DESIGN AND ANALYSIS OF ALGORITHMS ASSIGNMENT-4

INDIAN INSTITUTE OF INFORMATION TECHNOLOGY, ALLAHABAD

DEPARTMENT OF INFORMATION TECHNOLOGY

IIT2019081 - Somesh kumar maurya
Mail id - iit2019081@iiita.ac.in

IIT2019082 - Navdeep kumar
Mail id - iit2019082@iiita.ac.in

IIT2019083 - Sumit bakoliya
Mail id - iit2019083@iiita.ac.in

Abstract

Problem Statement: Given an n x n matrix, where every row and column is sorted in increasing order.
Given a key, how to decide whether this key is in the matrix.

In this paper, the problem has been solved by using a divide and conquer approach by dividing a given big problem into 4 smaller subproblems and implementing it recursively using concept of recursion.

INTRODUCTION

Divide and Conquer is an algorithm design paradigm in which we recursively break down a problem into two or more sub-problems of the same or related type, until these subproblems become simple enough to be solved directly. The solutions to the sub-problems are then combined to give a solution to the original problem.
The divide-and-conquer paradigm is often used to find an optimal solution of a problem in which we recursively simplify the problem by decreasing the constraints of a subproblem and selecting / dis-selecting a given subproblem based on some conditions.

ALGORITHM DESIGN

OVERVIEW: At first, the size of the matrix is inserted and then using a random function a matrix is generated. Now the number of keys to be checked is inserted and then that number of keys is inserted.
Now for finding whether the key is present in the matrix or not, we have used the following procedure.

Step 1: Generating function call

A function named check is created which contains parameters left x coordinate (lx), left y coordinate (ly), right x coordinate (rx), right y coordinate (ry) and key to be checked.
A particular value of lx,ly,rx,ry,key means to check for the key to be present in column lx+1 to rx and in row ly+1 to ry.

Step 2: Checking Base condition

The several base condition to be checked are:

A: If the given row column pair does not contain any value i.e. number of elements is 0, then return false.

B: If the given row column pair contains a single value i.e. number of elements is 1, then return true if the element is equal to key else return false.

C:If the biggest element in the row column pair is less than the key element or the smallest element in the row column pair is greater than the key element, then return false else check for subproblems.

Step 3: Function call for subproblems

If the given problem has row range from ly to ry and column range from lx to rx, then consider mid x coordinate mx=(lx+rx)/2 and mid y coordinate my=(ly+ry)/2.
Now function call for all 4 subproblems by using this pair of midpoint i.e. (lx,ly,mx,my) , (mx,ly,rx,my) , (lx,my,mx,ry) , (mx,my,rx,ry) .

ALGORITHM DESCRIPTION

Let the numbers be stored in a matrix mat and size of the matrix is stored in n and assuming all rows and columns are sorted in increasing order.
function prototype is
bool check(int lx , int ly , int rx , int ry , int key)

Step 1: Generating function call

int key
cin>>key
bool ans = check(0,0,n,n,key)
if(ans is true) Key is present
else Key is not present

Step 2: Checking Base condition

```
if(lx==rx || ly==ry)return false

if(rx-lx==1 && ry-ly==1)
        if(mat[rx][ry]==key)return true
        else return false

if(mat[lx][ly]>key || mat[rx][ry]<key )return false
```

Step 3: Function call for subproblems

```
mx=(lx+rx)/2
my=(ly+ry)/2
bool ans=false
ans|=check(lx,ly,mx,my,key)
ans|=check(mx,ly,rx,my,key)
ans|=check(lx,my,mx,ry,key)
ans|=check(mx,my,rx,ry,key)
return ans
```

Example for explanation of code

Assuming n=10 and the matrix to be

0	1	2	3	4	5	6	7	8	9
10	11	12	13	14	15	16	17	18	19
20	21	22	23	24	25	26	27	28	29
30	31	32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47	48	49
50	51	52	53	54	55	56	57	58	59
60	61	62	63	64	65	66	67	68	69
70	71	72	73	74	75	76	77	78	79
80	81	82	83	84	85	86	87	88	89
90	91	92	93	94	95	96	97	98	99

Searching for key 81

Example for explanation of code

check(0 , 0 , 10 , 10 , 81)
mx=5 my=5
check(0 , 0 , 5 , 5 , 81)
Does not contain key so return false
check(5 , 0 , 10 , 5 , 81)
Does not contain key so return false
check(0 , 5 , 5 , 10 , 81)
Contain key, checking further
check(5 , 5 , 10 , 10 , 81)
Does not contain key so return false

