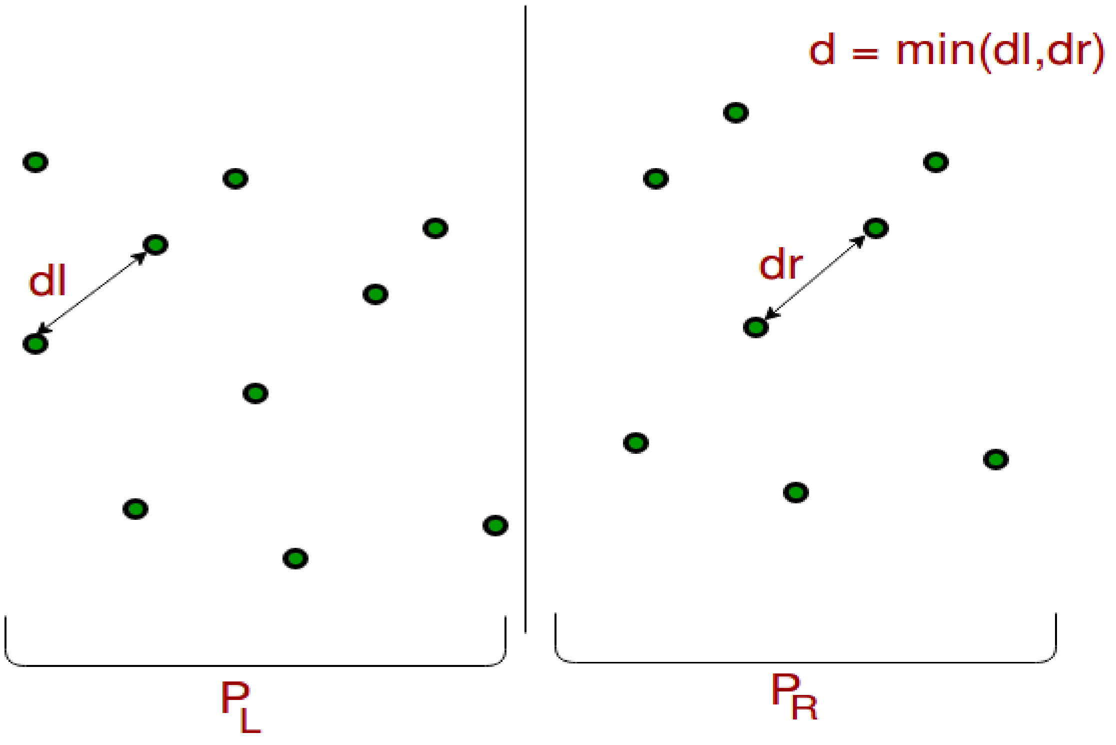
Closest Pair of Points using Divide and Conquer algorithm

We are given an array of n points in the plane, and the problem is to find out the closest pair of points in the array. This problem arises in a number of applications. For example, in air-traffic control, you may want to monitor planes that come too close together, since this may indicate a possible collision. Recall the following formula for distance between two points p and q.

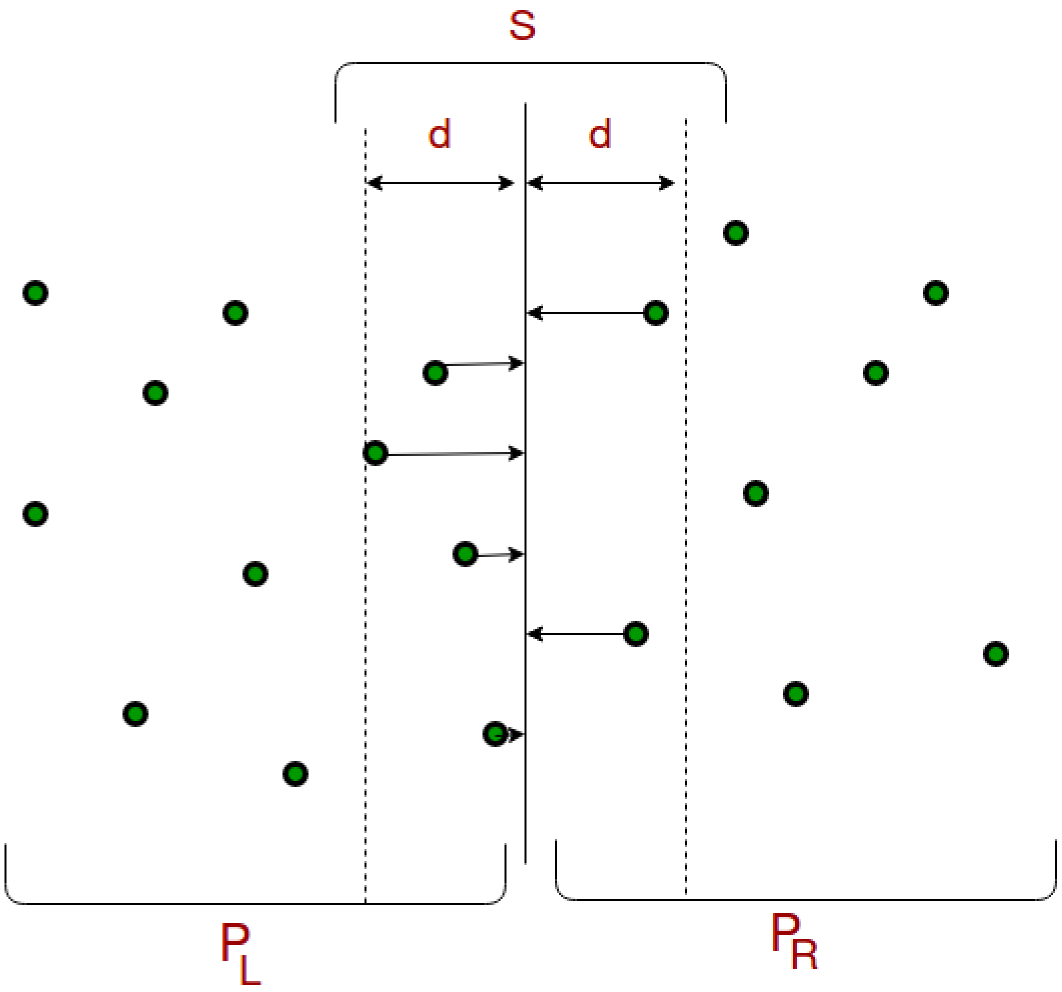
The Brute force solution is O(n^2), compute the distance between each pair and return the smallest. We can calculate the smallest distance in O(nLogn) time using Divide and Conquer strategy. In this post, a O(n x (Logn)^2) approach is discussed. We will be discussing a O(nLogn) approach in a separate post.

