



Dash



All



Articles



Videos



Problems



Quiz

<< Prev

Next >>

Median of a Row Wise Sorted Matrix

We are given a row-wise sorted matrix of size $r \times c$, we need to find the median of the matrix given. It is assumed that $r \times c$ is always odd.

Examples:

Input : 1 3 5

2 6 9

3 6 9

Output : Median is 5

If we put all the values in a sorted
array $A[] = 1\ 2\ 3\ 3\ 5\ 6\ 6\ 9\ 9$)

Input: 1 3 4

2 5 6

7 8 9

Output: Median is 5

Simple Method: The simplest method to solve this problem is to store all the elements of the given matrix in an array of size $r \times c$. Then we can either sort the array and find the median element in $O(r \times c \log(r \times c))$ or we can use the approach discussed here to find the median in $O(r \times c)$. Auxiliary space required will be $O(r \times c)$ in both cases.

An **efficient approach** for this problem is to use a binary search algorithm. The idea is that for a number to be median there should be exactly $(n/2)$ numbers that are less than this number. So, we try to find the count of numbers less than

all the numbers. Below is the step by step algorithm for this approach:

Algorithm:

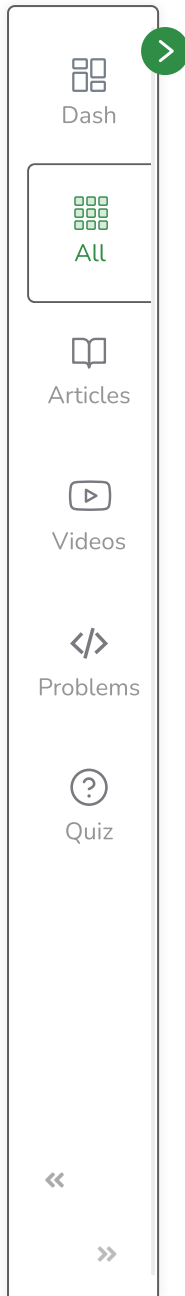
1. First, we find the minimum and maximum elements in the matrix. The minimum element can be easily found by comparing the first element of each row, and similarly, the maximum element can be found by comparing the last element of each row.
2. Then we use binary search on our range of numbers from minimum to maximum, we find the mid of the min and max and get a count of numbers less than or equal to our mid. And accordingly change the min or max.
3. For a number to be median, there should be $(r*c)/2$ numbers smaller than that number. So for every number, we get the count of numbers less than that by using `upper_bound()` in each row of the matrix, if it is less than the required count, the median must be greater than the selected number, else the median must be less than or equal to the selected number.

Below is the implementation of the above approach:

C++**Java**

```
// Java program to find median of a matrix
// sorted row wise
import java.util.Arrays;

public class MedianInRowSorted
{
    // function to find median in the matrix
    static int binaryMedian(int m[][],int r, int c)
    {
        int max = Integer.MIN_VALUE;
```



```
int min = Integer.MAX_VALUE;

for(int i=0; i<r ; i++)
{
    // Finding the minimum element
    if(m[i][0] < min)
        min = m[i][0];

    // Finding the maximum element
    if(m[i][c-1] > max)
        max = m[i][c-1];
}


int desired = (r * c + 1) / 2;
while(min < max)
{
    int mid = min + (max - min) / 2;
    int place = 0;
    int get = 0;


    // Find count of elements smaller than mid
    for(int i = 0; i < r; ++i)
    {


        get = Arrays.binarySearch(m[i],mid);


        // If element is not found in the array the
        // binarySearch() method returns
```






Dash


All


Articles


Videos


Problems


Quiz


Courses

Tutorials

Jobs

Practice

Upcoming
Contests



«

»

```

// (-(insertion_point) - 1). So once we know
// the insertion point we can find elements
// Smaller than the searched element by the
// following calculation
if(get < 0)
    get = Math.abs(get) - 1;

// If element is found in the array it returns
// the index(any index in case of duplicate). So we go to last
// index of element which will give the number of
// elements smaller than the number including
// the searched element.
else
{
    while(get < m[i].length && m[i][get] == mid)
        get += 1;
}

place = place + get;
}

if (place < desired)
    min = mid + 1;

return min;
}

```





Dash



All



Articles



Videos



Problems



Quiz



```
// Driver Program to test above method.
public static void main(String[] args)
{
    int r = 3, c = 3;
    int m[][]= { {1,3,5}, {2,6,9}, {3,6,9} };

    System.out.println("Median is " + binaryMedian(m, r, c));
}

// This code is contributed by Sumit Ghosh
```



Output

Median is 5

Time Complexity: $O(32 * r * \log(c))$. The upper bound function will take $\log(c)$ time and is performed for each row. And since the numbers will be max of 32 bit, so binary search of numbers from min to max will be performed in at most 32 ($\log_2(2^{32}) = 32$) operations.

Auxiliary Space: $O(1)$

[Mark as Read](#)

 Report An Issue

If you are facing any issue on this page. Please let us know.

