Design and Analysis of Algorithms Assignment - 4

Department of Information Technology,

Indian Institute of Information Technology, Allahabad 211015, India

Ankush Sonker(IIT2019072), Aditya Rathi(IIT2019073), Sanidhya Gupta(IIT2019074)

Abstract: ***Given an array of n points in a plane. The problem is to find out the closest pair of points in an array. In this paper, this trivial problem of finding the pair of closest points in the given 2D plane has been solved using the techniques and ideas based on Divide and Conquer Programming Paradigm. Also, this paper discuss the distance between these pair of closest points also. Using the Divide and Conquer approach, the algorithm achieves its best time complexity of O(nlogn),thus reducing the time to a much significant extent. Thus the approach of Divide and Conquer exhibits the logarithmic time and speeds up the mechanism of finding the closest pair of points.***

Index Terms: ***Arrays, Divide and Conquer, Merge Sort, Sorting and Points***

INTRODUCTION

We have been given a set (or array) of points in 2D plane. In computational world, an Array is a "Linear data structures which stores elements of similar kind (same data types) in contiguous location in memory. Therefore it makes the accession of elements in constant time". In this paper sorting is another significant paradigm, which finds the prime importance. We have used the concept of Merge Sort to sort the points in order to be used efficiently and reduce the time complexity while finding the closest pair of points using the Divide and Conquer Approach.

Divide and Conquer Divide and Conquer is a programming paradigm based on the concept of dividing then merging. We may say it to be sometimes a greedy approach of finding the solution in the most speedy manner. A divide-and-conquer algorithm recursively breaks down a problem into two or more sub-problems of the same or related type, until these 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 technique is the basis of efficient algorithms for many problems, such as sorting (e.g., quicksort, merge sort), multiplying large numbers (e.g., the Karatsuba algorithm),finding the closest pair of points. The approach of divide and conquer generally follows three important aspects:

1. **Divide:** This involves dividing the problem into some sub problem.
2. **Conquer:** Sub problem by calling recursively until sub problem solved.
3. **Combine:** The Sub problem Solved so that we will get find problem solution.Let us have a pictorial look of the image which makes the concept of Divide and Conquer Paradigm self explanatory.

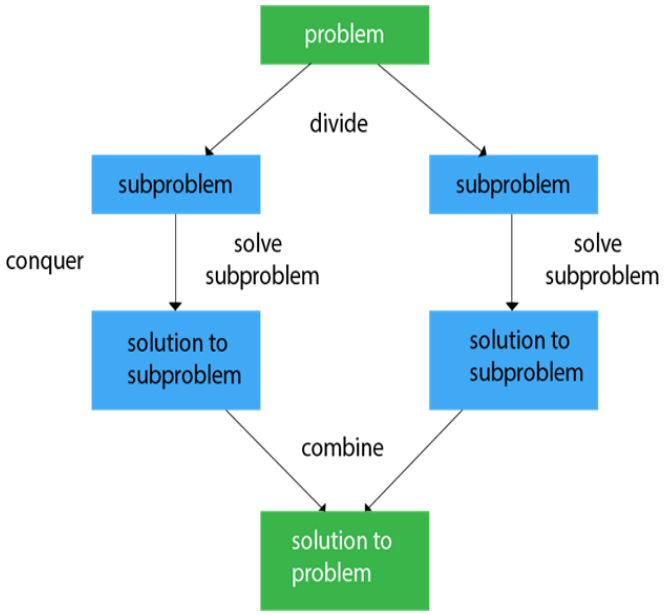

Figure 1: Divide and Conquer

Advantages of Divide and Conquer The Divide and Conquer technique generally has following mentioned advantages over Brute Force Algorithms.

Time Efficiency This approach generally reduces the running time of the algorithm because the running time of algorithm based on Brute Force is in general high order in nature. But when we are using the approach based on Divide and Conquer, this running time generally decreases and becomes logarithmic in nature. This is because the approach of Divide and Conquer keeps on dividing which makes it logarithmic in nature.

Space Efficiency: As no need to mention the algorithm, but still, the space complexity also reduces up to a much extent due to the nature of dividing and then merging up. Also it is worthy to note that some divide and conquer algorithms which are based on the recursive algorithms generally tend to occupy more space in internal allotted stack of the process in RAM(Random Access Memory).

Note that, we have also used the distance formula to calculate the distance between the points. The step of dividing the plane recursively and the finding the closest pair using the conquer technique is used in this approach. The time complexity of the algorithm based on Divide and Conquer paradigm is O(nlogn) and the space complexity is of the nature O(n), since we require to store all the points of the given 2D plane(or given as an input).

