Bangladesh University of Engineering & Technology

1805004

COURSE

CSE 204

Data Structures and Algorithms I Sessional

Report On:

Offline 8 Complexity Analysis

Topic:

Divide & Conquer

Author:
Syed Jarullah Hisham
Roll: 1805004
CSE'18 Section A1

June 17, 2021

Time Complexity Analysis

Divide and conquer approach was used to correctly find the second nearest pair of houses. This approach takes O(nlogn) time to execute. The step by step time complexity with its algorithm is described below:

Preprocess

Step 1:

Firstly, two array were created and sorted according to Y and X-coordinates. We will call these arrays as X-array and Y-array in the upcoming discussions. By default, java uses merge sort algorithm to sort user defined objects' array. So, this step requires O(nlogn) time. Other operations like initialiing or copy array all take less time i.e O(n) time. **So, overall the time complexity of this step is O(nlogn)**.

Step 2:

If total number of points are 3, then brute force algorithm is used which takes constant amount of time, **so this step requires O(1) time**.

Next, nearest pair of point algorithm is called.

Nearest Pair of Point Algorithm

Step 3:

Divide & Conquer:

It is one of the most important step of this algorithm. The points which were sorted by X-coordinates are divided into two equal halves (left part and right part) and recursively call nearest pair of point algorithm to find the value of delta (the lowest distance of two points in a particular subarray). As we divide original array into two subarrays and recursively call the algorithm, this step is called the divide & conquer step of this algorithm. This step does two things. One, finding value of delta and other is recursively sort the Y-array. If we T(n) be the running time of this whole step then the divide step requires T(n/2) time. And after calling these methods, we call merge method to ensure that the Y-array are correctly sorted according to Y-coordinates. This merge method requires O(n) time.

So, by observing the step 2 & 3, we can figure out a simple recursive equation.

Let, T(n) is the running time of this step.

$$T(n) = \begin{cases} O(1) & \text{if } n = 3 \\ 2T(n/2) + O(n) & \text{if } n > 3 \end{cases}$$

Now, using the master theorem, we can prove that T(n) = O(nlogn). **So, the divide & conquer step requires O(nlogn) time**.

Step 4:

Merge Step:

This is the trickiest and most important part of the analysis. In this step, firstly, based on the value of delta, we will create an array of strip covering the region with all points which are at most delta distance away from the middle line dividing the two sets. **This strip creation takes O(n) time**.

Next, the smallest distance in strip is determined. For this, we use nested loop to find the minimum distance. So, at first look, it is seen that finding smallest distance requires O(n^2) time. **But actually, it is O(n)**. It can be proved geometrically that for every point in strip, we only need to check at most 7 points after it.

We can prove it by the following approach:

Theorem: Let, any point in the box P_k and any pair of point is Y_i and Y_j. We have to prove that there are at most 7 other points such that $Y_i - Y_j <=$ delta.

Proof:

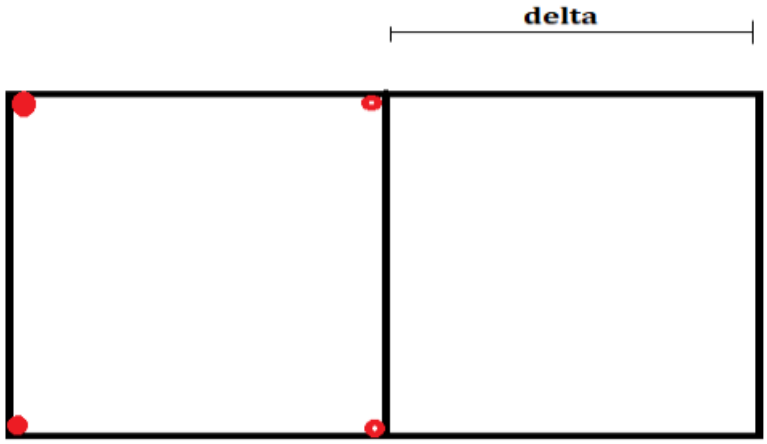

Figure 1: delta covered squared area

- In the figure, left and right square have equal area of delta * delta.
- So, all points Pk must lie in anyone of left and right sides.
- Having delta at each side, we can say that all points have distance <= delta from others.
- As we have two squares, and each having 4 points at the corners; we get total 8 points. If Pk is one point, then there are 7 other points.

$\therefore$ There are at most 7 other points in the range of delta other than the point itself. So, we have to check at most 7 points to find the nearest distance.

So, the total loop operation is at most 7*n times. As 7 is a constant, it takes constant time to execute the inner loop. $\therefore$ **total required runtime for merge step is O(n)**.

Second Nearest Pair of Points

Step 5:

This step simply applies nearest pair finder algorithm twice ignoring one of the nearest pair each time that we found in previous step. Finally, find the second nearest pair comparing the results of this two. **So, this step requires O(nlogn) time**.

So, total cost of the whole algorithm is:
Sorting + Nearest Pair of points + Second nearest pair of points
= O(nlogn) + O(nlogn) + O(nlogn)
= O(nlogn)

Time Complexity Using Code Snippet

Here is a code snippet showing a summary of time complexity of the whole algorithm