Example for explanation of code

check(0 , 5 , 5 , 10 , 81)
mx=2 my=7
check(0 , 5 , 2 , 7 , 81)
Does not contain key so return false
check(2 , 5 , 5 , 7 , 81)
Does not contain key so return false
check(0 , 7 , 2 , 10 , 81)
Contain key, checking further
check(2 , 7 , 5 , 10 , 81)
Does not contain key so return false

Example for explanation of code

check(0 , 7 , 2 , 10 , 81)
mx=1 my=8
check(0 , 7 , 1 , 8 , 81)
Does not contain key so return false
check(1 , 7 , 2 , 8 , 81)
Does not contain key so return false
check(0 , 8 , 1 , 10 , 81)
Does not contain key so return false
check(1 , 8 , 2 , 10 , 81)
Contain key, checking further

Example for explanation of code

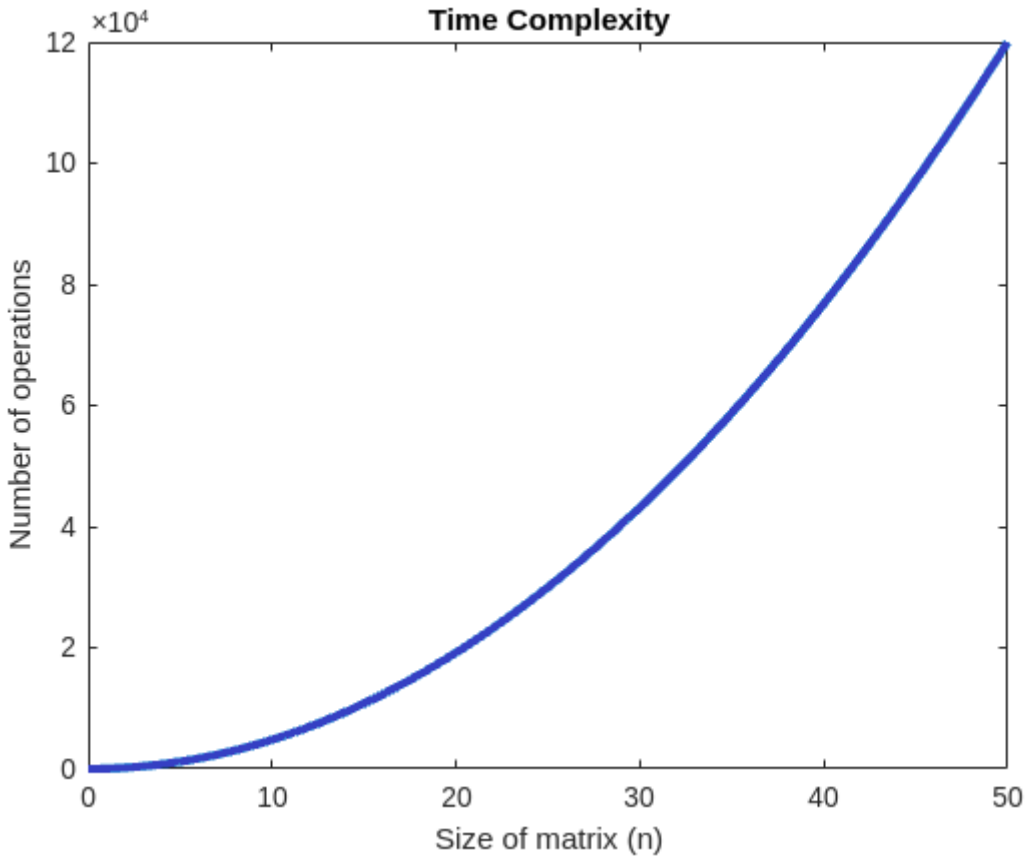
check(1 , 8 , 2 , 10 , 81)
mx=1 my=9
check(1 , 8 , 1 , 9 , 81)
lx==rx   so return false
check(1 , 8 , 2 , 9 , 81)
mat[2][9]==key   so return true
check(1 , 9 , 1 , 10 , 81)
lx==rx   so return false
check(1 , 9 , 2 , 10 , 81)
mat[2][10]!=key   so return false

TIME COMPLEXITY

Since the number of opertion in one function call is constant. So, time complexity of the code is propotional to number of function calls made during entire code.
In the worst case all elements in the matrix are equal.
In that case the number of function calls will be
(n*n/1) + (n*n/4) + (n*n/16) + …. + 1 = n*n*4/3

