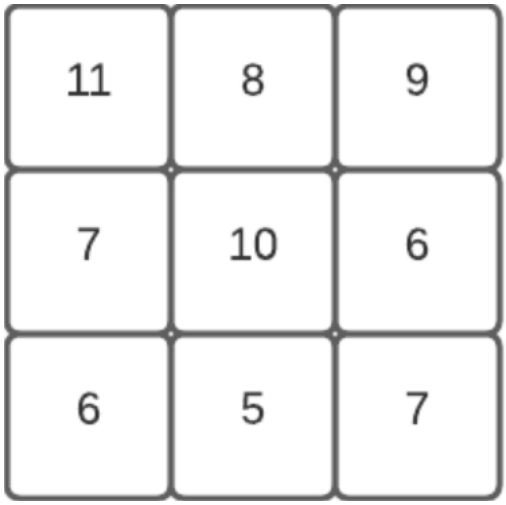
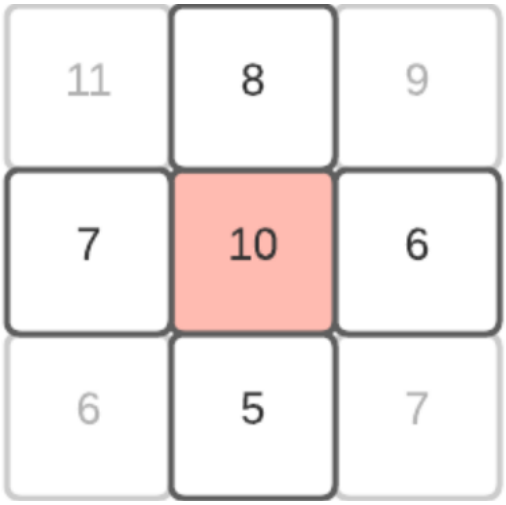
Assignment - 05

Team members:
Mahitha Gurrala(U19CS066)
Krithikha Balamurugan (U19CS076)
Guntuboina Venkata Sankirtana (U19CS068)

1.1 Find a computational problem that you can solve using the divide and conquer approach. The problem should be different from the problems discussed in class,and it should be unique and interesting.

PROBLEM STATEMENT: TO FIND A PEAK IN THE 2D ARRAY

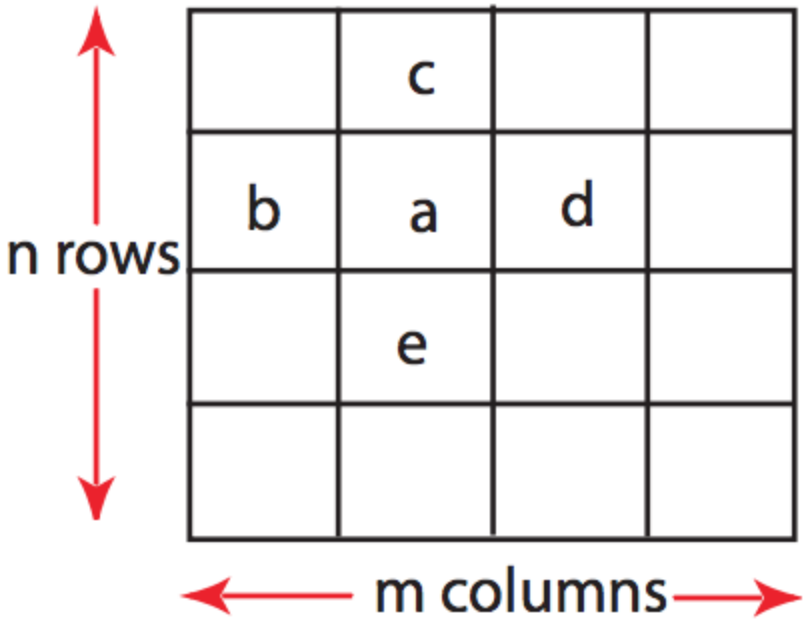




A peak element is defined as -

An element is greater than or equal to its four neighbors, **left, right, top and bottom**. For example neighbors for A[i][j] are A[i−1][j], A[i+1][j], A[i][j−1] and A[i][j+1],then the element is called a **peak element**.

The side elements are considered accordingly for their neighbours.

