# GROUP 51- Write an efficient algorithm for the following purpose. For a given value of n (3<n), generate an (n x n) matrix whose cells are filled with randomly generated digits 0,.., 9. Now, scan the matrix with a (3 x 3) mask and find out those masks (mask positions) which contain all the digits 1,2,..,9 in some order.

IHM2016501[1]IHM2016005[2] BIM2016004[3] IIM2016006[4] RIT2015050[5]

## I. INTRODUCTION AND LITERATURE SURVEY

A Matrix is a rectangular array of numbers, symbols, or expressions, arranged in fixed rows and columns.So we represent the matrix using a $2-D$ array(i.e,here we use $a[n][n]$ to represent a matrix of size $n$).In the question, given is a value of $n$ ,we have to generate the matrix of size $n$ where $3<n$ filled with randomly generated digits (i.e, 1,2,...,9).Now the aim of the question is to find a 3x3 sub matrix of the given $n$ x $n$ matrix in which all the randomly generated digits are present in any order with the help of a (3 x 3) mask.The meaning of Mask means to hide, and in programming this mask covers the desired portion to be solved and returns us the respective programmed output.So,we have to find those mask positions which contains all the digits.And we can report a mask position by only mentioning the top-left position of the mask.

## II. ALGORITHM DESIGN

This part of the report can be further divided into following sections.

### A. Input Format

In first few lines of the code we use $rand()$ function to generate a random number and assign it to the variable $n$ which defines the size of the matrix.We also use this $rand()$ function to generate the digits i.e 1,2..,9 and store them in the matrix(i.e 2-D array a[n][n]).All this is done by the $MatrixGenerator.c$ .So basically $MatrixGenerator.c$ uses a file pointer $fptr$ which points to $text.txt$ file and writes the randomly generated digits into it according to the value of $n$ generated.We then read these values from $text.txt$ into the array a[n][n] when required and this array is declared globally.

### B. Finding the Sub-matrix (3 x 3)

- The function $mask(i,j)$ contains a $check[j]$ array of size $j$ initialized to 0.
- We then traverse through the 3x3 sub-matrix whose top-left position is given by i,j.
- We then increment the $check[index]$ where index is the digit present in the $a[i][j]$.
- The above steps continue till we traverse through the entire sub matrix by incrementing duplicate i,j(i.e row,col).
- We then check the entire $check[]$ array starting from 1 $th$ index .If anyone of its element is 0 we return 0 indicating all the digits are not present in the sub-matrix else we return 1 indicating all the digits are present in the sub-matrix.
- If the $mask()$ function returns 1 then main function stores i,j in x[],y[] arrays and later prints them.

### C. Algorithm

**Algorithm 1** Brute Force

```
1: procedure MASK(i,j)                          ▷ mask function
2:     for j=0 to 9 do  check[j]=0
3:     for row=i to i+2 do
4:         for col=i to j+2 do
5:             $check[a[row][col]]++$
6:     for i=1 to 9  do
7:         if check[i] $\neq$ 1 then
8:             $return$ 0
9:     $return$ 1
1: procedure MAIN                               ▷ main function
2:     $k \leftarrow 0$
3:     for i=0 to n-3 do
4:         for j=0 to n-3 do
5:             if mask(i,j) = 1 then
6:                 $x[k] \leftarrow i$
7:                 $y[k] \leftarrow j$
8:                 $k \leftarrow k+1$
9:     print(n)
10:    for i=0 to k-1 do
11:        print(x[i],y[i])
```

**Algorithm 2** Efficient Approach

```
procedure MASK($i$,$j$)                                  ▷ mask function
    if j=0 then
        for j=0 to 9 do  check[j]=0
        for row=i to i+2 do
            for col=i to j+2 do
                $check[a[row][col]]++$
    else
        for row=i to i+2 do
            $check[a[row][j+2]]++$
            $check[a[row][j-1]]--$
    for i=1 to 9  do
        if check[i] $\neq 1$ then
            $return\ 0$
    $return\ 1$
procedure MAIN                                           ▷ main function
    $k \leftarrow 0$
    for i=0 to n-3 do
        for j=0 to n-3 do
            if mask(i,j) $= 1$ then
                $x[k] \leftarrow i$
                $y[k] \leftarrow j$
                $k \leftarrow k + 1$
    print(n)
    for i=0 to k-1 do
        print(x[i],y[i])
```

### D. Output Format

The $mask()$ function if it returns 1 the main function stores the i,j values in the x[k],y[k] arrays and increments the k.After traversing through the entire array in prints the values stored in x[],y[] arrays .

## III. ANALYSIS

### A. Analysis

The function $mask(i, j)$ checks the variable $j$ is zero or not if $j$ is zero then it initializes the $check[j]$ array of size $j$ to 0. Then we traverse through the 3X3 sub matrix whose top left position is given by i,j.Then we increase the $check[index]$ where $index$ is the digit present in $a[i][j]$.The above steps continue till we traverse through the entire sub matrix by incrementing duplicate i,j(i.e row,col). if $j$ is not zero then we traverse through the matrix by increasing the $check[index]$ where the index is $a[row][j+2]$ and also decreasing the $check[index]$ where the index is $a[row][j-1]$.We then check the entire array starting from 1 $th$ index if anyone of its element is 0 we return 0 indicating all the digits are not present in the submatrix else we return 1 indicating all the digits are present in the submatrix . If the $mask()$ function return 1 then main function stores i,j in x[],y[] arrays and later prints them.