This report further contains:

- Algorithm Designs
- Algorithm Analysis
- Experimental Study and Profiling
- Conclusion
- References
- Appendix

ALGORITHM DESIGN

A general problem based on the divide and conquer approach generally has the branching them conquering. In other words, we divide or split the problem into sub tasks and then make decisions while capturing the answer and merging. So this problem of finding the pair of closest points is no exception. The data structure used in finding the pair of closest points are the standard and primary data structure of the computational domain. Following structures have been taken into account while making the code for the report.

Data Structures Used:

- Structure to store Point
- Array to store the set of Points

Algorithmic Steps: Basically, the approach based on Divide and Conquer paradigm of finding the closest pair points generally has the steps defined in following algorithmic procedure, which have been implemented while solving the problem:

1. Input points using random number generator function.
2. Create a function distance which will calculate the distance between the points using standard distance formula in analytical geometry.
3. Create a function sortX, which will sort points according to x-coordinates.
4. In a similar manner create a function sortY, which will sort the points according to y-coordinate.
5. Divide all points in two halves.
6. Recursively find the smallest distances in each of the above two divided planes.
7. Take minimum of these two smallest distances and store it in a variable say d.
8. Create an array strip[ ], hat stores all points which are at most d distance away from the middle line dividing the two sets.
9. Find the smallest distance in strip[ ] array and store it in variable say dis.
10. Return the min(d,dis).

The above returned distance is the minimum distance between any pair of points in given 2D plane.

```
declare : point1 point 2

Double:
Function dis(P1, P2)
    return sqrt(P1.x–P2.x)–(P1.y–P2.y))

Double:
Function solveStrip(Strip_Array, int n, double d)
        min = d

        sort(Strip_Array)

        loop i=0 to n with i++
            loop j=i+1 to n with ++j
                if dis(strip[i],strip[j]) < min
                   min = dis(strip[i], strip[j])
                   ans[0]=strip[i]
                   ans[1]=strip[j]
        return min

Double:
Function solve(Points_Array, int start, int end)
        if end - start <= 3
        double min = 1e9
        loop i=start to end with i++
            loop j=i+1 to end with ++j
            if dis(points[i], points[j]) < min
                min = dis(points[i], points[j])
                ans[0]=points[i]
                ans[1]=points[j]
                return min

        int mid = (start+end)/2

        Point midPoint = points[mid]
        double x = solve(points, start, mid)
        double y = solve(points, mid+1, end)
        double d = min(x,y)

Int:
```

```
Function main()
    int n
    Input n

    points :array

    loop i=0 to n with i++
        Input Points[i]

    sort(points)

    print The smallest distance is
    print solve(points, 0, n)
    print The corresponding pair of points is
    print point 1 point 2
    return 0
```

ALGORITHM ANALYSIS

APRIORI ANALYSIS: We have tried to do the Apriori Analysis of the approach based on Divide and Conquer.

Let T(n) and S(n) is the time and space respectively with input parameters defined above.

TIME COMPLEXITY DERIVATION: Let Time complexity of above algorithm be T(n). Let us assume that we use a O(nlogn) sorting algorithm. The above algorithm divides all points in two sets and recursively calls for two sets. After dividing, it finds the strip in O(n) time. Also, it takes O(n) time to divide the Py array around the mid vertical line. Finally finds the closest points in strip in O(n) time. So T(n) can be expressed as follows:

TIME COMPLEXITY DERIVATION:

$T(n) = 2T(n/2) + O(n) + O(n) + O(n)$
$T(n) = 2T(n/2) + O(n)$

Using above relation, we get for T/2, T/8 etc as:
$T(n/2) = 2T(n/4) + O(n/2)$
$T(n/4) = 2T(n/8) + O(n/4)$
$T(n/8) = 2T(n/16) + O(n/8)$ *and so on.....*
.
.
.
.
Thus on combining we get the overall time complexity as:$T(n) = T(nlogn)$

DIFFERENT CASES: Now let us consider our algorithm in different scenarios.

BEST CASE: The time complexity of Divide and Conquer approach in all the three cases is same , i.e, O(nlogn). Space complexity is the O(n) because we need to store all the points.

AVERAGE CASE: The time complexity of Divide and Conquer approach in all the three cases is same , i.e, O(nlogn). Space complexity is the O(n) because we need to store all the points.

