REPORT
on
Assignment 8

Name: Abdus Samee
ID: 1805021
Course: CSE - 204
Topic: Divide and Conquer

Pseudo code:

The pseudo code for my approach for finding the second nearest pair is as follows:

```
1  n ← input number of pair of points
2  arr ← array of points
3  xSort ← sorted array of points with respect to abscissa in ascending order
4  ySort ← sorted array of points with respect to ordinate in ascending order
5  from ← null
6  to ← null
7  foundMin ← false

   SECOND-NEAREST-PAIR()
8      from, to, d = NEAREST-PAIR(0, xSort.length-1)
9      from, to, d = NEAREST-PAIR(0, xSort.length-1)

10     return from, to, d

   DIST(x, y)
11     return Euclidian distance between two points x and y

   NEAREST-PAIR(a, b)
12     n ← b-a+1

13     if n == 2
14         if foundMin==false or CHECK-NOT-MIN(a, b)
15             return a, b, DIST(a, b)
16         else
17             return null, null, +infinity
18     if n == 3
19         d1 ← DIST(a, b)
20         d2 ← DIST(a, a+1)
21         d3 ← DIST(a+1, b)
22         if foundMin == false
23             return minimum(d1, d2, d3) with start and end indices
24         else
25             if minimum(d1, d2, d3) is not actual minimum
26                 return minimum(d1, d2, d3) with start and end indices
```

```
27                      else
28                              return minimum out of rest two with start and end indices

29      mid ← (a+b)/2
30      leftFrom, leftTo, leftMin = NEAREST-PAIR(a, mid)
31      rightFrom, righto, rightMin = NEAREST-PAIR(mid+1, b)

32      d ← minimum(leftMin, rightMin)
33      from, to ← start and end points' serial number(from and to values)
                from the selected minimum
34      stripe ← list of points from ySort whose abscissa within range
                        [mid.x-d, mid.x+d]
35      for i=1 to stripe.size
36              for j=1 to 7
37                if DIST(stripe[i], stripe [i+j]) < d
38                  if ((from!=null and to!=null) or CHECK-NOT-MIN(stripe[i],stripe[i+j]))
39                    d = DIST(stripe[i], stripe[i+j])

40      return from, to, d

    CHECK-NOT-MIN(a, b)
41      return if (a, b) is previously calculated minimum possible pair
```

Analysis:

Upon analysis of the above pseudo code, we find that there are four methods used, SECOND-NEAREST-PAIR(), DIST(X, Y), NEAREST-PAIR(a, b), and CHECK-NOT-MIN(a, b). Before that, the lines **1** and **2** are executed in O(n). Next, lines **3** and **4** execute merge sort whose recurrence relation becomes $T(n) = 2T(n/2) + n$, which can be calculated as $\Theta(n\log n)$. Then lines from **5** to **7** require O(1) time. So, the presorting and assignments here require O(nlogn) in total.

The run-time of both the methods CHECK-NOT-MIN(a, b) at line **41** and DIST(x, y) at line **11** is $\Theta(1)$.

Now, the main calculation starts upon calling the method SECOND-NEAREST-PAIR(). It calls the method NEAREST-PAIR(a, b) two times. Suppose, the running time of NEAREST-PAIR(a, b) is T(n). In NEAREST-PAIR(a, b), there are two base cases which take constant amount of time from lines **12** to **28**. Thus, it runs in order of $\Theta(1)$.

Divide: In divide step at line **29**, only the computation of the middle element is done in constant time. So, $D(n) = \Theta(1)$.
Conquer: Again, in the conquer step of lines **30** and **31**, we recursively solve two subproblems, each of size n/2, which contributes $2T(n/2)$ to the running time.
Combine: In the lines **32** and **33**, $\Theta(1)$ is contributed to the running time while the lines from **34** to **39**, $\Theta(n)$ is contributed. In fact, the inner for loop in line **36** takes a constant amount of time $O(1)$. Thus, $C(n) = \Theta(n)$.

Thus, we have:

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

Using the master theorem, we get

$$T(n) = \Theta(n\log n)$$

This is the running time for a single call of the method NEAREST-PAIR(a, b). For the two calls, it also takes the same amount of time. Along with the presorting with merge sort, the whole complexity of the program remains $\Theta(n\log n)$, which can also be written as $O(n\log n)$.