CSE-204:
Data Structures and Algorithms I Sessional

Offline-8:
Divide and Conquer Algorithms

Submitted by:

Tanjeem Azwad Zaman

Roll: 1805006

Dept. of CSE, BUET

Section: A1 Batch:18

Date of Submission:

18.06.2021

The algorithm employed to solve this problems uses a two-step approach. The first step uses a function that works on a divide and conquer approach and finds out the **closest distance (minD)** between the given set of points. This step is O(nlogn).

The second step uses the "minD" found in Step 1. With the help of the same function and some conditionals, this step disregards all the distances corresponding to "minD" and thus calculates the second closest distance on the basis of the same "Divide and Conquer" Approach. This is also O(nlogn)

These two steps are completed consecutively in the "FindSecondClosest" function.

Analysis for the algorithm (Works for both the steps mentioned above):

Pre-processing:

- Copies two arrays: **Takes O(n) each**
- Sorts them using Merge Sort. **Takes O(nlogn)**
- The "Point" class has an extra field "xidx" , that stores the position of the point when sorted in ascending order. This is assigned iteratively in preprocessing. **Takes O(n)**

The function "FindClosestPair" takes in the following parameters:

- Int nth:
  - 1 denotes finding the pair of points with smallest distance between them.
  - 2 denotes finding the second closest pair.
  - This is the only parameter that is also not present in "FindSecondClosest"
- Int Xstart:
  - Starting index of the x-based sorted Array (inclusive)
- Int Xend:
  - Ending index of the x-based sorted Array (inclusive)
- Point [] px:
  - Entire n-length array of the Points sorted in ascending order of x
- Point[] py:
  - (Not necessarity contiguous) Sub-array of Points sorted in ascending order of y. Length Xstart – Xend + 1.
- Double mind:
  - If nth ==2, this will store the minimum distance of closest pair found earlier.

In each step:

1. Checks for the base cases of n =2 and n =3. If yes, handles them. **Done in O(1)**
2. Finds mid-point of the Sub-array under consideration. **Done is O(1)**
3. Divides "py" into smaller subarrays "pylow" and "pyhigh", on the basis of "xidx". **Takes O(n)**
4. Calls "FindClosestPair" on lower and upper half of the array, through recursive calls. **Takes T(n/2)**
5. Finds smaller distance between the two calls. **Takes O(1)**
6. Populates new array with points within Strip, by an iterative approach through "py". **Takes O(n)**
7. Compares each point with Seven / Fifteen of the next points, for calculating smallest distance/ 2^{nd} smallest Distance respectively. **Takes O(n)**

Here,

- Steps 2,3,4 fall under "Divide",
- Step 1 falls under "Conquer" (Base-Cases) and
- Step 5,6, 7 falls under "Combine"

Post-Processing:

- A linear search through the original array gives us the indices for the corresponding points. Done in O(n)

Recursive Relation and Time Complexity:

- The preprocessing is **O(nlogn)** and post processing is **O(n)**
- Recursive Relation for main algorithm: **T(n) = T(n/2) + O(n)**
- Solving which gives us **T(n) = θ(nlogn)** for *each of the two steps*.
- Combining, we get the total tome complexity for the entire algorithm, which comes to **θ(nlogn)**
- This complies with the given algorithm constraints