COMPLEXITY ANALYSIS OF THE ALGORITHM ON FINDING SECOND CLOSEST POINT

1805087 - Fardin Anam Aungon

CSE204

Description:

To find the second closest point on a plane, a divide and conquer approach is followed. The main 'divide and conquer' step has been written in the method below:

```
Point[][] divideAndFindSecondClosest(Point[] sortedX, Point[] sortedY, int startIxd, int endIdx)
```

This method has initialized with another method below:

```
Point[] findSecondClosestPoints(Point[] points, int n)
```

The above method just sorts the points along both the axes and pass those to the *"divideAndFindSecondClosest"* method.

Complexity Analysis:

If we look closer into the method "*findSecondClosestPoints*" it generates two arrays of size n and sorts it. And then it calls the method that is implemented with divide and conquer approach.

The complexity of sorting is **O(nlogn)**.

Let the method *"divideAndFindSecondClosest"* requires T(n) time. And the method *"divideAndFindSecondClosest"* requires T'(n) time. T'(n) is the total complexity of my program.

So, the total running time is T'(n) = T(n) + O(nlogn)

Complexity of "*divideAndFindSecondClosest*" method:

Base case: If n <= 5 then we consider it to be the base case. In the base condition, at max 5 sorted points are compared with each other. As the value of n is at max 5 and the points are sorted so, this will run at max (5+4+3+2=14) times. So, it's time complexity is O(1).

Divide Step: In this step, the set of arrays are divided into two sets of arrays along the middle position of x axis. Middle point is calculated in O(1). And the set of sorted points along y axis are divided through a loop which takes O(n) time.

Conquer Step: Here, the method *divideAndFindSecondClosest* is called twice recursively. So, it will cost 2T(n/2) time.

Combine Step: When we try to combine the two sets from where we have already found the two pairs of points with closest distance, it is seen that we haven't considered any pair of points where one is from the left set and the other is from the right set. As the second min distance can only be possible between the two subsets at a distance smaller than or equal to the

present second closest distance on either side of the vertical division line. So, only those points that are inside that range are taken. An array of the points that are inside that range are taken from the presorted array. So, the time complexity is O(n). Finally, the two pairs of points with closest distance in the region is found and compared with the previously found closest points. This takes O(n) time because, the dataset is already sorted.

So, the total complexity of time becomes

$T(n) = 2T(n/2) + O(1) + O(1) + O(n) + O(n)$

⇨ $T(n) = 2T(n/2) + O(n)$

Therefore,

$T(n) = O(1)$; $n <= 5$

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

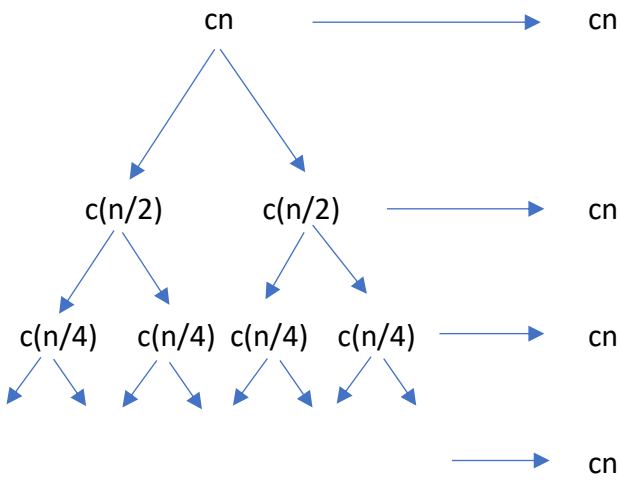
According to master theorem,

Here, $a = 2$, $b = 2$ and $f(n) = O(n)$

And $n \wedge \log_b a = n \wedge \log_2 2 = n^1 = n$

So, we can guess the complexity to be O(nlogn)

Proving the complexity O(nlogn) with recursion tree:



Here, the height of the tree is $\log_2 n$

So, total number of levels is $\log_2 n + 1$

Cost of every level is = cn

Therefore, total cost is $cn(\log_2 n + 1) = cn\log_2 n + cn$

So the complexity is O(nlogn)

Therefore, T(n) = O(nlogn)

So, T'(n) = O(nlogn) + O(nlogn) = O(nlogn)

Hence, total time complexity of the algorithm is **O(nlogn)**