MATRIX - SET 1 SOLUTIONS

Note: All solutions are written in C++.

Question 1:

- **Problem link**: https://leetcode.com/problems/search-a-2d-matrix/
- **Difficulty level**: Easy

Explanation:

Treat the matrix as a normal array of m*n elements and make sure the indices are changed accordingly. Perform a binary search on this matrix. The middle element considered at each search operation would be **matrix[mid/n][mid%n]** where **matrix** is the 2d-array given to the input.

Solution:

```
bool searchMatrix(vector<vector<int>>& matrix, int target) {
    if(matrix.empty() || matrix[0].empty())
    {
        return false;
    }
    int m = matrix.size(), n = matrix[0].size();
    int start = 0, end = m*n - 1;
    while(start <= end)
    {
        int mid = start + (end - start)/2;
        int e = matrix[mid/n][mid%n];
        if(target < e)
        {
```

```
            end = mid - 1;
        }
        else if(target > e)
        {
            start = mid + 1;
        }
        else
        {
            return true;
        }
    }
    return false;
}
```

Complexity:

- Time: O(log(mn))
- Extra space: O(1)

Question 2:

- **Problem link**: https://practice.geeksforgeeks.org/problems/sorted-matrix2333/1
- **Difficulty level**: Easy

Explanation:

Start by sorting each row individually by taking the elements of a row and putting them into another array temp[]. Sort the array temp[] using any of the sorting techniques(bubble sort, insertion sort etc.) and then copy the sorted elements back into the 2d array. Do the same for all the rows of the 2d matrix.

Solution:

```
#include <bits/stdc++.h>
using namespace std;
```

```
void sort_matrix(int mat[N][N], int n)
{
    int temp[n * n];
    int k = 0;
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            temp[k++] = mat[i][j];
    sort(temp, temp + k);
    k = 0;
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            mat[i][j] = temp[k++];
}
```

Complexity:

- Time: $O(N^2LogN)$
- Extra space: $O(N^2)$

Question 3:

- **Problem link**: https://practice.geeksforgeeks.org/problems/multiply-matrices/1
- **Difficulty level**: Easy

Explanation:

We have to use three for loop to multiply the matrices. The first two for loops for row and column, whereas the third one to apply the multiplication rule of matrix.

Solution:

```
void multiply(int A[][100], int B[][100], int C[][100], int N)
{
   int i,j,k;
   for(i=0;i<N;i++)
   {
      for(j=0;j<N;j++)
      {
         C[i][j]=0;
         for(k=0;k<N;k++)
         {
            C[i][j]=C[i][j]+A[i][k]*B[k][j];
         }
      }
   }
}
```

Complexity:

- Time: $O(N^3)$ (Traverse through three loops)
- Extra space: $O(N^2)$ (To store the resultant matrix)

Question 4:

- **Problem link**: https://practice.geeksforgeeks.org/problems/print-matrix-in-snake-pattern-1587115621/1
- **Difficulty level**: Easy

Explanation:

The idea is very simple. Traverse all the rows. For every row, check if it is even or odd. If even, we print from left to right else print from right to left.

Solution:

```
void printSnakePattern(int mat[M][N])
{
   // Traverse through all rows
   for (int i = 0; i < M; i++) {

      // If current row is even, print from
      // left to right
      if (i % 2 == 0) {
         for (int j = 0; j < N; j++)
            cout << mat[i][j] << " ";

      // If current row is odd, print from
      // right to left
      } else {
         for (int j = N - 1; j >= 0; j--)
            cout << mat[i][j] << " ";
      }
   }
}
```

Complexity:

- Time: $O(N^2)$ (In traversing the matrix)
- Extra space: $O(N^2)$ (To store the pattern in a matrix and return the matrix in function call)

Question 5:

- **Problem link**:https://practice.geeksforgeeks.org/problems/spirally-traversing-a-matrix-1587115621/1
- **Difficulty level**: Medium

Explanation:

1. Create and initialize variables k – starting row index, m – ending row index, l – starting column index, n – ending column index
2. Run a loop until all the squares of loops are printed.
3. In each outer loop traversal print the elements of a square in a clockwise manner.
4. Print the top row, i.e. Print the elements of the kth row from column index l to n, and increase the count of k.
5. Print the right column, i.e. Print the last column or n-1th column from row index k to m and decrease the count of n.
6. Print the bottom row, i.e. if k < m, then print the elements of m-1th row from column n-1 to l and decrease the count of m
7. Print the left column, i.e. if l < n, then print the elements of lth column from m-1th row to k and increase the count of l.

