**Minimum Cost Path**

**Hard**Accuracy: 50.09% Submissions: 17096 Points: 8

Given a square **grid**of size **N**, each cell of which contains integer cost which represents a cost to traverse through that cell, we need to find a path from top left cell to bottom right cell by which the total cost incurred is minimum.  
From the cell (i,j) we can go (i,j-1), (i, j+1), (i-1, j), (i+1, j).

**Note:**It is assumed that negative cost cycles do not exist in the input matrix.

**Example 1:**

**Input:** grid = {{9,4,9,9},{6,7,6,4},

{8,3,3,7},{7,4,9,10}}

**Output:** 43

**Explanation:** The grid is-

9 4 9 9

6 7 6 4

8 3 3 7

7 4 9 10

The minimum cost is-

9 + 4 + 7 + 3 + 3 + 7 + 10 = 43.

**Example 2:**

**Input:** grid = {{4,4},{3,7}}

**Output:** 14

**Explanation:** The grid is-

4 4

3 7

The minimum cost is- 4 + 3 + 7 = 14.

**Your Task:**  
You don't need to read or print anything. Your task is to complete the function **minimumCostPath()**which takes grid as input parameter and returns the minimum cost to react at bottom right cell from top left cell.

**Expected Time Compelxity:**O(n2\*log(n))  
**Expected Auxiliary Space:**O(n2) 

**Constraints:**  
1 ≤ n ≤ 500  
1 ≤ cost of cells ≤ 1000

class Solution {

    public:

    //Function to return the minimum cost to react at bottom

    //right cell from top left cell.

    int minimumCostPath(vector<vector<int>>& grid) {

        // Code here

        int n=grid.size();

        vector<vector<int>> visited(n, vector<int> (n, 0)), sol(n, vector<int> (n, INT\_MAX));

        sol[0][0]=grid[0][0];

        priority\_queue<vector<int>, vector<vector<int>>, greater<vector<int>> > pq;

        pq.push({grid[0][0], 0, 0});

        while (!pq.empty()) {

            vector<int> v=pq.top();

            pq.pop();

            int x=v[0], i=v[1], j=v[2];

            if (i==n-1 and j==n-1) return sol[n-1][n-1];

            if (i-1>=0 and !visited[i-1][j] and sol[i-1][j]>x+grid[i-1][j]) {

                sol[i-1][j]=x+grid[i-1][j];

                pq.push({sol[i-1][j], i-1, j});

                visited[i-1][j]=1;

            }

            if (j-1>=0 and !visited[i][j-1] and sol[i][j-1]>x+grid[i][j-1]) {

                sol[i][j-1]=x+grid[i][j-1];

                pq.push({sol[i][j-1], i, j-1});

                visited[i][j-1]=1;

            }

            if (i+1<n and !visited[i+1][j] and sol[i+1][j]>x+grid[i+1][j]) {

                sol[i+1][j]=x+grid[i+1][j];

                pq.push({sol[i+1][j], i+1, j});

                visited[i+1][j]=1;

            }

            if (j+1<n and !visited[i][j+1] and sol[i][j+1]>x+grid[i][j+1]) {

                sol[i][j+1]=x+grid[i][j+1];

                pq.push({sol[i][j+1], i, j+1});

                visited[i][j+1]=1;

            }

            //visited[i][j]=1;

        }

        return sol[n-1][n-1];

    }

};