Algorithm
Following are the detailed steps of a O(n (Logn)^2) algortihm.
Input: An array of n points *P[]*
Output: The smallest distance between two points in the given array.
As a pre-processing step, the input array is sorted according to x coordinates.

1) Find the middle point in the sorted array, we can take *P[n/2]* as middle point.
2) Divide the given array in two halves. The first subarray contains points from P[0] to P[n/2]. The second subarray contains points from P[n/2+1] to P[n-1].
3) Recursively find the smallest distances in both subarrays. Let the distances be dl and dr. Find the minimum of dl and dr. Let the minimum be d.



4) From the above 3 steps, we have an upper bound d of minimum distance. Now we need to consider the pairs such that one point in pair is from the left half and the other is from the right half. Consider the vertical line passing through P[n/2] and find all points whose x coordinate is closer than d to the middle vertical line. Build an array strip[] of all such points.



5) Sort the array strip[] according to y coordinates. This step is O(nLogn). It can be optimized to O(n) by recursively sorting and merging.
6) Find the smallest distance in strip[]. This is tricky. From the first look, it seems to be a O(n^2) step, but it is actually O(n). It can be proved geometrically that for every point in the strip, we only need to check at most 7 points after it (note that strip is sorted according to Y coordinate). See this for more analysis.

7) Finally return the minimum of d and distance calculated in the above step (step 6)

```
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Comparator;
import java.util.List;
/**
public class ClosestPairOfPoint {

    static double min = Integer.MAX_VALUE;
        static Point p1 =null ,p2 = null;
        public static class Point {
            private int x;
            private int y;

            public Point(int x, int y) {
                this.x = x;
                this.y = y;
            }
        }

        private static double getMin(){
            return min;
        }

        public static void mindistance(List<Point> list) throws
IllegalArgumentException{
            if(list==null || list.size()<2) throw new
IllegalArgumentException("We need atleast 2 points");
            for(int i=0;i<list.size();i++) {
                if(list.get(i)==null)
                    throw new IllegalArgumentException("Point is not
initialised");
            }
            int n = list.size();
            Point[] pointsbyX = new Point[n];
            for(int i=0;i<n;i++){
                pointsbyX[i] = list.get(i);
            }
            Arrays.sort(pointsbyX, new Comparator<Point>() {
                @Override
                public int compare(Point o1, Point o2) {
                    if(o1.x!=o2.x)
                        return o1.x-o2.x;
                    else
                        return o1.y-o2.y;
                }
            });
            for(int i=0;i<n-1;i++){
                if(pointsbyX[i]==pointsbyX[i+1]){
                    min = 0;
                    p1 = pointsbyX[i];
                    p2 = pointsbyX[i+1];
                    break;
                }
            }
            Point[] pointsbyY = new Point[n];
            for (int i = 0; i < n; i++)
                pointsbyY[i] = pointsbyX[i];
            Point[] aux = new Point[n];
            closest(pointsbyX, pointsbyY, aux, 0, n-1);
        }

        private static double closest(Point[] pointsByX, Point[] pointsByY,
Point[] aux, int lo, int hi) {
            if (hi <= lo) return Double.POSITIVE_INFINITY;
            int mid = lo + (hi - lo) / 2;
            Point median = pointsByX[mid];

            // compute closest pair with both endpoints in left subarray or
both in right subarray
            double delta1 = closest(pointsByX, pointsByY, aux, lo, mid);
            double delta2 = closest(pointsByX, pointsByY, aux, mid+1, hi);
            double delta = Math.min(delta1, delta2);

            // merge back so that pointsByY[lo..hi] are sorted by y-coordinate
            merge(pointsByY, aux, lo, mid, hi);

            // aux[0..m-1] = sequence of points closer than delta, sorted by
y-coordinate
            int m = 0;
            for (int i = lo; i <= hi; i++) {
                if (Math.abs(pointsByY[i].x - median.x) < delta)
                    aux[m++] = pointsByY[i];
