Length of largest sorted component reversed horizontally in a matrix

DAA ASSIGNMENT-2 , GROUP 6

Hritik Sharma
IIT2019020

Biswajeet Das
IIT2019019

Shreyansh Patidar
IIT2019018

***Abstract*—This Paper contains the algorithm to create a matrix of size 50 x 50 of numbers ranging from 0 to 9 and to find the length of the largest sorted component reversed horizontally. Two approaches have been taken and we will see the difference in complexity between both.**

I. Introduction

Let's first formally define what $Subsequence$ and sorting are.
Sorting refers to arranging data in a particular format. Sorting algorithm specifies the way to arrange data in a particular order. Most common orders are in numerical or lexicographical order. A subsequence is a sequence contained in or forming part of another sequence.

II. Algorithmic Design

A. *Approach 1*

1. Assign values to a 2 D array of desired length using random function (n=50 in problem),

2. Iterate over a loop through the entire 2D array row wise.

3. In each row , use a $dynamic\_programming\_approach$ to obtain its largest sorted sequence length.

4. Compute optimized LIS values in bottom up manner for each row.

5. For each row , store the value of its longest sorted sequence in an array row_wise_max[n].

6. Print the maximum of all numbers in row_wise_max[n] array.

B. *Approach 2*

1. Traverse 2D array row-wise.

2. Make a new arr[] array and assign value a[0][n-1] to arr[0] for each row i . Now using pointer to arr[] elements iterate remaining array a[0][j] row wise , if the next element in a[0][j] is greater than the last element of arr[] then insert this element into arr[] else replace this element in place of element in a[0][j] which is just greater than or equal to that element.

3. Insertion here will be based on $binary\_search$ technique(divide and conquer) and simple comparison.

4. Store the length of the longest sorted sequence of each row in the row_wise_max[] array.

5.For each row , store the value of its longest sorted sequence in an array row_wise_max[n].

6. The maximum of all elements in the row_wise_max[] array would be the answer.

Algorithm 1: Longest Sorted Subsequence Horizontally

```
Input: Array of size nxn
Output: Length Of LIS
Function LISequence(A,nxn):
    array a , LIS
    for i ← 0 to n − 1 do
        LIS[n] = 1
        for j ← n − 2 to 0 do
            for k ← n − 1 to j do
                if (a[i][j] > a[i][k] && LIS[j] <
                 LIS[k]+1) then
                    LIS[j]=LIS[k]+1;
        Row_wise_max[i] = maximum(LIS[n]);
    Ans = maximum(row_wise_max[n]);
```

Algorithm 2: Longest Sorted Subsequence Horizontally

```
Input: Array of size nxn
Output: Length Of Longest Sorted Sequence
Function DynamicProgramming(a[][],n):
    for i ← 0 to n − 1 do
        pntr=0,arr[0] = a[i][n-1];
        for j ← n − 2 to 0 do
            if a[i][j] >= arr[pntr] then
                arr[++pntr]=a[i][j] ;
            else
                Index = search(arr,0,pntr,a[i][j]);
                arr[index] = a[i][j];
        Row_wise_max[i] = pntr + 1;
    Ans = maximum(row_wise_max[n]);
```