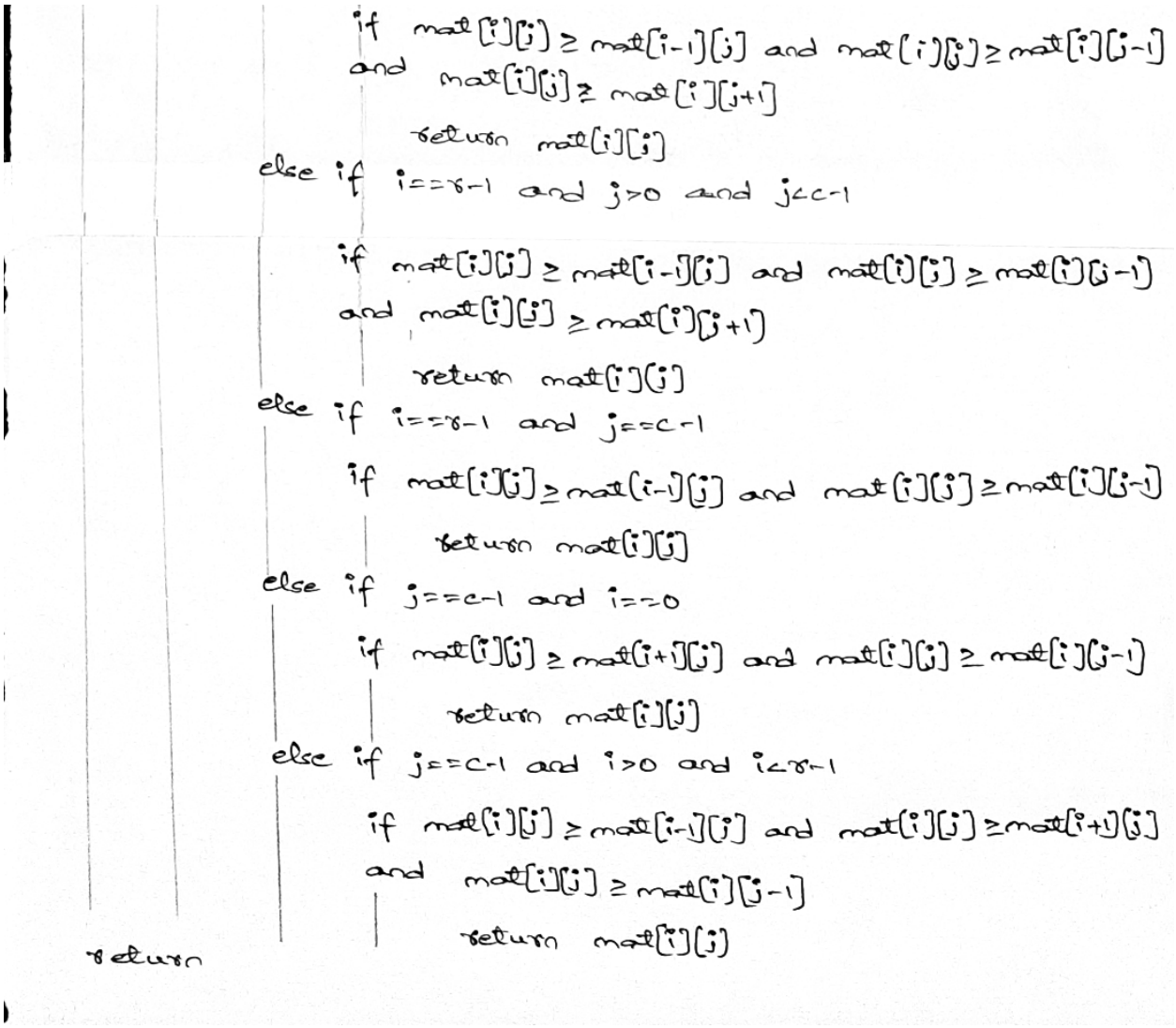


9	3	5	2	4	9	8
7	2	5	1	4	0	3
9	8	9	3	2	4	8
7	6	3	1	3	2	3
9	0	6	0	4	6	4
8	9	8	0	5	3	0
2	1	2	1	1	1	1

1.2 Write pseudocodes to design algorithms for the above mentioned computational problem using the brute-force approach (incremental approach) and the divide and conquer approach.

Brute force:

```
Let mat be the 2D array
Let r and c be the number of rows and columns
respectively
Peak (mat)
if r==1 and c==1
    return mat[0][0]
for (i=0; i<r; i++)
    for (j=0; j<c; j++)
        if i!=0 and j!=0 and i!=r-1 and j!=c-1
            if (mat[i][j] ≥ mat[i-1][j] and mat[i][j] ≥ mat[i+1][j]
            and mat[i][j] ≥ mat[i][j-1] and mat[i][j] ≥ mat[i][j+1])
                return mat[i][j]
        else if i==0 and j>0 and j<c-1
            if mat[i][j] ≥ mat[i+1][j] and mat[i][j] ≥ mat[i][j-1]
                and mat[i][j] ≥ mat[i][j+1]
                    return mat[i][j]
        else if j==0 and i>0 and i<r-1
            if mat[i][j] ≥ mat[i-1][j] and mat[i][j] ≥ mat[i+1][j]
            and mat[i][j] ≥ mat[i][j+1]
                return mat[i][j]
        else if j==0 and i==0
            if mat[i][j] ≥ mat[i+1][j] and mat[i][j] ≥ mat[i][j+1]
                return mat[i][j]
        else if i==r-1 and j==0
```



Divide and conquer:

EFFICIENT ALGORITHM APPROACH EXPLAINED-

- Pick middle column j = m/2
- Find global maximum on column j at (i, j)
- Compare (i, j − 1),(i, j),(i, j + 1)
- Pick left columns of (i, j − 1) > (i, j)
- Similarly for right
- (i, j) is a 2D-peak if neither condition holds
- Solve the new problem with half the number of columns.
- When you have a single column, find global maximum and you‘re done.

