Design And Analysis Of Algorithms

CSE 224

Spring 2019

Submitted by:

Engy Samy Salah (16P3004)

Gina Emil Attia (16P3022)

Mayar Wessam (16P3008)

Rowan Hazem (16P3023)

Yara Hossam (16P3002)

Ayman Hesham (16P3037)

Introduction:

We are given an array of n points in the plane, and the problem is to find out the closest pair of points in the array. This problem arises in a number of applications. For example, in air-traffic control, you may want to monitor planes that come too close together, since this may indicate a possible collision. Recall the following formula for distance between two points p and q.

$$||pq|| = \sqrt{(p_x - q_x)^2 + (p_y - q_y)^2}$$

The Brute force solution is O(n^2), compute the distance between each pair and return the smallest. Here we calculate the smallest distance in O(nLogn) time using Divide and Conquer strategy. In this project, we discuss an implementation with time complexity as O(nLogn).

Description:

In this project we discuss how to find out the closest pair of points in the array using a divide and conquer algorithm.

The idea is to presort all points according to y coordinates. Let the sorted array be Py[]. When we make recursive calls, we need to divide points of Py[] also according to the vertical line. We can do that by simply processing every point and comparing its x coordinate with x coordinate of middle line.

In our project we use two types of functions to calculate the distance of the closest pair. The 1st one is exhaustive search function that is used only when we have less than 3 points to calculate the distance between the closest pair without using the divide and conquer style. Then we use a function called closestUtil that calculates the distance of the closest pair using the divide and conquer style. We first enter the numbers that are then added to an array of type points. Then we sort the array by the x coordinates. Then we check if the array is less than or equal to three points then we calculate and return the distance between them immediately using a function called bruteForce. Otherwise, if the array is more than three points we divide it into two parts and for each part we call recursively the divide and conquer function(closestUtil).

Complexity:

Time Complexity: 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 array around the mid vertical line. Finally finds the closest points in strip in O(n) time. So T(n) can expressed as follows

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

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

$T(n) = T(nLogn)$

Example:

Step 1:

Number of points is declared by user.

Number of Points (N) = 4.

Step 2:

Create a Struct named Point that has integers x and y. Create an array named P of type Point (plane has two points x and y).

Step 3:

Point 1: 89,66

Point 2: 167,37

Point 3: 69,27

Point 4: 85,162

Step 4:

We call the function **closest** that creates two arrays of type point Px and Py.

Using Function qsort (function declared in C++) to sort the numbers of arrays Px and Py.

Point 1: 69,27

Point 2: 85,162

Point 3: 89,66

Point 4: 167,37

Step 5:

We make assumption that all the x points are distinct as to avoid having similar points.

In this example all the x points are distinct.

Step 6:

We call the divide and conquer function closestUtil and it takes (2 Arrays of type point Px (sorted on X coordinates) and Py (sorted on Y coordinates) and an integer (n=4).

Step 7:

We check if the array have 3 points or less or more. We here have 4 points.

Step 9:

First we calculate the mid index of the array = n/2 = 4/2 = 2.

Creat a midpoint of type Point and assign to it the mid of Px array (89,66).

We then create two arrays of type Point for the array Py to divide it into to half. The 1st Pyl (left) is from the 1st element in Py till the midpoint. The 2nd Pyr (right) is from the element after the midpoint of Py till the end.

Step 10:

We then call the divide and conquer function recursively twice.

The 1st time is till the mid. Using Px and Pyl.

Step 11 (into the function):

We find that we have three points only in the array (Pyl) The 1st Point (69, 27), the 2nd Point (89,66) and the 3rd Point (85,162).

We then call the bruteForce function that calculates the distance between the two or three points.

BruteForce (Px,n=mid=2)

The minimum distance between the 1s two points = dl=135.945.

The 2nd time is from the mid+1 to high. Using Px and Pyr.

Step 12 (into the function):

We find that we have only two points in the array Point (167,37) and the mid-point (89,66).

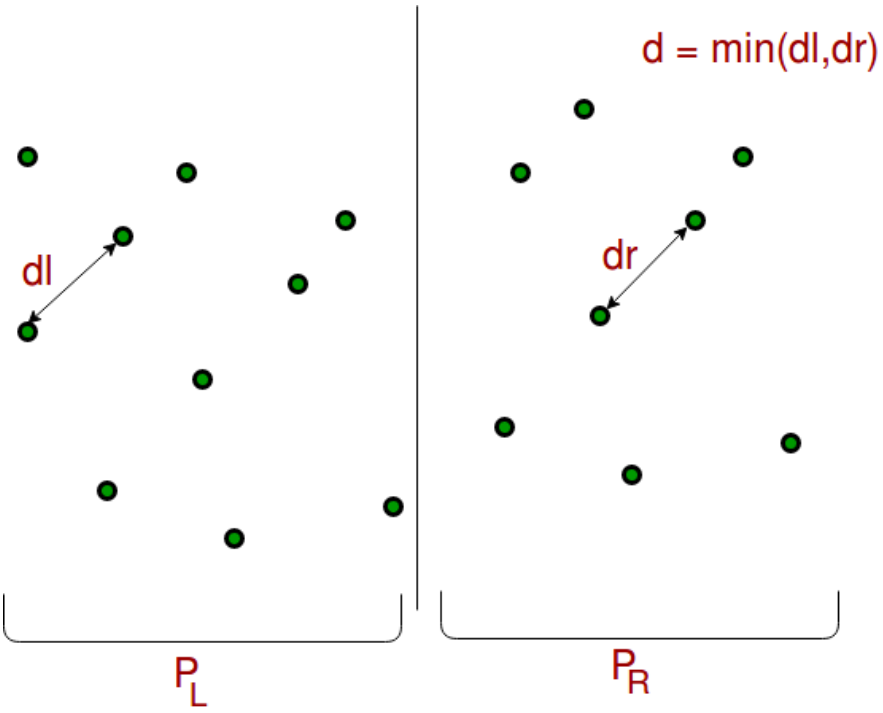
We then call the distance function that calculates the distance between the two points

The minimum distance between the two points =dr= 83.21658.

Step 13:

We use the min function to get the minimum between the minimum two distances.

d= 83.21658.



Step 14:

We build an array strip[] that contains points close (closer than d) to the line passing through the middle point. Then we find the closest points in strip and finally return the minimum of d and closest distance in strip[].

Then The distance between the closest pair is = 43.8292.

Output:

```
N= 4
The points are:
89,66
167,37
69,27
85,162
The smallest distance is 43.8292
Process returned 0 (0x0)   execution time : 2.920 s
Press any key to continue.
```

```
int main()
{
    Point P[] = {{89, 66}, {167, 37}, {69, 27}, {85, 162}};
    int n = sizeof(P) / sizeof(P[0]);
    cout << "N= " << n <<endl;
    cout << "The smallest distance is " << closest(P, n);
    return 0;
}
```