Matrix Median

BRUTE FORCE :

we store all elements in a 1d array and then sot the array then return the median.

```
#include <bits/stdc++.h>
using namespace std;
// Function to find median  of row wise sorted matrix
int Findmedian(int arr[3][3], int row, int col)
{
  int median[row * col];
  int index = 0;
  for (int i = 0; i < row; i++)
  {
    for (int j = 0; j < col; j++)
    {
      median[index] = arr[i][j];
      index++;
    }
  }

  return median[(row * col) / 2];
}
int main()
{
  int row = 3, col = 3;
  int arr[3][3] = {{1, 3, 8},
                   {2, 3, 4},
                   {1, 2, 5}};
  cout <<"The median of the row-wise sorted matrix is: "<<Findmedian
  (arr, row, col) << endl;
  return 0;
}
```

- Time Complexity : row*col log(row*col)
- Space Complexity : O(row*col)

Optimal Approach :

In the problem we are given range 1 to 10^5. So our search space becomes this range with low as 1 and high as 10^5. We find a mid element and check the number of elements less than this mid element if the size of vector is m*n then the median will be at position median=(m*n)/2 we compare the number of elements less than the mid

element in the vector using binary search with the median. We do this until low crosses high.

If median and mid element count is less then move to right and if count is greater we move towards left . When low and high crosses then the ans will be stored in low. For finding count of element less than mid element we again use binary search on all rows as the row is sorted of the matrix and add all up. The motive is to keep minimizing the search space.

```
int findCount(int mid,vector<int>& mat)
{
    int n=mat.size();
    int l=0;
    int h=n-1;
    while(l<=h)
    {
        int md=l+(h-l)/2;
        if(mat[md]<=mid)
        {
            l=md+1;
        }
        else
        {
            h=md-1;
        }
    }
    return l;
}
int getMedian(vector<vector<int>> &matrix)
{
    // Write your code here.
    int m=matrix.size();
    int n=matrix[0].size();
    int middle=(m*n)/2;
    int low=1;
    int high=1e5;
    while(low<=high)
    {
        int mid=low+(high-low)/2;
        int count=0;
        for (int i = 0; i < matrix.size(); i++)
        {
          count += findCount(mid,matrix[i]);
        }
        if(count<=(n*m)/2)
        {
            low=mid+1;
        }
        else{
```

```
            high=mid-1;
        }
    }
    return low;

}
```

- Time Complexity : O(row* log(col))
- Space Complexity :O(1)