Bangladesh University of Engineering and Technology

Course No: CSE 204

Course Title: Data Structure and Algorithm – 1

Offline Assignment – 8

Finding 2nd Nearest Pair using Divide & Conquer Algorithm

Name: Md. Azizur Rahman Anik

Student Id: 1805115

Department: CSE

Section: B2

Date of Submission: 18-06-2021

Problem Statement: The given problem is to find 2nd nearest pair of houses using divide and conquer algorithm in O(nlogn).

Analysis:

In the program, the problem is solved using the following methods:

i. secondMinimum(Point[] points) => T(n)=O(nlogn)+O(nlogn) = O(nlogn)
ii. findSecondMinimum(Point[] Px, int start, int end) => T(n) = O(nlogn)
iii. baseCaseHandler(Point[] Px, int start, int end) => T(n) = Θ(1)
iv. minFinder(SecondNearestPair p, SecondNearestPair q) => T(n) = Θ(1)
v. secondNearestStrip(Point[] strip, int size, SecondNearestPair ans) => T(n) = O(n)

Now, here the divide and conquer algorithm is mainly used in findSecondMinimum() method and the rest of the methods are auxiliary methods for findSecondMinimum().

In method (i), at first the points are being sorted with respect to x co-ordinates using merge sort and it takes **O(nlogn) time----------(1)**. Then it calls method (ii) for the rest of the job.

In method (ii), the following snippet code takes **O(n) time----------(2)** which is used to create the intervals for the left and right sub problems.

```
56          for(int i=start; i<end; i++)
57          {
58              if((Double.compare(Px[i].x, midPoint.x) <=0) && left < mid)
59              {
60                  leftIntervalPoints[left++] = Px[i];
61              }else
62              {
63                  rightIntervalPoints[right++] = Px[i];
64              }
65          }
```

Next, the algorithm has 2 recursive function calls having n/2 inputs. Therefore the time complexity is **T(n/2) + T(n/2) = 2T(n/2)----------(3).** The snipped code is given below.

```
66  SecondNearestPair leftInterval = findSecondMinimum(leftIntervalPoints, start: 0, left);    //recursively calling for left sub problem
67  SecondNearestPair rightInterval = findSecondMinimum(rightIntervalPoints, start: 0, right); //recursively calling for right sub problem
68
```

In order to solve the problem with points from both sides of the vertical line which may have the second minimum distance, we have sorted the Px points with respect to y coordinates. As this is inside the sub problem which is equally being divided every time, therefore, we're calling only the **"merge()"** method of the merge sort. Therefore the time complexity for this part becomes **O(n)----------(4).**

```
74      //Now we are merging the Px (which is already sorted with respect to x coordinate) with respect to y coordinate so that we can create the strip
75      //Strip is the pair of points where one point is from the left interval and the other from the right interval and there may have a minimum and 2nd minimum distance among the pairs.
76      MergeSort.merge(Px, start, mid, r: end-1, compare: "y");
77
```

Finally, creating the required strip and the method to solve it takes **O(n)+O(n) = 2O(n) time----------(5).** The snipped code is given below:

```
        Point[] strip = new Point[n];
        int k=0;
        for(int i=start; i<end; i++)
        {
            if(Double.compare(Math.abs(Px[i].x - midPoint.x) , ans.secondMinDistance) <=0)
                strip[k++] = Px[i];
        }
        //secondNearestStrip method solves the strip points
        return secondNearestStrip(strip, k, ans);
    }
```

All other operations take constant time, i.e. Θ(1).

Now, looking into the method (v) may seem to have a complexity of $O(n^2)$ times in the worst case but the inner loop runs at most 8 times in the worst case which has been hard coded here. Therefore, this function has O(n) time complexity.

Time Complexity of the Algorithm:

Combining equations (2), (3), (4) and (5), the recurrence relation for findSecondMinimum() is:

T(n)= O(n) + 2T(n/2) + O(n) + O(n) = 2T(n/2) + 3O(n) = 2T(n/2) + O(n)

whose solution is **T(n) = O(nlogn)**-------(6) [Using master theorem].

Finally, combining equation (1) and (6), the method, secondMinimum() has the time complexity**: T(n) = O(nlogn) + O(nlogn) = O(nlogn).**

So, time complexity of the algorithm is: O(nlogn).