**Assignment Week 2**

**Code:**

import java.util.ArrayList;

import java.util.Arrays;

import java.util.Collections;

import java.util.Comparator;

import java.util.List;

public class Main {

static class Point {

int x, y;

Point(int x, int y) {

this.x = x;

this.y = y;

}

}

public static double distance(Point p1, Point p2) {

return Math.sqrt(Math.pow(p1.x - p2.x, 2) + Math.pow(p1.y - p2.y, 2));

}

public static Point[] bruteForceClosestPair(List<Point> points) {

double minDistance = Double.MAX\_VALUE;

Point[] closestPair = new Point[2];

int n = points.size();

for (int i = 0; i < n; i++) {

for (int j = i + 1; j < n; j++) {

double dist = distance(points.get(i), points.get(j));

if (dist < minDistance) {

minDistance = dist;

closestPair[0] = points.get(i);

closestPair[1] = points.get(j);

}

}

}

return closestPair;

}

public static Point[] closestPairDivideAndConquer(List<Point> points) {

Point[] pointsArray = points.toArray(new Point[0]);

Arrays.sort(pointsArray, Comparator.comparingInt(p -> p.x));

return closestPairRecursive(pointsArray);

}

private static Point[] closestPairRecursive(Point[] points) {

int n = points.length;

if (n <= 3) {

return bruteForceClosestPair(Arrays.asList(points));

}

int mid = n / 2;

Point[] left = Arrays.copyOfRange(points, 0, mid);

Point[] right = Arrays.copyOfRange(points, mid, n);

Point[] leftClosest = closestPairRecursive(left);

Point[] rightClosest = closestPairRecursive(right);

double leftDist = distance(leftClosest[0], leftClosest[1]);

double rightDist = distance(rightClosest[0], rightClosest[1]);

double minDist = Math.min(leftDist, rightDist);

Point[] closestPair = leftDist < rightDist ? leftClosest : rightClosest;

List<Point> strip = new ArrayList<>();

for (Point p : points) {

if (Math.abs(p.x - points[mid].x) < minDist) {

strip.add(p);

}

}

Point[] stripArray = strip.toArray(new Point[0]);

Arrays.sort(stripArray, Comparator.comparingInt(p -> p.y));

for (int i = 0; i < stripArray.length; i++) {

for (int j = i + 1; j < stripArray.length && (stripArray[j].y - stripArray[i].y) < minDist; j++) {

double dist = distance(stripArray[i], stripArray[j]);

if (dist < minDist) {

minDist = dist;

closestPair[0] = stripArray[i];

closestPair[1] = stripArray[j];

}

}

}

return closestPair;

}

public static void simulateDroneDelivery(List<Point> points, Point[] closestPair) {

Collections.sort(points, (p1, p2) -> {

double d1 = Math.min(distance(p1, closestPair[0]), distance(p1, closestPair[1]));

double d2 = Math.min(distance(p2, closestPair[0]), distance(p2, closestPair[1]));

return Double.compare(d1, d2);

});

double totalDistance = 0;

for (int i = 0; i < points.size() - 1; i++) {

totalDistance += distance(points.get(i), points.get(i + 1));

}

System.out.println("Total Distance Traveled: " + totalDistance);

}

public static List<Point> createpoints(){

List<Point> p = new ArrayList<>();

int[][] a = {

{79, 10}, {79, 26}, {96, 51}, {64, 53}, {70, 28}, {44, 8}, {79, 87}, {35, 98}, {62, 64}, {61, 61}, {89, 40},

{90, 44}, {81, 72}, {22, 69}, {54, 65}, {24, 16}, {25, 78}, {6, 42}, {89, 53}, {52, 44}, {100, 80}, {99, 31},

{49, 97}, {91, 76}, {11, 26}, {35, 86}, {85, 73}, {43, 98}, {9, 19}, {8, 98}, {72, 25}, {55, 84}, {57, 21},

