Minimum absolute difference between any of the two adjacent elements of an array of size N, which is created by picking one element from each row of the matrix.

Aditya Singh Machhaiya IIT2019111
IV Semester
Department of IT
IIIT Allahabad

Vangmayi Payili IIT2019112
IV Semester
Department of IT
IIIT Allahabad

Nimish Upadhyay IIT2019113
IV Semester
Department of IT
IIIT Allahabad

Problem: Given a matrix of N rows and M columns, the task is to find the minimum absolute difference between any of the two adjacent elements of an array of size N, which is created by picking one element from each row of the matrix. Note the element picked from row 1 will become arr[0], element picked from row 2 will become arr[1] and so on. Solve using divide and conquer algorithm.

***Abstract*—In this paper, we have devised an algorithm to find the minimum absolute difference between any of the two adjacent elements of an array of size N, which is created by picking one element from each row of the matrix by using divide and conquer algorithm.**

I. Introduction

In this problem we have to give the minimum absolute difference between any of the two adjacent elements of an array of size N, which is created by picking one element from each row of the matrix. This is being done using divide and conquer algorithm.

i)Divide and Conquer Algorithm:-
A divide-and-conquer algorithm recursively breaks down a problem into two or more sub-problems of the same or related type, until these become simple enough to be solved directly. The solutions to the sub-problems are then combined to give a solution to the original problem.

This technique can be divided into the following three parts:

- Divide: This involves dividing the problem into some sub problem.
- Conquer: Sub problem by calling recursively until sub problem solved.
- Combine: The Sub problem Solved so that we will get find problem solution.

ii)Binary Search:- Binary search is a search algorithm that finds the position of a target value within a sorted array. Binary search compares the target value to the middle element of the array. If they are not equal, the half in which the target cannot lie is eliminated and the search continues on the remaining half, again taking the middle element to compare to the target value, and repeating this until the target value is found. If the search ends with the remaining half being empty, the target is not in the array.
In each step, the algorithm compares the input element x with the value of the middle element in array. If the values match, return the index of the middle. Otherwise, if x is less than the middle element, then the algorithm recurs for left side of the middle element, else recurs for the right side of the middle element.

II. Algorithm

A. Algorithm Description

The idea is to sort all rows individually and then do binary search to find the closest element in next row for each element. To do this in an efficient manner, sort each row of the matrix.

Starting from row 1 to $rowN{-}1$ of matrix, for each element $m[i][j]$ of current row in the matrix, find the smallest element in the next row which is greater than or equal to the current element, say p and the largest element which is smaller than the current element, say q. This can be done using Binary Search. Finally, find the minimum of the difference of current element from p and q and update the variable.

B. Algorithm Analysis

1) Taking input about no. of rows and no. of columns and generating input from input generator function storing it in a 2-d matrix.

2) Sorting the rows of the matrix using merge sort which is divide and conquer algorithm.

3) Iterating for every row except the last row ,and finding upper bound for each element of the row which we are

iterating.

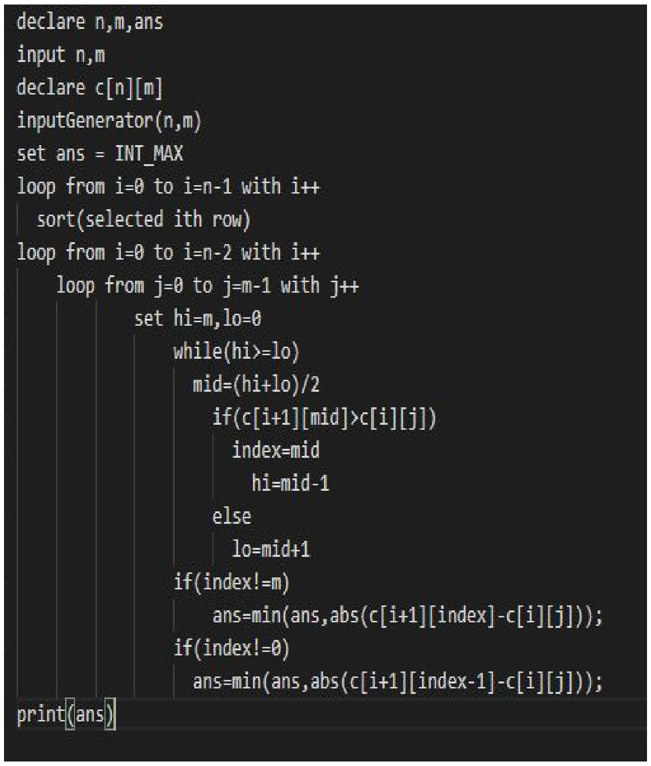
4) We will find upper bound using binary search.