III. Algorithm Analysis

A. *Approach 1*

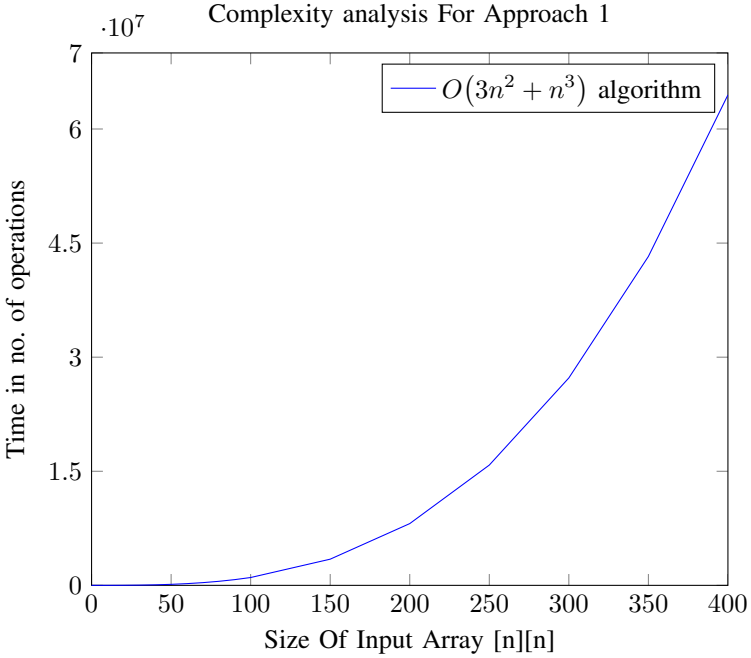
For each row, the DP approach for LIS takes time $\propto n^2$. for each row , we need extra time $\propto n^2$ to calculate the maximum element in LIS.
Here the maximum value of $n = 100$. So the time complexity will be
$O\left(n\left(n + 2 \cdot n \cdot \frac{n+1}{2} + n\right)\right) = O(3n^2 + n^3)$
t_{best}: when $n = 0, = O(0) = 0ms$
t_{worst}: when $n = 100$, t_{worst} = $O(2.03 \cdot (10^6))$

B. *Approach 2*

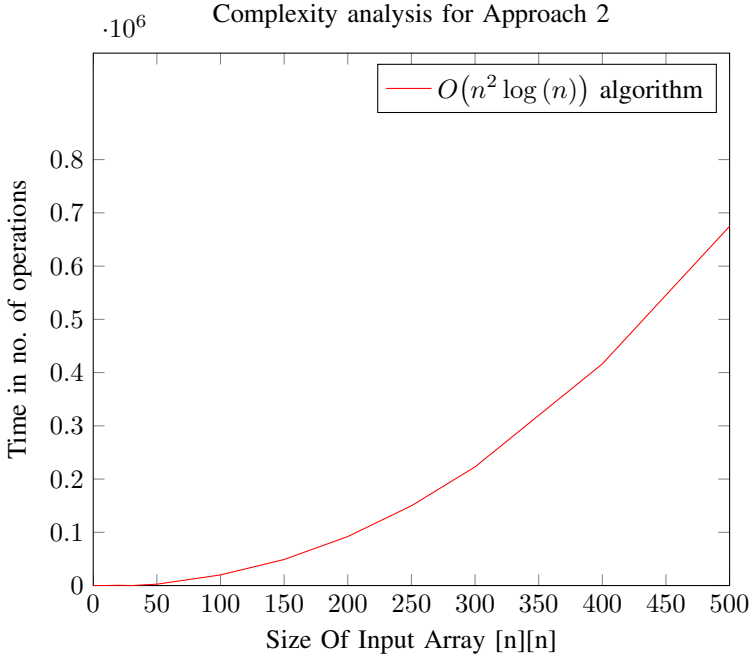
Here, traversing over each row with size n will take time $\propto n$. Finding pos for each element will take maximum log(n) time. And insertion operation will take time n. So, the time complexity will be $O(n^2 \log(n))$
t_{best}: when $n = 0, = O(0) = 0ms$
t_{worst}: when $n = 100$, t_{worst} = $O(2 \cdot 10^4)$

IV. Experimental Study

Complexity analysis For Approach 1



Complexity analysis for Approach 2



V. Conclusion

Above two methods have different time complexities and meet to fulfill the problem statement. The order in which they are good can be listed as:
I. Approach 2
II. Approach 1
Based on the time complexities.

VI. References

1).https//en.wikipedia.org/wiki/Sequence
2).https://www.geeksforgeeks.org/longest-increasing-subsequence-dp-3/
3).https://www2.cs.duke.edu/courses/spring18/compsci330/Notes/dynamic.p