To create a matrix of size 50×50 of numbers ranging from 0 to 9. Find the length of the largest sorted component horizontally.

Akash Anand, Parth Kataria, Shruti Nanda
iit2019015@iiita.ac.in, iit20190016@iiita.ac.in, iit2019017@iiita.ac.in
IV semester,Department of Information Technology
Indian Institute of Information Technology, Allahabad

Abstract-This paper contains the algorithms to find the length of the largest sorted component horizontally in a 2-D matrix. We will discuss time complexities of all algorithms and differences in them.

1. INTRODUCTION

Given a 2d array with n number of rows and m number of columns where n and m less than 50. The elements of the array will range from 0 to 9. We will print the largest sorted component in the array horizontally.
This report further contains:

1. Algorithm Design and Analysis
2. Experimental study and complexity.
3. Conclusion

2. ALGORITHM DESIGN

We designed three algorithms to find the largest sorted array horizontally in 2-D matrix. Basic steps for designing the algorithm are-

1. Create a 2-d array of size(m*n), where (m,n<50)
2. Take the elements of array as user input.
3. Now pass the array along with the size as arguments in the functions of following algorithms to print the length of the largest sorted array horizontally.

Algorithm 1-

```
int Brute(int a[][], int n, int m)
{
   int max_cnt=0,cnt=1;
      for(i =0 to  n){
for( j=1 to j=m)
         {
            if(a[i][j]>a[i][j-1])
               cnt++;
           else{
            max_cnt=max(cnt,max_cnt);
            cnt=1;}
         }
Return max_cnt;
}
```

Analysis: In this algorithm we have checked for each row if the array is increasing or decreasing we have increased the cnt variable and else we have updated the max_cnt variable and make the cnt variable 0.

Algorithm 2-

```
int sorting_cnt(int a[][], int n, int m)
{
   int n, m;
   int max_cnt=0,cnt=1;
   vector<int>count;
   for( i=0 to i=n){
   for( j=1 to m)
   {
     if(a[i][j]>a[i][j-1]){
          cnt++;
          v.push_back(cnt);
    }
   else
          cnt=1;
   }}
   sort(count.begin(), count.end());
          Return count[count.size()-1];
```

}
Analysis:
In this algorithm we are storing the length of the sorted subarrays of the array in a vector count using nested for loop . Then we are sorting it printing the largest length of the sorted subsequence.

3. EXPERIMENTAL STUDY AND ANALYSIS

A.brute()

The Number of iteration in first for loop is the no. of rows present in the array and in the second for loop, number of iteration is equal to the number of columns in array. So, the complexity will be O(n*m).
t $_{\text{worst}}$: O(n+n*m).

B.sorting_cnt()
In this algorithm we are storing the length of each sorted subsequence in the complexity of o(n^2) using a for loop. Then we are sorting it in the complexity of O(n^2logn). So, the overall complexity will be:
t $_{\text{worst}}$: O(n^2logn).

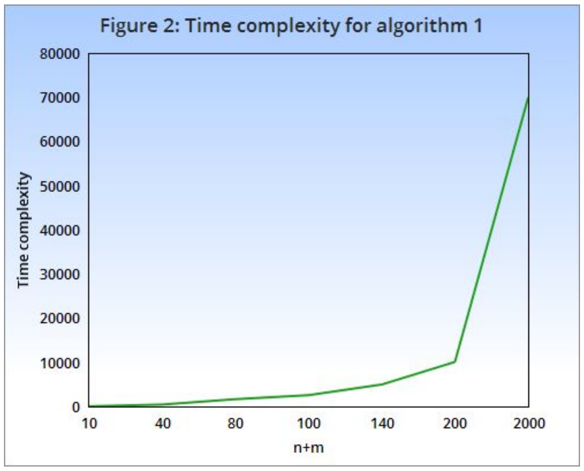
S no.	N	M	Algorithm1	Algorithm2
1	5	5	30	25.219280
2	20	20	420	800.87712
3	40	40	1640	1678.75424
4	50	50	2550	6687.38561
5	70	70	4970	10067.9962
6	100	100	10100	43000.7712
7	1000	1000	1001000	600000.568
8	1000	10000	10001000	70000000

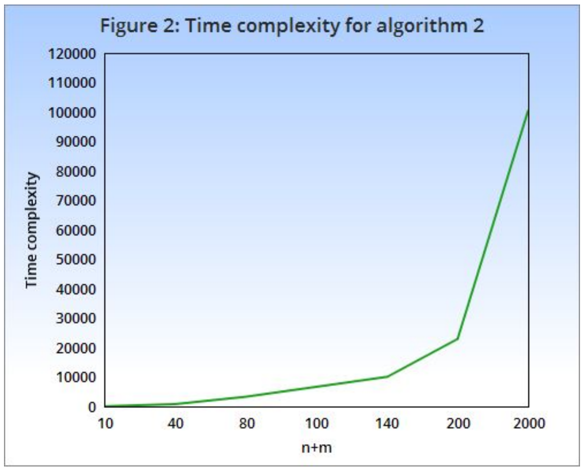
4. TIME COMPLEXITY

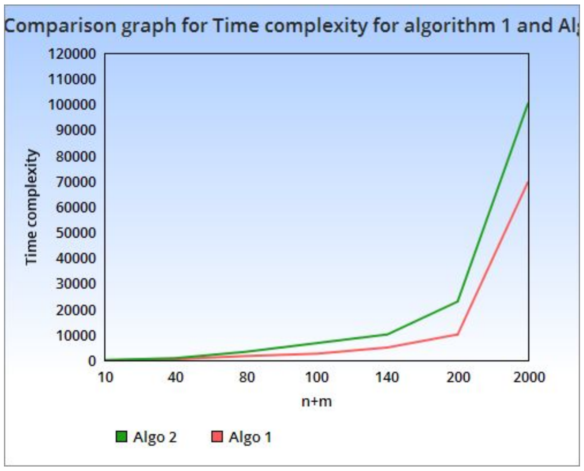
The algorithms were tested against positive random sets of variable sizes.The result thus obtained from this experiment is given below:

For brute():
Best case : Ω(n^2)
Average case : θ(n^2)
Worst case : O(n^2)

For sorting_cnt():
Best case : Ω(n^2)
Average case : θ(n^2.logn)
Worst case : O(n^2logn)







5.CONCLUSION

Time complexity for finding the length of the largest sorted subarray will be minimum in brute force which is O(n^2) and for sorting count algorithm time complexity will be O(n^2logn) for worst case and for best case it will be o(n^2).

6. REFERENCES

1. https://www.quora.com/How-would-one-use-Arrays-sort-on-a-multidimensional-array-of-ints-by-the-first-element-of-each-sub-array-in-Java
2. https://www.geeksforgeeks.org/longest-increasing-path-matrix/