CS263 ASSIGNMENT 5

NAME:

ARCHIT AGRAWAL

ROLL NO. :

202052307

SECTION:

A

1. **In this problem, a set of n points are given on the 2D plane. You have to find the minimum distance pair of points.**
   For example:- P[] = {2, 3}, {12, 30}, {40, 50}, {5, 1}, {12, 10}, {3, 4}.
   Minimum distance is 1.41 between pair of points {2, 3} and {3, 4}.

   Write the brute force and divide and conquer algorithm with a complete analysis. If you find that there is still a scope to reduce the complexity further using the divide and conquer technique, then write the new algorithm and give the complete analysis of complexity.

Let us first solve this problem using **Brute Force** method. In this method, each given point's distance will be computed with every other point given and the minimum distance will be tracked.

Algorithm (Brute Force)

- Declare an integer variable 'min' and initialise it with the maximum integer value (Integer.MAX_VALUE). Declare two variable p1 and p2. These will store the index of minimum pair of points.
- Run a loop from i = 0 to i less than the length of points array.
- Run a loop inside the first loop from j = i + 1 to less than the length of points array.
- Declare x and initialise it with the difference of x co-ordinates at index i and j in the points array.
- Declare y and initialise it with the difference of y co-ordinates at index i and j in the points array.
- Compute (x * x + y * y), if it is less than min, update min with (x * x + y * y), p1 with i and p2 with j.

- Print the points[p1] and point[p2].
- Return square root of min after the loop ends.

CODE

```java
import java.util.*;

class ClosestDistance {

    public static double minimumDistance(int[][] points){
        int min = Integer.MAX_VALUE;
        int p1 = 0;
        int p2 = 0;
        for(int i = 0; i < points.length; i++){
            for(int j = i + 1; j < points.length; j++){
                int x = points[i][0] - points[j][0];
                int y = points[i][1] - points[j][1];

                if(x * x + y * y < min){
                    min = x * x + y * y;
                    p1 = i;
                    p2 = j;
                }
            }
        }

        System.out.print("The points that are at a minimum distance are
: (" + points[p1][0] + "," + points[p1][1] + "), ");
        System.out.print("(" + points[p2][0] + "," + points[p2][1] +
")");
        System.out.println();

        return Math.sqrt(min);
    }

    public static void main (String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.print("Enter the number of points : ");
        int n = sc.nextInt();

        int[][] points = new int[n][2];

        System.out.println();
```

```
        System.out.println("Enter the points, one in each line, with a
space between x and y co-ordinate.");

        for(int i = 0; i < n; i++){
            points[i][0] = sc.nextInt();
            points[i][1] = sc.nextInt();
        }

        System.out.println("The minimum distance between any two points
is : "+minimumDistance(points));
    }
}
```

OUTPUT

```
PS C:\Users\Archit\Desktop\cprog> cd "c:\Users\Archit\Desktop\cprog\" ; if ($?) { javac C
Enter the number of points : 5

Enter the points, one in each line, with a space between x and y co-ordinate.
1 2
3 5
-1 4
5 1
4 4
The minimum distance between any two points is : 1.4142135623730951
PS C:\Users\Archit\Desktop\cprog>
```

```
PS C:\Users\Archit\Desktop\cprog> cd "c:\Users\Archit\Desktop\cprog\" ; if ($?) { javac C
Enter the number of points : 5

Enter the points, one in each line, with a space between x and y co-ordinate.
10 0
0 0
8 7
2 -6
-5 -4
The minimum distance between any two points is : 6.324555320336759
PS C:\Users\Archit\Desktop\cprog>
```

Time Complexity

The steps inside the inner loop takes constant time to execute and hence it take O(1) time. Now, to get the overall time complexity, we need to find how many times the inner loop executes.

When i = 0: the inner loop runs (n – 1) times and i updates to 1.

When i = 1: the inner loop runs (n – 2) times and i updates to 2.

This process keeps repeating until i = (n - 1) and the inner loop runs 0 times in this case.

Therefore, the time function can be represented as:

$$T(n) = (n-1) + (n-2) + \cdots + (1) + O(1)$$

$$T(n) = \frac{n(n-1)}{2} + O(1)$$

$$\therefore T(n) = O(n^2)$$

Now, let us try solving this problem using Divide and Conquer Approach.

Algorithm (Divide & Conquer)

The number of points is n i.e the length of points array is n.

- Firstly, sort the points array based on the x co-ordinates of points.
- Find the middle index in the array and divide the array in two parts. Let us say point[n/2] be the middle point. The first subarray contains points from points[0] to points[n/2] and the other subarray contains points from points[n/2 + 1] to points[n - 1].
- Recursively find the smallest distance in both the subarrays. Let the smallest distance in the left subarray be minLeft and in the right subarray be minRight. Let the minimum of minLeft and minRight be min.
- From the above steps, we have an upper bound of minimum distance i.e min. Now, we need to consider the pairs such that one point is from the left half and the other is in right half. Consider the vertical line passing through points[n/2] and find all the points whose x co-ordinate is closer than ‘min’ to the vertical line. Build an array say ‘crossPoints’ of all such points.
- Sort this crossPoints array according to their y co-ordinates. Find the smallest distance in crossPoints array, and call it minCross.

- Return the minimum of minCross and min. This will be the minimum distance we want to calculate.

Time Complexity

Let us say that the time complexity of this algorithm be $T(n)$. Let us assume that we are using the best sorting algorithm for sorting, so it takes $O(n \log n)$. We divide the array into two halves and recursively call the algorithm on left and right subarray. After dividing it, the crossPoints array is formed in O(n) time as it checks for all the points in the subarray. Then the crossPoints is sorted in $O(n \log n)$ time and finally finds the closest point in the crossPoints array in the strip. This step seems to be done in $O(n^2)$ time but it actually takes O(n) time as there is a need to look for closest distance for a maximum of 7 points in the y co-ordinate sorted crossPoints array. This can be proved geometrically.

$$\therefore T(n) = O(n \log n) + 2T\left(\frac{n}{2}\right) + O(n) + O(n \log n) + O(n)$$

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n \log n)$$

Using the Master's Theorem, we have,

$$\log_b a = 1, k = 1, p = 1$$

$$\therefore \log_b a = k \; and \; p > -1$$

$$\therefore T(n) = O(n \log n^2)$$

Time Complexity Using Recursion Tree:

Points = {{2,3}, {12,30}, {40,50}, {5,1}, {12,10}, {3,4}, {0,0}}

{{0,0}, {2,3}, {3,4}, {5,1}, {12,10}, {12,30}, {40,50}} --------O(n logn)

{{0,0}, {2,3}, {3,4}, {4,1}} ---T(n/2)

//find distance using brute force

//minLeft = 1.414

{{12,10}, {12,30}, {40,50}} ----T(n/2)

//find distance using brute force

//minRight = 20.0

Update min = 1.414

//find the crossPoints array ---------------------O(n)

{{3,4}, {4,1}}

//sort it on basis of y co-ordinate -------------------O(n logn)

{{4,1}, {3,4}}

//check for the minimum distance in this array ------------------O(n)

// minimum = 3.162

Return min(min, minimum)