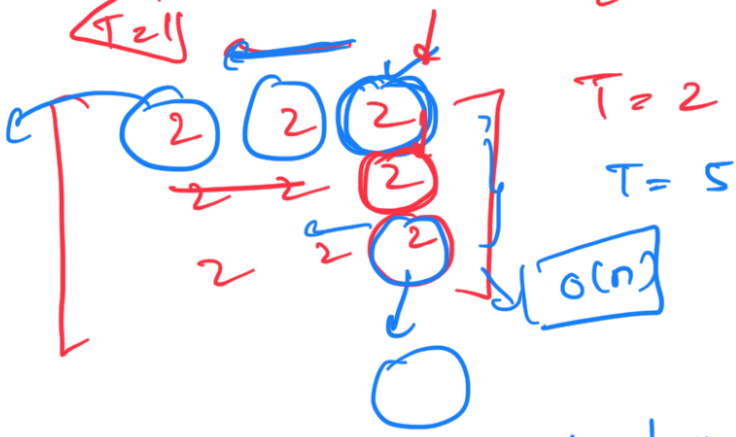
Arr. length()

O(1)

Input:

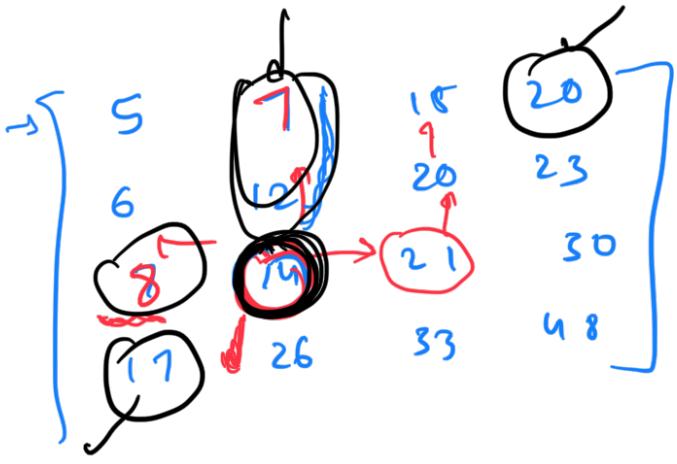
Output:

(meta data) => length
=> start Addr
End



Adv Math

T.