WORST CASE: The time complexity of Divide and Conquer approach in all the three cases is same , i.e, O(nlogn). Space complexity is the O(n) because we need to store all the points.

PROFILING

So, after the above analysis of Apriori Analysis, we come to the Posteriori Analysis or Profiling. Now let us have the glimpse of space and time graph and then comparison between both the approaches as a follow-up.

TIME ANALYSIS: Following is the graph representing the time complexity of the algorithm.

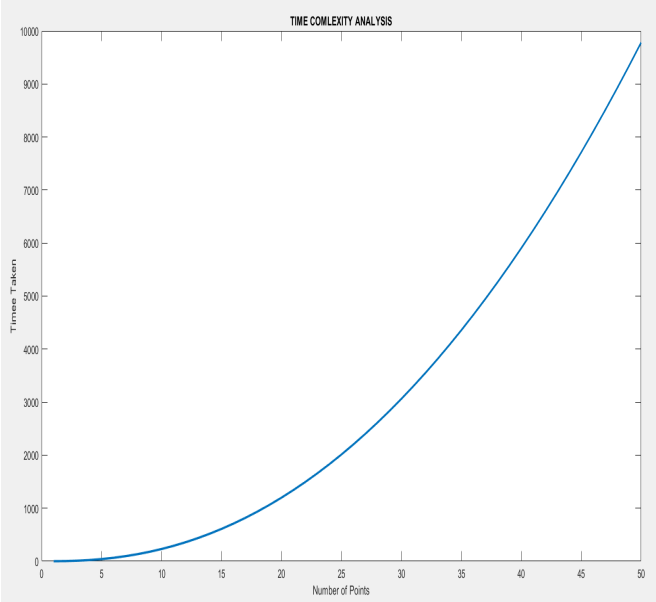

Figure 2: Time Complexity Graph

By the experimental analysis, we found that in case of optimized approach, on increasing the no. of points in the set (increasing the size of set) the graph is strictly increasing with a bend (or concavity) towards horizontal axis. Thus the overall time increases with an increase in no. of points.

SPACE ANALYSIS: Following is the graph representing the space complexity of the algorithm.

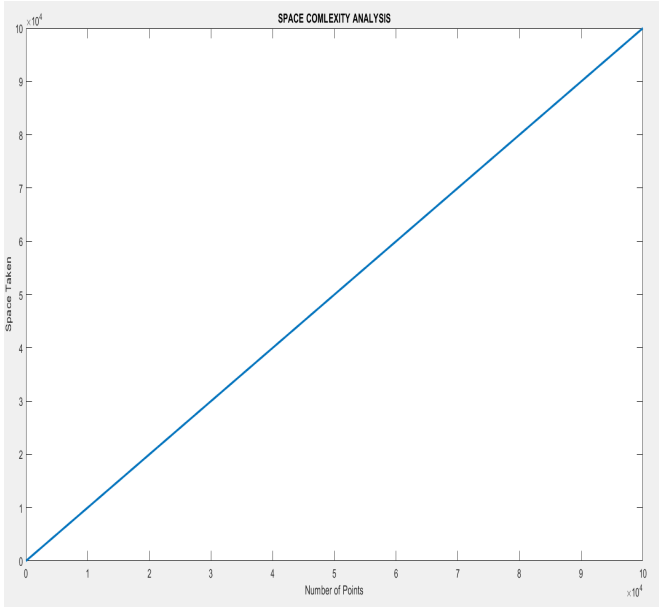

Figure 3: Space Complexity Graph

By the experimental analysis, we found that in case of optimized approach, on increasing the no. of points in the set (increasing the size of set) the graph is strictly increasing Thus the overall space increases with an increase in no. of points

APPLICATIONS

Divide and Conquer is a wide variety of algorithmic paradigmn which beleives on the strategy of dividing the task and then applying an internal mechanism or algorithm in order to get the required answer. This programming paradigmn has several predefined algorithms which work on the basis of Divide and Conquer approach. Some of the applications of the Divide and Conquer Approach are:

1. **Binary Search:** It is a powerful programming algorithm based on the divide and conquer paradigmn which works very efficiently and gives answer in generally time cost, which is in general logarithmic in nature.

2. **Segment Trees:** It is another data structure which uses three standard operations:insert,query and update. All these three operations are based on divide and conquer because for example for building a segment tree, we first need to create in a recursive bottom up divide and conquer based technique.

