DSA

22/8/2025

classmate
Date
Page 50

2D Matrices & Basic Sorting Algorithms

- **Introduction -**
- A 2D matrix is a specific type of 2D array that has a rectangular grid of numbers, where each number is called an element.
- It is a mathematical structure that consists of a set of numbers arranged in rows & columns
- 2D Matrix can be declared as:

```
mat [N] [M]
 ↙      ↑     ↑
name  no.of  no.of
      rows   columns
```

mat [i] [j] → i^{th} row, j^{th} column

	0	1	2 ← col. index
0	(0,0)	(0,1)	(0,2)
1	(1,0)	(1,1)	(1,2)
2	(2,0)	(2,1)	(2,2) → (N-1, M-1) index

N → N-1
M → M-1

↑ Row index

NXM
↑ rows ↑ columns

Total no.of elements = N * M

	0	.....	M-1
0	(0,0)		(0,M-1)
⋮			
N-1	(N-1,0)		(N-1,M-1)

NXM

- **Print row-wise sum :**

Given a 2D matrix mat[N][M], print the row-wise sum.

```
mat[3][4] = { {1,2,3,4},
              {5,6,7,8},
              {9,10,11,12} }
```

o/p: 10
26
42

Approach : - Traverse each row
- While traversing take sum of all elements present in that row.

```
def sumRow (mat, N, M):
    for i in range (0,N): #iterate over all rows
        sum = 0
        for j in range (0,M):
            sum += mat[i][j]
        print (sum)
```

#iterate over that row and get sum

TC : O(N*M)
SC : O(1)

- **Print column wise sum:**

```
def matColRow(mat, N, M):
    for i in range(0, M):
        sum = 0
        for j in range(0, N):
            sum += mat[j][i]

        print(sum)
```

TC: O(N*M)
SC: O(1)

- **Print diagonals of a square matrix:**

Given a 2D square matrix, print diagonals.
(cols = rows)

Q) How many main diagonals are there in a square matrix?

→ 2

1. Principal diagonal: From top left to bottom right
2. Anti-diagonal: From top right to bottom right. left

	0	1	2
0	1	5	8
1	4	3	1
2	6	5	2

Principal diagonal
o/p: 1 3 2

	0	1	2
0	1	5	8
1	4	3	1
2	6	2	2

Anti diagonal
o/p: 8 3 6

Principal Diagonal

```
def printDiagonal(mat, N):
    i = 0
    while i < N:
        print(mat[i][i])
        i += 1
```

TC : O(N)
SC : O(1)

Anti-diagonal

```
def printDiagonal(mat, N):
    i = 0
    j = N-1
    while i < N:
        print(mat[i][j])
        i += 1
        j -= 1
```

TC : O(N)
SC : O(1)

✡ Introduction to Sorting:

Sorting is arranging data in a particular manner.

eg: 1, 7, 2, 9, 24

ele →	factor
1 →	1
7 →	2
2 →	2
9 →	3
24 →	8

∴ 1, 7, 2, 9, 24 → Yes, increasing order of factors.

So, arranging data in increasing or decreasing order on the basis of any parameter is known as sorting.

Why is sorting important?

Helps in:
- Organized data
- Analysing data
- Searching
- Presenting

li.sort()
↓
default → Inc. order of value.

☆ Minimize the cost to empty array:

Given an array of n integers, minimize the cost of to empty given array where cost of removing an element is equal to sum of all elements left in an array.

Note: Add cost of removed element & remove it.