Let matrix be the 2D array

Let c and r be the number of columns and rows respectively

Let mid be a variable.

Initially $mid = \frac{c}{2}$

```
Peak (matrix, mid)
max = 0, index = 0
if   mid == 1  or  mid == c
     return max

for (i = 0; i < r; i++)
     if  max < matrix[i][mid]
          index = i
          max = matrix[i][mid]
if  max ≥ matrix[index][mid-1]
and  max ≥ matrix[index][mid+1]
     return max
if  max < matrix[index][mid-1]
     return Peak(matrix, mid - ceil(mid/2))
if  max < matrix[index][mid+1]
     return Peak(matrix, mid + ceil(mid/2))
return
```

1.3 Analyze the time complexity of above algorithms. Analyze the divide and conquer algorithm using different methods such as (1) Recursion Tree Method (2) Iterative Method (3) Master Method (4) Substitution Method. Try to analyze the algorithm using as many methods as discussed above.

ITERATIVE

Let $C_1, C_2, C_3, \ldots, C_{31}$ be the costs.

Best Case:

Running time $= C_1 + C_2$

$=$ constant

Time Complexity $= O(1)$

Worst Case:

Running time $= C_1 \times 1 + C_3 \times r + C_4 \times (r \times c) + C_5 \times (r \times c) + C_6 \times ((r-1) \times (c-1)) + C_7 \times 0$
$+ C_8 \times (r \times c) + C_9 \times (c-2) + C_{11} \times (r \times c) + C_{12} \times (r-2) + C_{14} \times (r \times c)$
$+ C_{15} \times 1 + C_{17} \times (r \times c) + C_{18} \times 1 + C_{20} \times (r \times c) + C_{21} \times (r-2)$
$+ C_{22} \times (r \times c) + C_{23} \times (1) + C_{25} \times (r \times c) + C_{26} + C_{28} \times (r \times c)$
$+ C_{29} \times (r-2) + C_{30}$

Running time $\Rightarrow b_3 * (r * c) + b_1 * r + b_2 * c + b_4$

where, b_1, b_2, b_3, b_4 are constants.

$\therefore$ Time Complexity $= O(r \times c)$

$r \rightarrow$ number of rows

$c \rightarrow$ number of columns

SUBSTITUTION

Substitution method

$T(n,m)$ denotes work required to solve matrix with n rows m columns

$$T(n,m) = T(n, m/2) + \Theta(n) \quad \text{—(1)}$$

$\rightarrow \Theta(n)$ to find global maximum on column

$$T(n, m/2) = T(n, m/4) + \Theta(n)$$

Substituting in (1)

$$T(n,m) = \Theta(n) + \Theta(n) + T(n, m/4)$$

$$\vdots$$

$$T(n, m/k) = T(n, \frac{m}{k*2}) + \Theta(n)$$

$$\vdots$$

it goes on until $m/k = 1$ and $T(1) = O(1)$

since $T(1) = O(1)$

$\Rightarrow T(n,m) = \Theta(n) + \Theta(n) + \Theta(n) \cdots$ [log m times]

$= \log m \times \Theta(n)$

$= \Theta(n \log m)$

RECURSION TREE

Recursive Tree method

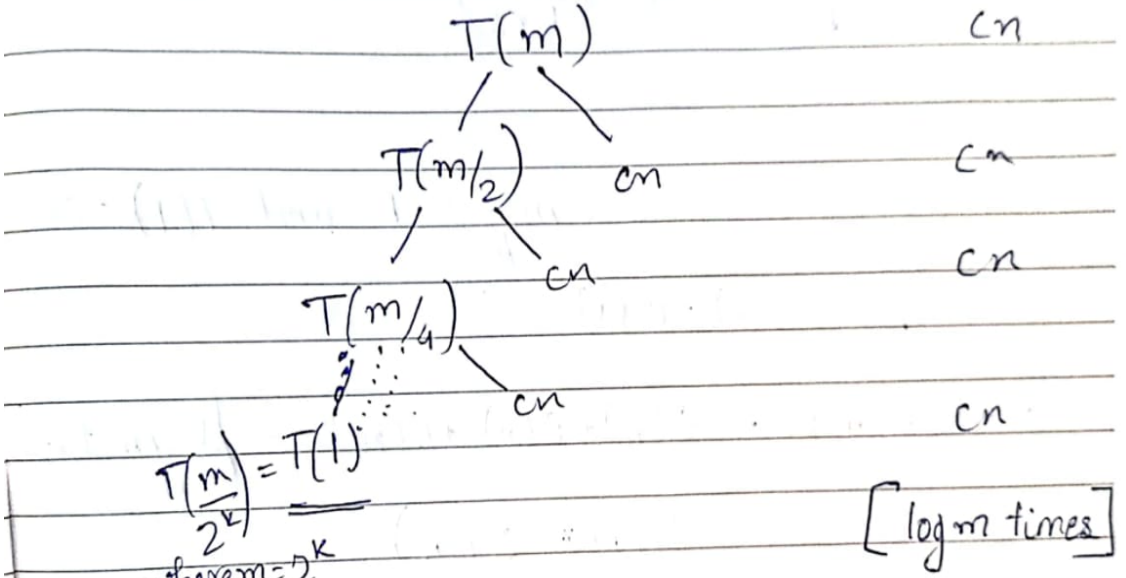
Each Matrix $(n \times m)$ has n rows and m colow
To find the maximum element in each row it taks $\theta(n)$