            }

            // compare each point to its neighbors with y-coordinate closer
than delta
            for (int i = 0; i < m; i++) {
                // a geometric packing argument shows that this loop iterates
at most 7 times
                for (int j = i+1; (j < m) && (aux[j].y - aux[i].y < delta);
j++) {
                    double distance = getDistance(aux[i], aux[j]);
                    if (distance < delta) {
                        delta = distance;
                        if (distance < min) {
                            min = delta;
                            p1 = aux[i];
                            p2 = aux[j];
                            // StdOut.println("better distance = " + delta + "
from " + best1 + " to " + best2);
                        }
                    }
                }
            }
            return delta;
        }

        private static void merge(Point[] a, Point[] aux, int lo, int mid, int
hi) {
            // copy to aux[]
            for (int k = lo; k <= hi; k++) {
                aux[k] = a[k];
```

```
            }

            // merge back to a[]
            int i = lo, j = mid+1;
            for (int k = lo; k <= hi; k++) {
                if      (i > mid)              a[k] = aux[j++];
                else if (j > hi)               a[k] = aux[i++];
                else if (less(aux[j], aux[i])) a[k] = aux[j++];
                else                           a[k] = aux[i++];
            }
        }

        private static boolean less(Point v, Point w) {
            return v.x<w.x;
        }

        public static double getDistance(Point a, Point b){
            int x = a.x-b.x;
            int y = a.y-b.y;
            return Math.sqrt(x*x+y*y);
        }


        public static void main(String[] args) {
            Point p1 = new Point(2,3);
            Point p2 = new Point(12,30);
            Point p3 = new Point(40,50);
            Point p4 = new Point(5,1);
            Point p5 = new Point(12,10);
            Point p6 = new Point(3,4);
            List<Point> list = new ArrayList<>();
            list.add(p1);  list.add(p2); list.add(p3); list.add(p4);
list.add(p5); list.add(p6);
            mindistance(list);
            System.out.println("The closest pair of points are
("+p1.x+","+p1.y+") ("+p2.x+","+p2.y+") and the distance between them is "+
min);
        }


}

 */


// BruteForce Approach
public class ClosestPairOfPoint {

    public static class Point {
        private int x;
        private int y;
        public Point(int x, int y) {
            this.x = x;
            this.y = y;
        }
    }
```

```java
    public List<Point> mindistance(List<Point> list){
        if(list==null || list.size()==0) return new ArrayList<>();
        List<Point> result = new ArrayList<>();
        int n = list.size();
        int min = Integer.MAX_VALUE;
        for(int i=0;i<n;i++){
            for(int j=i+1;j<n;j++){
                int distance = getDistance(list.get(i), list.get(j));
                if(min>distance || result.size()==0){
                    result.clear();
                    result.add(list.get(i));
                    result.add(list.get(j));
                    min = distance;
                }
            }
        }
        return result;
    }
    public int getDistance(Point a, Point b){
        int x = a.x - b.x;
        int y = a.y - b.y;
        return x*x + y*y;
    }
    public void display(List<Point> list){
        Point a = list.get(0);
        Point b = list.get(1);
        System.out.print("{"+a.x+"  "+a.y+"}");
        System.out.print("{"+b.x+"  "+b.y+"}");
        System.out.println(" ");
    }
    public static void main(String[] args) {
        ClosestPairOfPoint m = new ClosestPairOfPoint();
        Point p1 = new Point(2,3);
        Point p2 = new Point(12,30);
        Point p3 = new Point(40,50);
        Point p4 = new Point(5,1);
        Point p5 = new Point(12,10);
        Point p6 = new Point(3,4);
        List<Point> list = new ArrayList<>();
        list.add(p1);  list.add(p2); list.add(p3); list.add(p4); list.add(p5);
list.add(p6);
        System.out.println("The closest pair of points are ");
        List<Point> result = m.mindistance(list);
        m.display(result);
        System.out.println("The distance between the closest points is
"+Math.sqrt(m.getDistance(result.get(0), result.get(1))));
    }

}
```