So the number of function calls are proportional to n^2
Thus **time complexity = O (n^2)**

Graph for Time Complexity

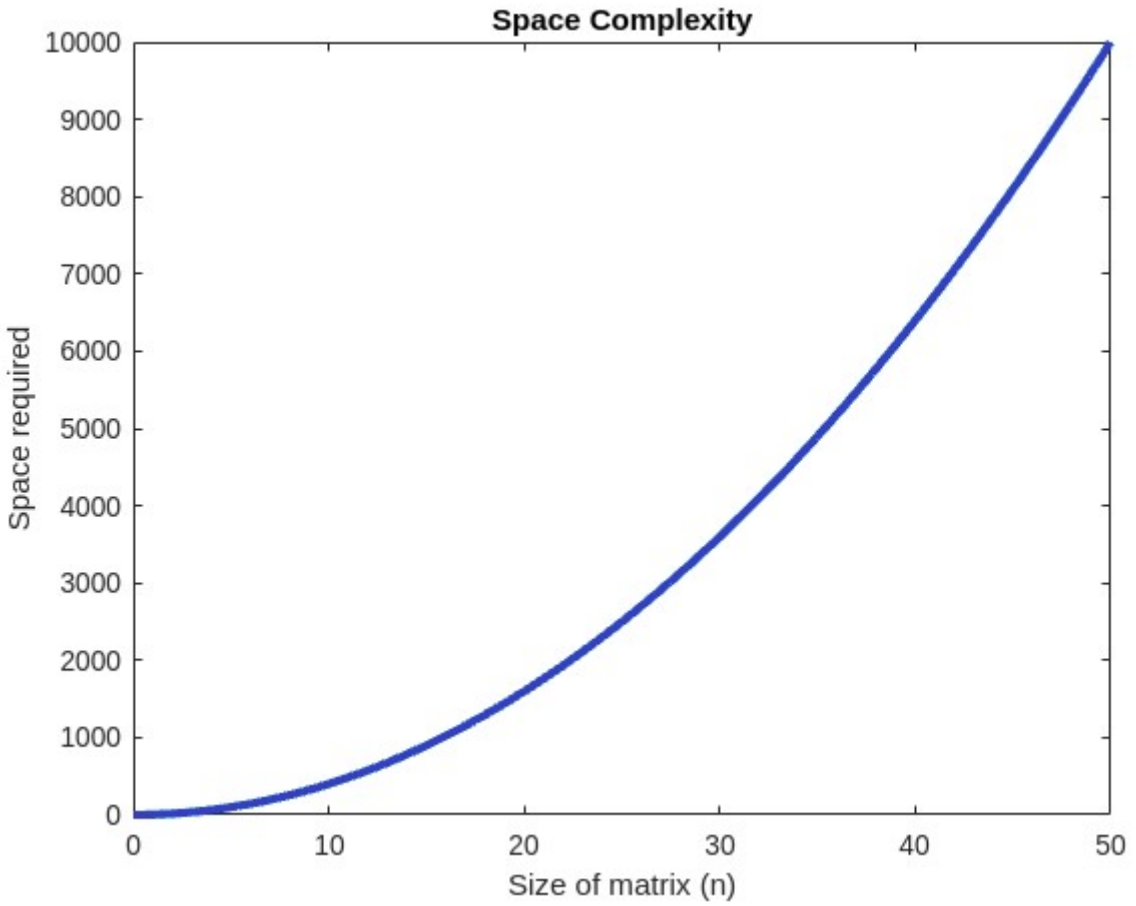


SPACE COMPLEXITY

Since the number of variables used in one function call is constant. So, space complexity of the code is propotional to number of function calls made during entire code.
In the worst case the number of function calls will be
(n*n/1) + (n*n/4) + (n*n/16) + …. + 1 = n*n*4/3

So the number of function calls are proportional to n^2
Thus **space complexity = O (n^2)**

Graph for Space Complexity



CONCLUSION

We have used a divide and conquer approach with the motive to reduce the time and space complexity of the algorithm. But we find that despite using divide and conquer approach our complexity is similar to a version using brute force for solving the same problem. Also in some cases the brute force version can give in fact a better time complexity with respect to divide and conquer approach.

Their are also some cases in which the divide and conquer approach will give a whopping time complexity of Log(n) but in the average case time complexity of divide and conquer is a constant times greater than that of brute force.

So we can conclude that divide and conquer approach in this problem is a failure as brute force approach is a better approach than divide and conquer.