For dividing it takes $\theta(\log m)$.

$T(m)$

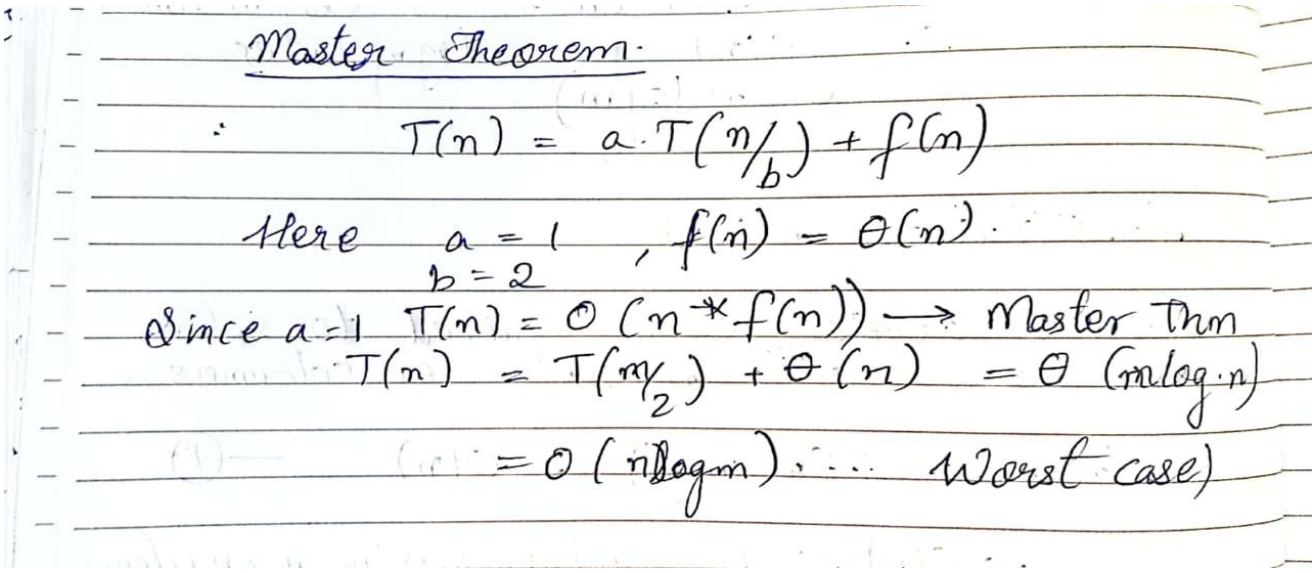
do

Divide works like



$$\text{do } T(n) = cn + cn + \cdots \text{ (log m times)}$$
$$= \theta(n) \cdots \log n \text{ times}$$
$$= O(n \log m)$$

MASTER METHOD



1.4 Provide the details of Hardware/Software you used to implement algorithms and to measure the time.

Compiler	Dev C++ 5.11
OS Name	Microsoft Windows 10 Home (i5 8th Gen)
Version	10.0.19042 Build 19042
System Name	DESKTOP-BLE6CMQ
System Model	HP Pavilion x360 Convertible 14-ba1xx
System Type	x64-based PC
Processor	Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz, 1800 Mhz, 4 Core(s), 8 Logical Processor(s)
BIOS Version/Date	Insyde F.54, 04-12-2019
Installed Physical Memory (RAM)	8.00 GB
Total Physical Memory	7.88 GB
Available Physical Memory	1.75 GB
Total Virtual Memory	12.4 GB
Available Virtual Memory	4.59 GB
Page File Space	4.50 GB