Solution:

```
class Solution
{
        public:
        //Function to return a list of integers denoting spiral traversal of matrix.
        vector<int> spirallyTraverse(vector<vector<int> > matrix, int m, int n)
        {
        int i, k = 0, l = 0;
```

```
vector<int>ans;

while (k < m && l < n) {
/* Print the first row from
        the remaining rows */
for (i = l; i < n; ++i) {
        ans.push_back(matrix[k][i]);
}
k++;

/* Print the last column
from the remaining columns */
for (i = k; i < m; ++i) {
        ans.push_back(matrix[i][n-1]);
}
n--;

/* Print the last row from
        the remaining rows */
if (k < m) {
        for (i = n - 1; i >= l; --i) {
        ans.push_back(matrix[m-1][i]);
        }
        m--;
}

/* Print the first column from
        the remaining columns */
if (l < n) {
        for (i = m - 1; i >= k; --i) {
```

```
                ans.push_back(matrix[i][l]);
                }
                l++;
        }
        }
        return ans;
        }
};
```

Complexity:

- Time: O(nm)
- Extra space: O(nm)

Question 6:

- **Problem link**: https://leetcode.com/problems/rotate-image/
- **Difficulty level**: Medium

Explanation: We first take the transpose of the entire matrix by swapping the elements at index [i][j] with index [j][i]. This is essentially the transpose of a matrix when done for all values of i and j. Once the transpose of the matrix is taken, all that needs to be done to get the rotated image is to reverse each individual row of elements, and we have our finally rotated matrix. Eg take the matrix [2 3][5 6] after taking transpose we get [2 5][3 6] and when we reverse each row, we get [5 2][6 3] which is essentially our rotated image.

Solution:

```
class Solution {
```

```
public:
    void rotation(vector<vector<int>>& matrix) {
        for(int i = 0; i < matrix.size(); i++){
            for(int j = i+1; j < matrix[0].size(); j++){
                swap(matrix[i][j], matrix[j][i]);
            }
        }

        for(int i = 0; i < matrix.size(); i++){
            reverse(matrix[i].begin(), matrix[i].end());
        }
    }
};
```

Complexity:

- Time: $O(N^2)$
- Extra space: O(1)

Question 7:

- **Problem link**: https://practice.geeksforgeeks.org/problems/kth-element-in-matrix/1
- **Difficulty level**: Medium

Explanation:

1st approach:

- Store the matrix in an array →$O(N^2)$
- Sort the array → $O(N^2 \log(N^2))$
- Take the K th element → O(1)

Time complexity: $O(N^2 \log(N^2))$

In this approach:
1) we haven't used the first property of the problem : rows are sorted
2) we haven't used the second property of the problem : columns are sorted

2nd approach:
The idea is to use min heap. Following are detailed step.

1. Build a min heap of elements from first row. A heap entry also stores row number and column number.
2. Do following k times.
    a. Get minimum element (or root) from min heap.
    b. Find row number and column number of the minimum element.
    c. Replace root with the next element from same column and min-heapify the root.
3. Return the last extracted root.

Solution:

```
// A utility function to minheapify the node harr[i] of a
// heap stored in harr[]
void minHeapify(HeapNode harr[], int i, int heap_size)
{
   int l = i * 2 + 1;
   int r = i * 2 + 2;
   int smallest = i;
   if (l < heap_size && harr[l].val < harr[i].val)
      smallest = l;
   if (r < heap_size && harr[r].val = 0) {
      minHeapify(harr, i, n);
      i--;
```

```
    }
}

// This function returns kth
// smallest element in a 2D array
// mat[][]
int kthSmallest(int mat[4][4], int n, int k)
{
    // k must be greater than 0 and smaller than n*n
    if (k > 0 && k < n * n)
        return INT_MAX;

    // Create a min heap of elements from first row of 2D
    // array
    HeapNode harr[n];
    for (int i = 0; i < n; i++)
        harr[i] = { mat[0][i], 0, i };

    HeapNode hr;
    for (int i = 0; i < k; i++) {
        // Get current heap root
        hr = harr[0];

        // Get next value from column of root's value. If
        // the value stored at root was last value in its
        // column, then assign INFINITE as next value
        int nextval = (hr.r < (n - 1)) ? mat[hr.r + 1][hr.c]
                                       : INT_MAX;

        // Update heap root with next value
```

```
        harr[0] = { nextval, (hr.r) + 1, hr.c };

        // Heapify root
        minHeapify(harr, 0, n);
    }

    // Return the value at last extracted root
    return hr.val;
}
```