Way 1: { 2 1 4 }

Remove 2: 2+1+4 = 7
1: 1+4 = 5
4: 4 = 4

Cost of removal = 7+5+4 = (16)

Way 2: { 2 1 4 }

Remove 4: 4+2+1 = 7
2: 2+1 = 3
1: 1 = 1

Cost of removal = 7+3+1 = (11)

☆ Observation:

arr[4] = { a b c d }

Remove a: a+b+c+d
Remove b: b+c+d
Remove c: c+d
Remove d: d

cost = a+2b+3c+4d

arr[0]*1 + arr[1]*2 + arr[2]*3 + arr[3]*4

In general: arr[i]*(i+1)

✱ Pseudocode :

```
def costmin(arr):
    arr.sort(reverse = True)
    total_cost = 0

    for i in range(0, len(arr)):
        total cost += arr[i] * (i+1)

    return total_cost
```

TC : #sorting + #operation
nlogn + n
⇒ O(nlogn)

SC : O(n) #sort function

0	1	2
1	6	3
→ 6	3	1

total cost = 0

i = 0 → 6 * 1
+
i = 1 → 3 * 2
+
i = 2 → 1 * 3

15

☆ Selection Sort →

arr = [5 6 4 2]

Pass 1 : (5) 6 4 (2)

Pass 2 : 2 (6) (4) 5

Pass 3 : 2 4 (6) (5)

2 4 5 6

☆ Pseudocode –

```
def selectionsort (arr, size):
        i, j, minder                          → (size-2)
#no. of passes   for i in range (0, size-1):
                    minder = i
#compare & find smallest   for j in range (i+1, size):
                               if (arr[j] < arr[minder]):
                                   minder = j
    # swap the smallest ele with the first ele.
                    temp = arr [minder]
                arr [minder] = arr[i]
                    arr[i] = temp
```

TC : O (N*N)
SC : O (1)

☆ Insertion Sorting :

Given an array[N], and first N-1 elements are sorted sort the entire array.

arr [6] : [10, 3, 6, 8, 2, 5]

1. Assume that elements from 0 to i is sorted.
2. We have to insert (i+1) element at its correct position in sorted array using insertion step.

```
          0  1  2  3  4  5
arr [6] : [10, 3, 6, 8, 2, 5]
```

Step 1 : 0 to 0 is sorted, place 1 correctly.

10 3 6 8 2 5

sorted ~~not sorted~~

Step 2 : 0 to 1 is sorted, place 2 correctly.

3 10 6 8 2 5

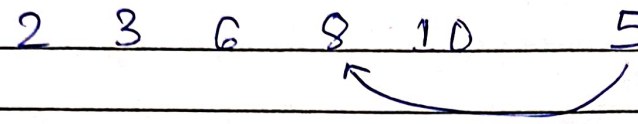
Step 3 : 0 to 2 is sorted, place 3 correctly.

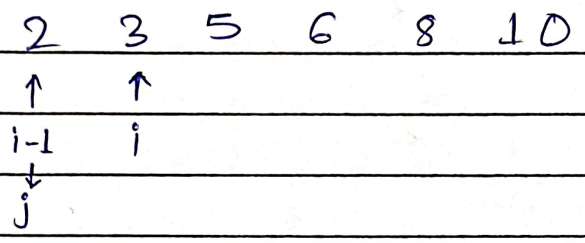
3 6 10 8 2 5

Step 4 : 0 to 3 is sorted, place 4 correctly.

3 6 8 10 2 5

Step 5: 0 to 4 is sorted, place 5 correctly.





★ Pseudocode -

```
def insertionsort (arr, n):
    for i in range (1, n):
        curele = arr [i]
        j = i-1

        while j>=0 and arr[j] > curele:
            a[j+1] = arr[j]
            j = j-1

        arr [j+1] = curele
```

j ↓ 3 10 [10, 3, 6, 8, 2, 5] ↑ i	j ↓ 6 10 [3, 10, 6, 8, 2, 5]	j ↓ [3, 6, 10, 8, 2, 5]
i = 1	i = 2	i = 3
curele = 3	curele = 6	curele = 8
j = 0 - 1	j = 1 → 0	j = 2 → 1
arr [0] = 3	arr [1] = 6	arr [2] = 8

TC: O(N²) SC: O(1)
(Inplane Algorithm)