Design And Analysis Of Algorithms-Assignment 3 Group-2

ADITYA RAJ(IIT2017005) MAYANK MRINAL(IIT2017006)
MOHIT PAHUJA(IIT2017007) ARASHPREET SINGH (IIT2017008)

***Abstract*— This document is complete design and analysis of algorithm of creating a matrix of size 50*50 of numbers ranging from 0 to 9 and finding the length of largest sorted component reverse diagonally.**

I. INTRODUCTION

In this problem we have to generate 50*50 matrix and find the length of largest sorted component. For generating random numbers between 0 to 9 we are using random function present in c++ library . For finding largest sorted component we are using l_i and l_d , for ascending and descending respectively , because sorted component can be either in ascending or in descending order. Our algorithm is running in order of n^2.

This report further contains::
II. Algorithm Design.
III.Algorithm Analysis
IV.Experimental Study.
V.Conclusions.
VI.References.

II. ALGORITHM DESIGN

A. Algorithm 1

- In this we have first created a 50*50 matrix using 2D array and filled it with random numbers between 0 to n.
- For generating random numbers we are using the present random function in c++ library.
- We have taken variable $maxl$ which will store the length of largest sorted component , initiated with value of minimum of int.
- Since we have to move reverse diagonal we iterate reverse diagonally and check whether a given number of particular block is greater/less than or equal to previous diagonal in diagonal fashion.
- For above process we are using two for loops.

Algorithm 1

Input:(A[50][50] , n)

$maxl \leftarrow 2$
for `<k=0;k<=n-1;++k>` **do**
  $i \leftarrow k$
  $j \leftarrow 0$
  $l_i \leftarrow 1$
  $l_d \leftarrow 1$
  while $i >= 0$ **do**
    if $j! = 0$ **then**
      if a[j][i]>a[j-1][i] **then**
        $l_i \leftarrow l_i + 1$
        $maxl \leftarrow max(maxl, l_i)$
        $l_d \leftarrow 1$
      else if a[j][i]<a[j-1][i+1] **then**
        $l_d \leftarrow l_d + 1$
        $maxl \leftarrow max(maxl, l_d)$
        $l_i \leftarrow 1$
      else
        $l_i \leftarrow l_i + 1$
        $l_d \leftarrow l_d + 1$
        $maxl \leftarrow max(maxl, l_i)$
        $maxl \leftarrow max(maxl, l_d)$
      end if
    end if
    $i \leftarrow i + 1$
    $j \leftarrow j + 1$
  end while
end for
for `<k=1;k<=n-1;++k>` **do**
  $i \leftarrow n - 1$
  $j \leftarrow k$
  $l_i \leftarrow 1$
  $l_d \leftarrow 1$
  while $j <= n - 1$ **do**
    if $i! = n - 1$ **then**
      if a[j][i]>a[j-1][i+1] **then**
        $l_i \leftarrow l_i + 1$
        $maxl \leftarrow max(maxl, l_i)$
        $l_d \leftarrow 1$
      else if a[j][i]<a[j-1][i+1] **then**
        $l_d \leftarrow l_d + 1$
        $maxl \leftarrow max(maxl, l_d)$
        $l_i \leftarrow 1$
      else
        $l_i \leftarrow l_i + 1$
        $l_d \leftarrow l_d + 1$
        $maxl \leftarrow max(maxl, l_i)$
        $maxl \leftarrow max(maxl, l_d)$
      end if
    end if
    $i \leftarrow i - 1$
    $j \leftarrow j + 1$
  end while
end for

output : $maxl$

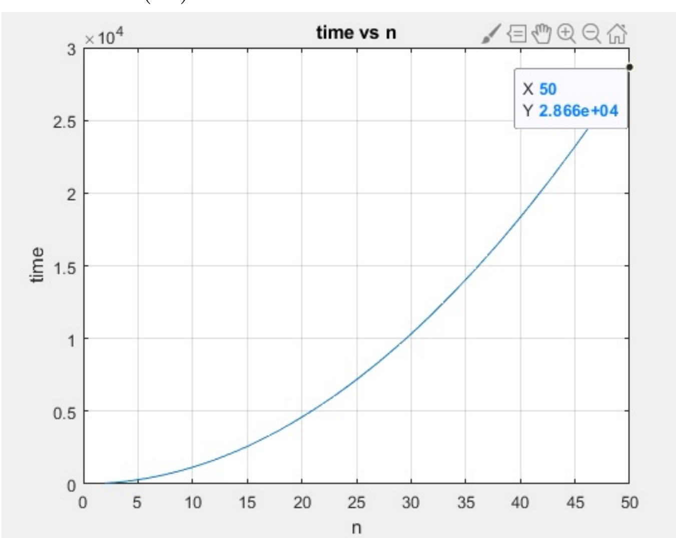
III. ALGORITHM ANALYSIS

A. Time Complexity

- We have calculated the time consumed in each step in terms of number of computations,as given in the above pseudo code.
- In question we have been restricted to generate a 50*50 matrix so to traverse the matrix in reverse diagonal time complexity we be in the order of 50^2.
- Thus for general value of size of matrix let say n , the time complexity is going to be in order of n^2.
- The best case , worst case and average case time complexity will be same i.e. $O(n^2)$.

B. Space Complexity

- Since the number of memory location depends on the size of the square matrix , let the variable size of square matrix be n ,therefore the space complexity will be of order $O(n^2)$.



Thus the graph depicts, our calculation, that is the algorithm takes polynomial of order n^2..

IV. EXPERIMENTAL STUDY

The graph depicts that the plot between the size of matrix i.e. n and time which comes out to be approximately $11.375n^2 + 1.375n - 0.25$.

The integrated Development environment of C++ is used for processing the algorithm and graphical analysis is done using MATLAB plot function.

V. CONCLUSIONS

In this document we conclude that our algorithm has a polynomial time complexity with order of n^2 for all the cases i.e. best,worst and average case . The space complexity is also of the order n^2(where n is the size of matrix.)
Therefore both time and space complexity is $O(n^2)$.

VI. REFERENCES

1.https://www.geeksforgeeks.org/zigzag-or-diagonal-traversal-of-matrix/

2.[CLR96] Thomas H. Cormen, Charles E. Leiserson, and Ronald L Rivest. Introduction to algorithms. The MIT press, 2nd edition, 1996.

3.[DW96] Nell Dale and Henry M. Walker. Abstract data types: specifications, implementations, and applications. D. C. Heath and Company, Lexington, MA, USA, 1996.