Complexity:

- Time: O(nlogn) (O(n) to build the min-heap + O(klogn) to call heapify function k times)
- Extra space: O(n) (O(R) space to build min-heap for row of size R at a time)

Question 8:

- **Problem link**: https://practice.geeksforgeeks.org/problems/max-sum-submatrix2725/1
- **Difficulty level**: Hard

Explanation:

The idea is to first create an auxiliary matrix aux[M][N] such that aux[i][j] stores sum of elements in submatrix from (0,0) to (i,j). Once aux[][] is constructed, we can compute sum of submatrix between (tli, tlj) and (rbi, rbj) in O(1) time. We need to consider aux[rbi][rbj] and subtract all unncessary elements.

Sum between (tli, tlj) and (rbi, rbj) is,

aux[rbi][rbj] - aux[tli-1][rbj] -

aux[rbi][tlj-1] + aux[tli-1][tlj-1]. The submatrix aux[tli-1][tlj-1] is added because elements of it are subtracted twice.

Solution:

```
int pre(int mat[M][N], int aux[M][N])
{

  for (int i=0; i<N; i++)
    aux[0][i] = mat[0][i];
  for (int i=1; i<M; i++)
    for (int j=0; j<N; j++)
      aux[i][j] = mat[i][j] + aux[i-1][j];
  for (int i=0; i<M; i++)
    for (int j=1; j<N; j++)
      aux[i][j] += aux[i][j-1];
}

int sumQuery(int aux[M][N], int tli, int tlj, int rbi,
                                    int rbj)
{
   // result is now sum of elements between (0, 0) and
   // (rbi, rbj)
   int res = aux[rbi][rbj];

   // Remove elements between (0, 0) and (tli-1, rbj)
   if (tli > 0)
     res = res - aux[tli-1][rbj];

   // Remove elements between (0, 0) and (rbi, tlj-1)
   if (tlj > 0)
```

```
    res = res - aux[rbi][tlj-1];


  if (tli > 0 && tlj > 0)
    res = res + aux[tli-1][tlj-1];

  return res;
}
```

Complexity:

- Time: O(Q*N*M)
- Extra space: O(N*M)

Question 9:

- **Problem link**:https://leetcode.com/problems/minimum-path-sum/
- **Difficulty level**: Hard

Explanation:

This problem is based on the topic of Dynamic Programming..Suppose the minimum path sum of arriving at point (i, j) is S[i][j], then it can be written as S[i][j] = min(S[i - 1][j], S[i][j - 1]) + grid[i][j].

Well, some boundary conditions need to be handled. The boundary conditions happen on the topmost row (S[i - 1][j] does not exist) and the leftmost column (S[i][j - 1] does not exist). Hence for the top most row, S[0][j] can only be reached by S[0][j-1] and similarly for the left most column, S[i][0] can only be reached directly from S[i-1][0];

Solution:

```
class Solution {
public:
```

```
    int minPathSum(vector<vector<int>>& grid) {
        int m = grid.size();
        int n = grid[0].size();
        vector<vector<int> > sum(m, vector<int>(n, grid[0][0]));
        for (int i = 1; i < m; i++)
            sum[i][0] = sum[i - 1][0] + grid[i][0];
        for (int j = 1; j < n; j++)
            sum[0][j] = sum[0][j - 1] + grid[0][j];
        for (int i = 1; i < m; i++)
            for (int j = 1; j < n; j++)
                sum[i][j]  = min(sum[i - 1][j], sum[i][j - 1]) + grid[i][j];
        return sum[m - 1][n - 1];
    }
};
```

Complexity:

- Time: $O(N^2)$
- Extra space: $O(N^2)$