*1) Best Case Analysis - (3 <n):* When the matrix is of size greater than 3 we will have to traverse through the entire matrix column wise in order to obtain the desired result.

$$time_{best(n>3)} \propto 13n^2 + k \text{ instructions.(k-constant)}$$
$$=> time_{best(n>3)} \propto n^2$$

$$=> O(n^2) \text{ - A Quadratic Time Computation}$$

*2) Worst Case Analysis :* As discussed above in the worst case senario also we will have to traverse through the entire matrix column wise in order to obtain the desired result.So,the worst case time complexity is equal to best case time complexity.

$$time_{worst} \propto 13n^2 + k \text{ instructions.(k-constant)}$$
$$=> time_{worst} \propto n^2$$

$$=> O(n^2) \text{ - A Quadratic Time Computation}$$

$$=> t_{best} = t_{worst}$$

*3) Average Case Analysis :* We know that

$$time_{best} <= t_{average} <= t_{worst}$$

But from above

$t_{best} = t_{worst}$

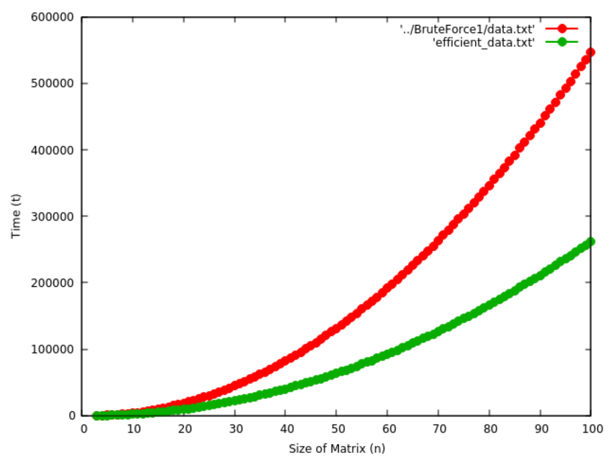
So

$=> t_{best} = t_{avg} = t_{worst}$

## IV. EXPERIMENTAL STUDY

The best way to study an algorithm is by graphs and profiling.

*1) Graphs-Time-Complexity:* We have seen that $t_{best} = t_{avg} = t_{worst}$ So,the graphs for best case,average,worst case time complexity remain the same.We can clearly see that as though Brute force and efficient algorithm are of order $n^2$ efficient algorithm is a lot better than the other one.



#### 2) *Profiling:*

$NumberOfComputations$

| $n$ | $t_{avg}$ | $t_{best}$ | $t_{worst}$ |
|---|---|---|---|
| 3 | 55 | 55 | 55 |
| 5 | 324 | 324 | 324 |
| 10 | 1999 | 1999 | 1999 |
| 20 | 9348 | 9348 | 9348 |
| 50 | 63288 | 63288 | 63288 |
| 70 | 126379 | 126379 | 126379 |
| 80 | 166266 | 166266 | 166266 |
| 100 | 261439 | 261439 | 261439 |

- We can seen that $t_{best} = t_{avg} = t_{worst}$
- We can also see that time increases as the value of n increases and $time_{worst} \propto n^2$,$time_{avg} \propto n^2$,$time_{best} \propto n^2$.

## V. DISCUSSION

### A. *Understanding the Use Of File Handling and generation of random numbers*

In this context the file handling is used to generate the random test cases through which analysis and experimentation of algorithm becomes easier for handling different test cases. We have used $srand()$ and $rand()$ to generate the random values to fill the matrix with the help of $< time.h >$ header file.

### B. *Efficient Approach*

We have already seen the explanation for brute force algorithm to solve this question. $time_{bruteforce} \propto 19n^2 + k$ instructions.(k-constant)

- But,Is this the best method to solve the question??

We have seen that for each value of i,j we traverse through a 3 x 3 matrix whose top left index is i,j.We then increment the value of i or j and traverse through the 3 x 3 matrix whose top left index is new i,new j.Here we observe that these two matrices are overlapping so why to traverse through the entire matrix when we can use the previously stored data.

In order to provide an efficient algorithm we used the previous stored values and edit them instead of creating an entirely new set.This does reduces the time taken by the algorithm which is clearly observed in the graphs mentioned above

## VI. CONCLUSION

The main module of the algorithm design is to find all the 3 x 3 matrices in a given n x n matrix which contain all the digits (i.e 1,2,...,9) by making use of mask.

We have made use of a is clearly seen from the graphs$< srand >$,$< rand >$ to generate random vales .We have also found a efficient algorithm (as tough it is of order $n^2$ it is better that the brute force algorithm which is clearly seen from the graphs).In this solution we have bounded the size of the matrix between 1 and 100 as these values are sufficient enough for us to understand and analyze the result and draw conclusions.

## REFERENCES

[1] H. Kopka and P. W. Daly, *A Guide to LaTeX*, 3rd ed. Harlow, England: Addison-Wesley, 1999.
[2] Introduction to Algorithms English By Thomas H. Cormen , By Charles E. Leiserson , Ronald L. Rivest ,Clifford Stein(3rd edition)