1.5 Implement the above algorithms and submit the code (complete programs).

ITERATIVE APPROACH-

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

clock_t begin, end;
double time_;

int** matrix_allocate(int n,int m)
{
  int i, j;
  int** mat = (int** ) malloc(n * sizeof(int* ));
  for (i = 0; i < n; ++i)
  {
          mat[i] = (int* )malloc(m * sizeof(int));
  }

  return mat;
}

int Peak(int **mat, int r,int c)
{
      if (r == 1&&c==1)
      {
       return mat[0][0];
      }
      for (int i = 0; i <r; i++)
      {
         for(int j=0;j<c;j++)
            {
                  if(j==0 && i==0)
                  {

                        if (mat[i][j] >= mat[i+1][j] && mat[i][j] >= mat[i][j+1])
                        {
                              return mat[i][j];
                        }
                  }
                  else if(i==0 && j>0 && j<c-1)
                  {
```

```
                        if (mat[i][j] >= mat[i+1][j]&&mat[i][j] >= mat[i][j-1]
&& mat[i][j] >= mat[i][j+1])
                        {
                              return mat[i][j];
                        }
                  }
                  else if(j==0 && i>0 && i<r-1)
                  {

                        if (mat[i][j] >= mat[i-1][j] && mat[i][j] >= mat[i+1][j] &&
mat[i][j] >= mat[i][j+1])
                        {
                              return mat[i][j];
                        }
                  }
                  else if(i==r-1 && j==0)
                  {
                        if (mat[i][j] >= mat[i-1][j] && mat[i][j] >= mat[i][j-1] &&
mat[i][j] >= mat[i][j+1])
                        {
                              return mat[i][j];
                        }
                  }
                  else if(i==r-1&& j>0 && j<c-1)
                  {
                        if (mat[i][j] >= mat[i-1][j] &&mat[i][j] >= mat[i][j-1] &&
mat[i][j] >= mat[i][j+1])
                        {
                              return mat[i][j];
                        }
                  }
                        else if(i==r-1 &&  j==c-1)
                  {
                        if (mat[i][j] >= mat[i-1][j] && mat[i][j] >= mat[i][j-1])
                        {
                              return mat[i][j];
                        }
                  }
                  else if(j==c-1 && i==0)
                  {
                        if (mat[i][j] >= mat[i+1][j]&&mat[i][j] >= mat[i][j-1])
                        {
                              return mat[i][j];
                        }
                  }
```

```
                else if(j==c-1 && i>0 && i<r-1)
                {
                      if    (mat[i][j]    >=    mat[i-1][j]    &&    mat[i][j]    >=
mat[i+1][j]&&mat[i][j] >= mat[i][j-1])
                      {
                            return mat[i][j];
                      }
                }
                else
                {
                      if    (mat[i][j]    >=    mat[i-1][j]    &&    mat[i][j]    >=
mat[i+1][j]&&mat[i][j] >= mat[i][j-1] && mat[i][j] >= mat[i][j+1])
                      {
                            return mat[i][j];
                      }
                }
          }
     }
}
int main()
{
          int r,c;
          printf("Enter rows and columns:");
          cin>>r>>c;
          int** mat = matrix_allocate(r,c);
          printf("\nEnter matrix elements:");
          for(int i=0;i<r;i++)
          {
                for(int j=0;j<c;j++)
                {
                      cin>>mat[i][j];
                      cout<<mat[i][j];
                }
          }
          int mid=c/2;
          begin = clock();
          cout << "\n\nFirst Peak element is : "<<Peak(mat, r, c);
          end = clock();
          time_= ((double)(end-begin)) / CLOCKS_PER_SEC;
                      cout<<"\nTime taken :"<<time_;
    return 0;
}
```

DIVIDE AND CONQUER APPROACH-

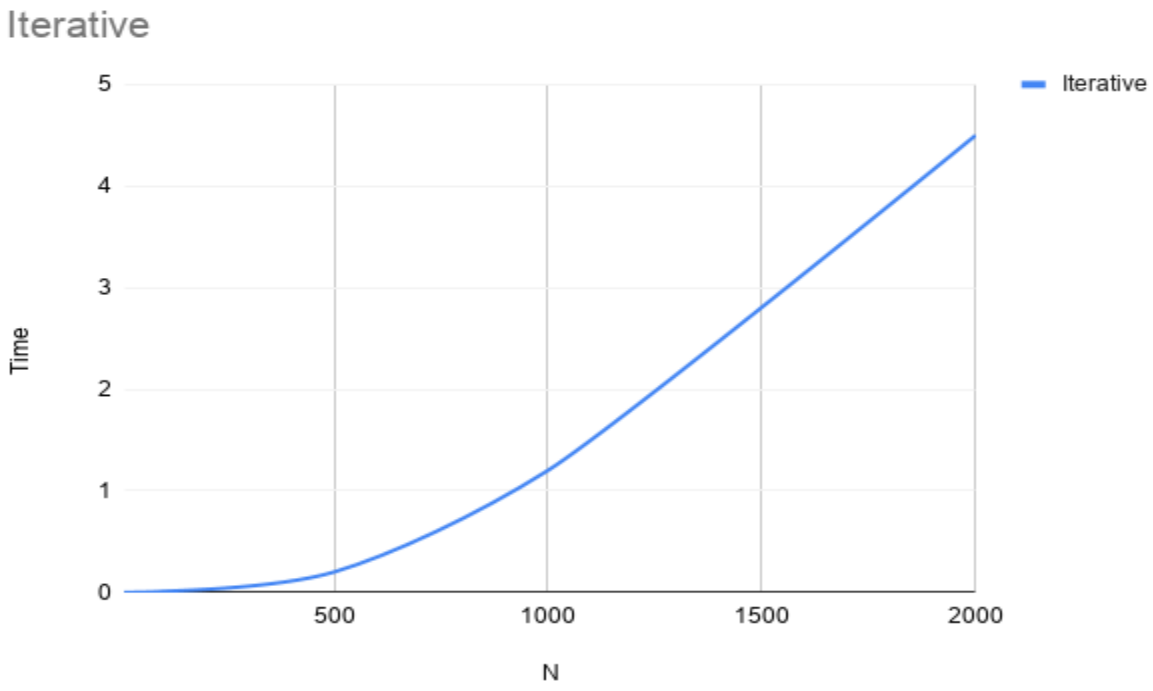
```
#include <bits/stdc++.h>
#include <time.h>
#define MAX 1000
using namespace std;