5) Find the answer corresponding the upper bound and also for upperbound-1 index (if it exist).

6) Check the answer is less than then answer stored earlier ,if yes then store it.

C. Pseudo Code

- sort
- consider diff between the first pair as min
- compare all "consecutive pair min" with the one in step2 to get the least min.



```
declare n,m,ans
input n,m
declare c[n][m]
inputGenerator(n,m)
set ans = INT_MAX
loop from i=0 to i=n-1 with i++
  sort(selected ith row)
loop from i=0 to i=n-2 with i++
    loop from j=0 to j=m-1 with j++
            set hi=m,lo=0
                while(hi>=lo)
                  mid=(hi+lo)/2
                    if(c[i+1][mid]>c[i][j])
                      index=mid
                        hi=mid-1
                    else
                      lo=mid+1
                if(index!=m)
                    ans=min(ans,abs(c[i+1][index]-c[i][j]));
                if(index!=0)
                  ans=min(ans,abs(c[i+1][index-1]-c[i][j]));
print(ans)
```

Fig. 1. pseudo code.

```
int mindiff(int c[1000][1000], int n, int m)
  {
  //Sort each row of the matrix.
  for (int i = 0; i ¡ n; i++)
    sort(c[i], c[i] + m);
  int ans = INT_MAX;
  for(int $i = 0; i < n - 1; i + +)\{$
    for(int $j = 0; j < m; j + +)\{$
        int lo=0,hi=m,mid,index=m;
          //applying upperbound(using binary search
          //which is divide and conquer alogrithm)
          while($hi >= lo$){
          mid=$(hi + lo)/2$;
            //checking the weather the $c[i + 1][mid]$
            //can be a upperbound or not
            if($c[i + 1][mid] > c[i][j]$){
            $index = mid$;
            $hi = mid - 1$;
            }
            else {
            $lo = mid + 1$;
            }
            if($index = m$)
            $ans = min(ans, abs(c[i + 1][index] - c[i][j]))$;
            // checking that answer with just
            //less than element $c[i][j]$
            if($index! = 0$)
            $ans = min(ans, abs(c[i+1][index-1]-c[i][j]))$;
            }
        }
      }
  return ans;
    }
```

D. Time Complexity

Let t(n,m) be the time complexity of the upper bound code time complexity for traversing n*m matrix :$O(n * m)$;

$$t(n, m) = o(n * m) + 2 * t(n, m/2)$$

$$t(n, m/2) = o(n * m/2) + 2 * t(n, m/4)$$

$$t(n, m/4) = o(n * m/4) + 2 * t(n, m/8)$$

$$t(n, m/8) = o(n * m/8) + 2 * t(n, m/16)$$

and so on ....
hence after solving these equations ,
we get $t(n, m) = O(n * m * log(m))$
let T(n,m) be the time complexity of the whole code
TIME COMPLEXITY of merge sort = $O(n * m * log(m))$

Hence, $T(n, m) = o(n * m * log(m)) + o(n * m * log(m))$
hence , $T(n, m) = o(n * m * log(m))$

Table for time with different input:-

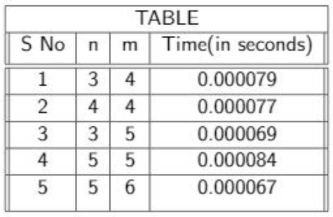


TABLE			
S No	n	m	Time(in seconds)
1	3	4	0.000079
2	4	4	0.000077
3	3	5	0.000069
4	5	5	0.000084
5	5	6	0.000067

Fig. 2. time complexity.

E. Space Complexity

Let n = no. of rows , m = No. of coloumns
space occupied by the 2-d array : $O(n * m)$
space complexity of merge sort : $O(m)$
Total Space complexity : $O(n * m) + O(m)$ hence ,
Total Space complexity :$O(n * m)$

REFERENCES

[1] Introduction to divide and conquer technique:https://www.geeksforgeeks.org/divide-and-conquer-algorithm-introduction
[2] Introduction to Algorithms by Cormen,Charles,Rivest and Stein.https://web.ist.utl.pt/ fabio.ferreira/material/asa