3. **Strassen's Algorithm:** It is an efficient algorithm to multiply two matrices. A simple method to multiply two matrices need 3 nested loops. Strassen's algorithm multiplies two matrices in efficient time.

4. **Quick Sort:** It is a sorting algorithm. The algorithm picks a pivot element, rearranges the array elements in such a way that all elements smaller than the picked pivot element move to left side of pivot, and all greater elements move to right side. Finally, the algorithm recursively sorts the subarrays on left and right of pivot element.

5. **Merge Sort:** is also a sorting algorithm. The algorithm divides the array in two halves, recursively sorts them and finally merges the two sorted halves.

CONCLUSION

So, with the above mentioned algorithms and their profiling, we come to the conclusion that this classical problem of finding the closest pair of points is achieving its best time complexity of O(nlogn) and space complexity of O(n).

Also, when we are dealing with large numbers such as the one occupying 64-bits (long long integers), the complexity is rather same almost whereas a condition of overflow can occur while calculating distances between points. So we can take the modulus of distance with 109+7 in order to make sure that it gets fit in the available data types(long long integer types more specifically). Thus, finding closest pair of points using Divide and Conquer proved to be the most efficient algorithm here.

ACKNOWLEDGMENT

We are very much grateful to our Course instructor Dr Mohammed Javed and our mentor, Md Meraz, who have provided the great opportunity to do this wonderful work on the subject of Data Structure and Algorithm Analysis specifically on the programming paradigm of Divide and Conquer.

REFERENCES

1. Introduction to Divide and Conquer Technique: https://www.geeksforgeeks.org/divide-and-conquer-algorithm-introduction/

2. Introduction to Algorithms by Cormen,Charles, Rivest and Stein.
https://web.ist.utl.pt/ fabio.ferreira/material/asa

APPENDIX

To run the code, follow the following procedure:

1. Download the code(or project zip file) from the github repository.
2. Extract the zip file downloaded above.
3. Open the code with any IDE like Sublime Text, VS Code, Atom or some online compilers like GDB.
4. If required, save the code with your own desirable name and extension is .cpp
5. Run the code following the proper running commands(vary from IDE to IDE)
   (a) **For VS Code:** Press Function+F6 key and provide the input on the terminal.
   (b) **For Sublime Text:** Click on the Run button and provide the input.

Code for Implementation is:

```
#include <bits/stdc++.h>
using namespace std;

pair<int,int> ans[2];

double dist(pair<int,int> p1, pair<int,int> p2)
{
        return (double)sqrt((p1.first-p2.first)*(p1.first-p2.first)+
        (p1.second-p2.second)*(p1.second-p2.second));
}

double solveStrip(vector<pair<int,int>> strip, int n, double d)
{
        double min = d;
        sort(strip.begin(),strip.end());
        for (int i = 0; i < n; ++i)
        {
            for (int j = i+1; j < n && (strip[j].second - strip[i].second) < min; ++j)
            {
                if (dist(strip[i],strip[j]) < min)
                {
                    min = dist(strip[i], strip[j]);
                    ans[0]=strip[i];
                    ans[1]=strip[j];
                }
            }
        }
        return min;
}

double solve(vector<pair<int,int>> points, int start, int end)
{
        if (end - start <= 3)
        {
            double min = 1e9;
            for (int i = start; i < end; ++i)
            {
                for (int j = i+1; j < end; ++j)
                {
                    if (dist(points[i], points[j]) < min)
                    {
                        min = dist(points[i], points[j]);
                        ans[0]=points[i];
                        ans[1]=points[j];
                    }
                }
            }
            return min;
        }
        int mid = (start+end)/2;
        pair<int,int> midPoint = points[mid];
```

```
        double d = min(solve(points, start, mid), solve(points, mid+1, end));
        vector<pair<int,int>> strip;
        for (int i = start; i < end; i++)
        {
            if (abs(points[i].first - midPoint.first) < d)
            {
                strip.push_back(points[i]);
            }
        }
        return min(d, solveStrip(strip, strip.size(), d));
}

int main()
{
    int n;
    cin>>n;
    vector<pair<int,int>> points(n);
    for(int i=0;i<n;i++)
    {
        cin>>points[i].first>>points[i].second;
    }
    sort(points.begin(),points.end());
    cout<<"The smallest distance is "<<solve(points, 0, n)<<endl;
    cout<<"The corresponding pair of points is {"<<ans[0].first<<" "<<ans[0].second<<"} and ";
    cout<<"{"<<ans[1].first<<" "<<ans[1].second<<"}";
    return 0;
}
```