clock_t begin, end;
double time_;
int** matrix_allocate(int n,int m)
{
  int i, j;
  int** mat = (int** ) malloc(n * sizeof(int* ));
  for (i = 0; i < n; ++i)
  {
          mat[i] = (int* ) malloc(m * sizeof(int));
  }
  return mat;
}

int Peak(int **mat, int r, int c, int mid)
{
    int max = 0;

    int ind=0;
      for (int i = 0; i < r; i++)
      {
         if (max < mat[i][mid])
            {
             max = mat[i][mid];
             ind = i;
         }
    }

       if (mid == 0 || mid == c - 1)
         return max;

    if (max >= mat[ind][mid - 1] && max >= mat[ind][mid + 1])
         return max;

    // If max is less than its left
    if (max < mat[ind][mid - 1])
         return Peak(mat, r, c, mid + ceil((double)mid / 2));

   // if (max < mat[ind][mid+1])                 //If max is less than its right
```

```
    return Peak(mat, r, c, mid + ceil((double)mid / 2));
}


int main()
{
            int r,c;
            printf("Enter rows and columns:");
            cin>>r>>c;
            int** mat = matrix_allocate(r,c);
            printf("\nEnter matrix elements:");
            for(int i=0;i<r;i++)
            {
                  for(int j=0;j<c;j++)
                  {
                        cin>>mat[i][j];
                  }
            }
            int mid=c/2;
            begin = clock();
            cout << "\n\nFirst Peak element is : "<<Peak(mat, r, c,mid);
            end = clock();
            time_= ((double)(end-begin)) / CLOCKS_PER_SEC;
            cout<<"\nTime taken :"<<time_;