Cross the border.
& B

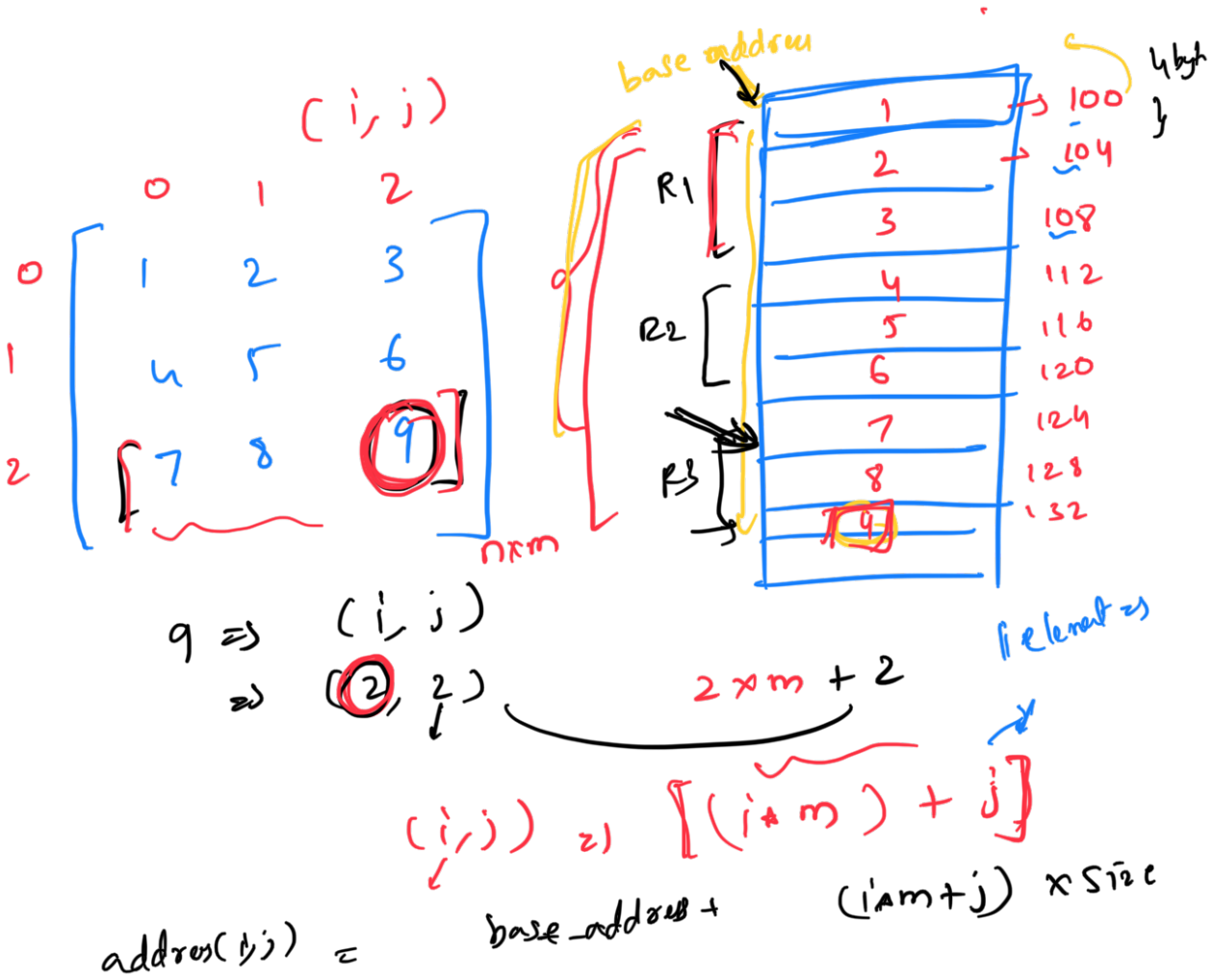
T = 15

T = 7

T = 7

T.C: O(mxn)

(Recursion)


(i, j)
0 1 2
0 1 2 3
1 4 5 6
2 7 8 9
n x m
base address
R1
R2
R3
1 100
2 104
3 108
4 112
5 116
6 120
7 124
8 128
9 132
4 byte
q => (i, j)
=> (2, 2)
2 x m + 2
i element
(i, j) => [(i * m) + j]
address(i,j) = base_address + (i*m+j) x size

Print Matrix in Spiral Way

```
printSpiral(int A[N)[M]{
      T = 0, B = N-1,  R = m-1, L = 0;

      while(L <= R && T <= B){
            for(int k = L; k <= R; k++)
                  print(A[T][k])
            T++;
            for(int k = T; k <= B; k++)
                  print(A[k][R]);
            R--;

            for(int k = R; k >= L; k--)
                  print(A[B][k])
            B--;

            for(int k = B; k >= T; k--)
                  print(A[k][L]);
            L++;
      }
}
```

Transpose of a Matrix:

```
void transpose(int A[N][M])
{
    for (int i = 0; i < N; i++)
        for (int j = i+1; j < M; j++)
            swap(A[i][j], A[j][i]);
}
```

Find a row with max 1s

```
int findRowWithMax1s(int arr[N][M]){
    int row = 0, i, j = M-1;
    for (i=0; i<N; i++){
        while (arr[i][j] == 1 && j >= 0){
            row = i;
            j--;
        }
    }
return row;
}
```

Search for target in Sorted Matrix

```
bool searchInMatrix(int arr[n][m], int target){
      row = 0, col = m-1;

      while(row < n && col >= 0){
            if(a[row][col] == target)
                  return true;
            else if(a[row][col] < T){
                  row++;
            }
            else
                  col--;
      }
      // If element is not found, return false
      return false;
}
```