BANGLADESH UNIVERSITY OF ENGINEERING AND TECHNOLOGY
Department of Computer Science and Engineering

Course: CSE204 (Data Structure and Algorithm I Sessional)

Assignment 8

Assignment name: DIVIDE AND CONQUER.

Name: Hasan Masum

Student ID: 1805052

Complexity analysis:

For finding 2nd nearest pair in O(nlogn) complexity we presorted the given array of points and then applied divide and conquer algorithm on the array. Presorting takes O(nlogn) complexity to sort the array with respect to x. Divide and conquer has three part: base case, divide step, combine part.

If T(n) is the running time of the algorithm, then

When n<=3, T(n) = T(base case)

And when n>3, T(n) = T(divide) + T(combine)

Now we will find complexity or remaining steps.

Base case:

When n<=3 we apply brute force to find the closest and second closest pair. We also sort the part of the array with respect to y so that we don't need to sort the array in combine step for n > 3. As the value n is very small we can say that base case takes **almost constant time** to find the closest pairs and also sort the array with respect to y. And we can ignore it.

```
// STEP-1: BASE CASE ============================================
// brute fource takes almost constant time
 if (n <= 3){
     for (int i = startIdx; i <= endIdx; i++){
         for (int j = i + 1; j <= endIdx; j++){
             findClosestFromPair(p[i], p[j]);
         }
     }

      //sort the sub array wrt y co ordinates
      sort(p + startIdx, p + endIdx + 1, Point::cmpY);

     return;
}
```

Divide:

In divide step we divide the points into two halves by a vertical line which divides the points such a way that each half side n/2 elements. Each half **will take T(n/2) time** to find the closest and second closest pair. So T(divide) = 2*T(n/2)

```
//STEP-2: DIVIDE
====================================================
int mid = startIdx + (endIdx - startIdx) / 2;
double midX = p[mid].getX();

// complexity T(n/2)
closestPair(startIdx, mid);
// complexity T(n/2)
closestPair(mid + 1, endIdx);
```

Combine:

Combine step has 2 part. In first part we merge the two sorted subarray(sorted in recursive calls) with respect to y co-ordinate in **O(n)** time complexity.

```
//STEP-3: MERGE
=====================================================
// merge the 2 sorted sub array wrt to y co-ordinate.
// Complexity O(n)
Point *tempAr = new Point[n];
for (int i = startIdx, j = mid + 1, k = 0; k < n; k++){
      if (i == mid + 1)
            tempAr[k] = p[j++];
        else if (j == endIdx + 1)
               tempAr[k] = p[i++];

            //else compare elements
        else if (Point::cmpY(p[i], p[j]))
               tempAr[k] = p[i++];
        else
               tempAr[k] = p[j++];
}

for (int i = 0, j = startIdx; i < n; i++, j++){
        p[j] = tempAr[i];
}

delete[] tempAr;
```

In the second part, we find closest pair with one point in each side of dividing line. If the second closest pair found in recursive calls has distance d, then we only need to consider points within d distance for the dividing line. The points are already sorted with respect to y co-ordinate in the strip which has 2*d width. So we need to check the points in 2d*d rectangle as points outside this rectangle has distance greater than d. And we can prove we need to check only next 7 points in the sorted list. So though we use nested loops to find the closest and second closest point in the strip, it actually has **O(n) complexity.** So T(combine) = O(n) + O(n).

```
for(int i = startIdx; i <= endIdx; i++){
     if (abs(p[i].getX() - midX) < secondClosestDist){
// check for the closest and 2nd closest point
// only in rectangle of (2secondClosetDist) * (secondClosetDist)
// and we check at most 7 points
     for (int j = i + 1;
      j <= endIdx && (p[j].getY() - p[i].getY()) <
secondClosestDist; j++){
             findClosestFromPair(p[i], p[j]);
        }
     }
}
```

So finally we can write, for divide and conquer part,

When n<=3, T(n) = O(1)

And when n>3, T(n) = 2*T(n/2) + O(n) + O(n)

We can ignore the lower order term and rewrite the complexity as

T(n) = 2*T(n/2) + O(n).

Now we will master theorem to determine the complexity.

By comparing T(n) with T(n) = a*T(n/b) + f(n),

We have a = 2, b = 2 and f(n) = O(n)

And $n^{log_a b} = n^{log_2 2} = n$. Since f(n) = Θ($n^{log_a b}$)

So we can write according to master theorem, **T(n) = O(nlogn)**

So finding second closest pair has time complexity O(nlogn)