    return 0;
}
```

1.6 Analyze the performance of both the implemented algorithms (performance of algorithms on your computers). Plot a graph.

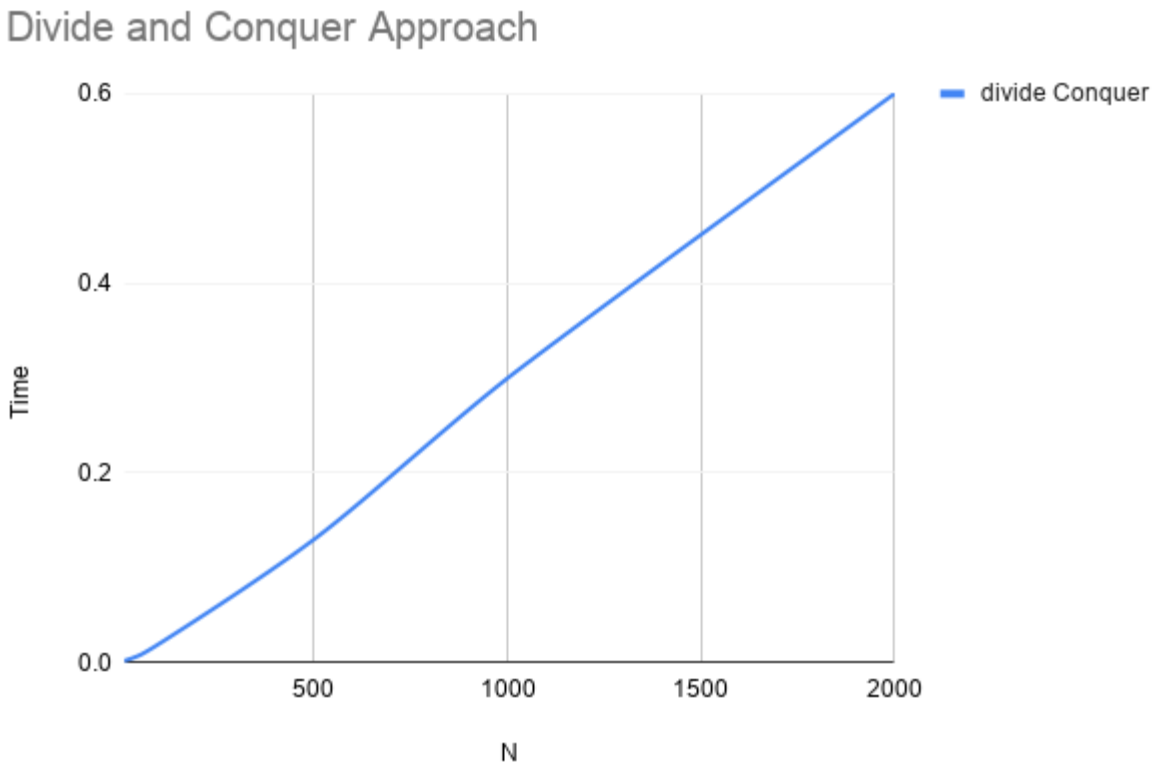
DIVIDE AND CONQUER

N	Divide Conquer
10	0.002
50	0.008
100	0.02
500	0.13
1000	0.32
2000	0.63



ITERATION METHOD

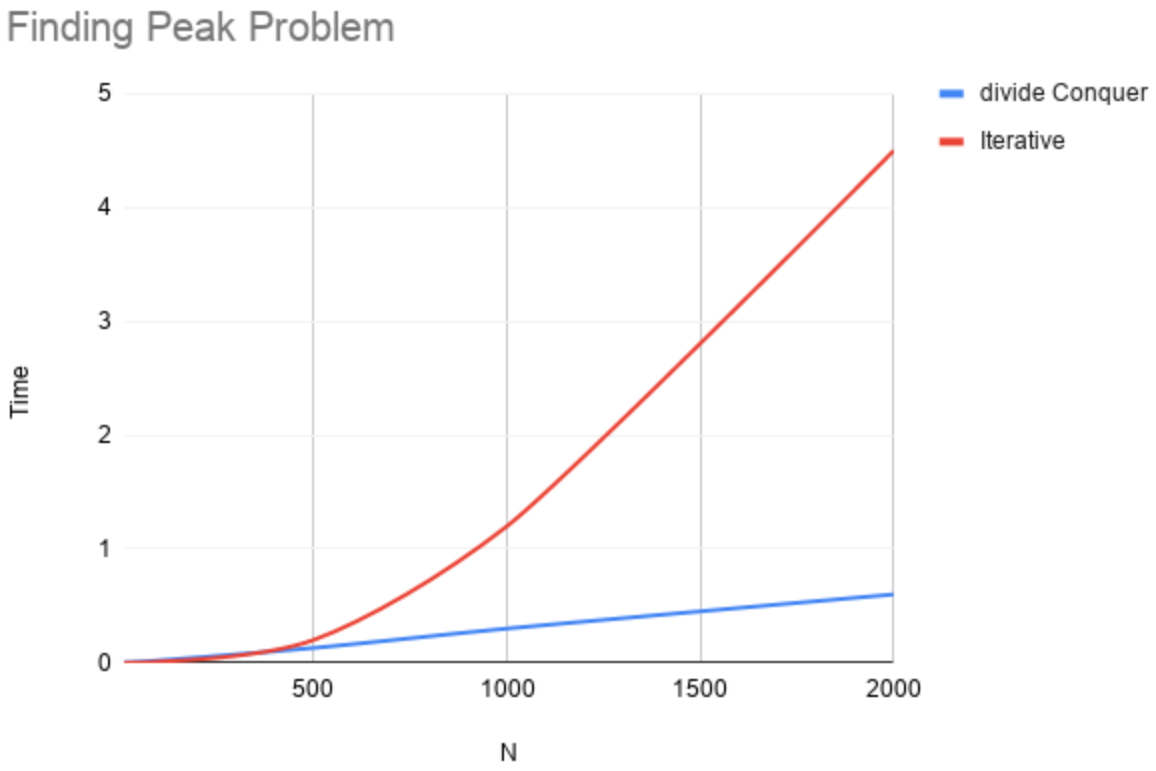
N	Iterative
10	0
50	0.002
100	0.01
500	0.2
1000	1.2
2000	4.9



1.7 Comparatively analyze the performance of above algorithms and plot a graph.

For simplicity we have taken only square matrices where n depicts the number of rows and columns.

N	Divide Conquer	Iterative
10	0.002	0
50	0.008	0.002
100	0.02	0.01
500	0.13	0.2
1000	0.32	1.2
2000	0.63	4.9



TIME COMPLEXITY

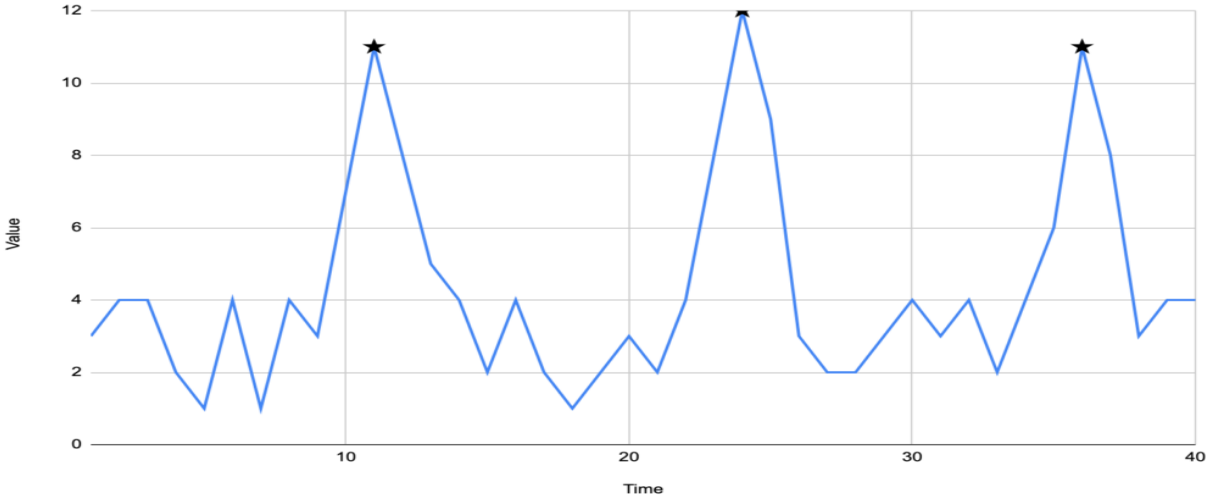
USING DIVIDE AND CONQUER: O(n*log(m))

USING ITERATION METHOD: O(n*m)

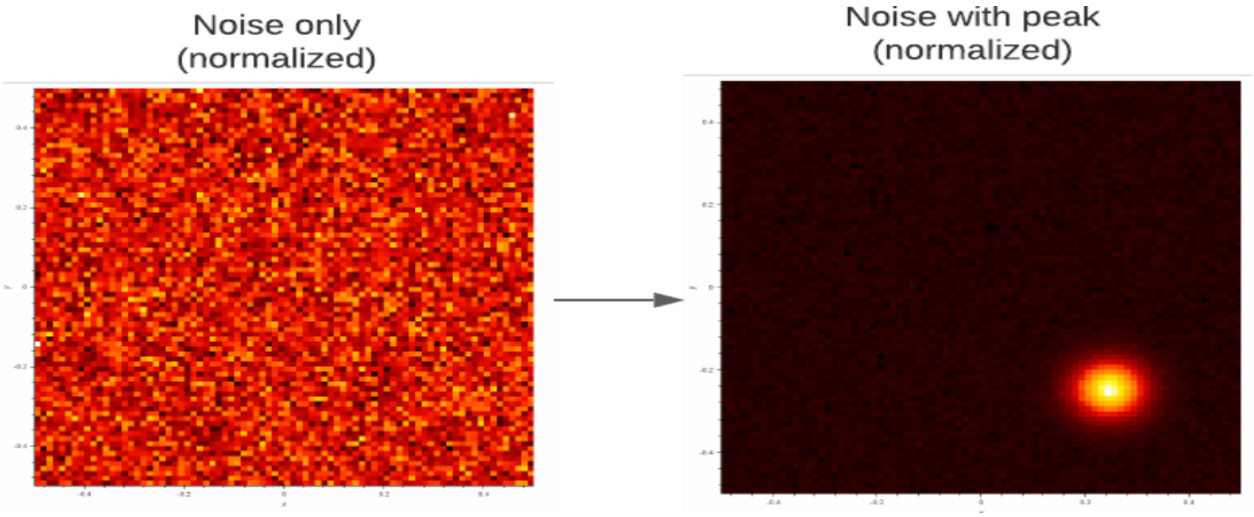
n → number of rows
m → number of columns

APPLICATIONS

- One of the common challenges in signal processing is to detect important points in time or in space like peaks. These points represent local maxima (or minima)



- **The noise is a general term of unwanted modifications of the signal**. Let's consider the example of our infrared camera. If we happen to point it to a white wall, our image would be homogeneous but due to the noise, there could be many pixel positions labeled as a peak if we take only the definition above.



- In special cases, more sophisticated algorithms are known and used like the Genetic algorithm or the famous optimization method Gradient Descent. They are appreciated mainly in scientific applications for instance