{41, 43}, {72, 55}, {88, 2}, {35, 15}, {24, 60}, {40, 33}, {33, 59}, {14, 44}, {69, 18}, {18, 2}, {92, 17},

{13, 83}, {12, 37}, {23, 90}, {22, 69}, {0, 13}, {65, 55}, {77, 5}, {62, 4}, {6, 75}, {83, 21}, {26, 11},

{8, 45}, {71, 71}, {12, 28}, {16, 92}, {41, 73}, {3, 61}, {80, 47}, {36, 92}, {69, 16}, {16, 74}, {20, 16},

{96, 96}, {100, 65}, {96, 61}, {61, 56}, {38, 91}, {35, 56}, {71, 91}, {41, 84}, {70, 21}, {5, 28}, {40, 91},

{20, 60}, {34, 99}, {7, 16}, {95, 4}, {55, 11}, {74, 53}, {18, 88}, {15, 12}, {21, 39}, {82, 44}, {25, 25},

{95, 35}, {43, 84}, {81, 14}, {78, 35}, {32, 47}, {52, 10}, {6, 57}, {77, 39}, {18, 32}, {45, 60}, {86, 41},

{52, 93}, {16, 36}, {14, 49}, {36, 29}, {70, 64}, {73, 80}, {85, 66}, {7, 1}, {44, 37}, {2, 61}, {29, 78},

{91, 88}, {56, 87}, {70, 33}, {13, 63}, {16, 80}, {92, 31}, {79, 88}, {28, 43}, {81, 19}, {56, 47}, {13, 67}, {6, 88},

{39, 98}, {7, 22}, {58, 22}, {95, 85}, {70, 18}, {57, 43}, {10, 40}, {0, 33}, {59, 59}, {74, 91}, {88, 66},

{17, 40}, {6, 30}, {72, 9}, {7, 51}, {100, 62}, {40, 39}, {91, 91}, {93, 15}, {71, 98}, {12, 33}, {26, 4},

{23, 58}, {25, 65}, {74, 8}, {27, 73}, {90, 68}, {4, 98}, {45, 82}, {35, 17}, {1, 44}, {87, 24}, {99, 10},

{19, 60}, {74, 1}, {93, 64}, {1, 0}, {80, 85}, {99, 68}, {64, 70}, {27, 43}, {14, 53}, {59, 39}, {84, 59},

{82, 8}, {40, 42}, {48, 34}, {36, 71}, {52, 62}, {53, 80}, {76, 50}, {71, 17}, {93, 56}, {21, 49}, {57, 96},

{40, 53}, {88, 2}, {66, 61}, {37, 87}, {39, 56}, {17, 6}, {46, 72}, {23, 88}, {36, 18}, {16, 96}, {96, 5},

{45, 36}, {37, 63}, {37, 6}, {65, 72}, {10, 80}, {39, 35}, {83, 33}, {91, 35}, {55, 77}, {93, 80}, {46, 97},

{11, 42}, {0, 99}, {14, 15}, {20, 24}, {48, 43}, {71, 88}, {77, 96}, {47, 67}, {71, 95}, {74, 99}, {72, 11},

{37, 49}, {51, 94}, {97, 26}, {86, 97}, {10, 41}, {89, 91}, {77, 64}, {72, 0}, {20, 35}, {81, 100}, {25, 52},

{47, 49}, {26, 34}, {4, 14}, {39, 68}, {15, 35}, {24, 51}, {14, 99}, {26, 1}, {99, 0}, {47, 50}, {89, 61}, {38, 72},

{21, 80}, {73, 8}, {26, 42}, {76, 17}, {8, 44}, {31, 33}, {55, 92}, {41, 79}, {23, 56}, {49, 31}, {64, 60}, {10, 18},

{22, 84}, {69, 32}, {42, 24}, {73, 3}, {40, 18}, {26, 5}, {93, 44}, {44, 68}, {66, 33}, {47, 47}, {49, 78}, {54, 49},

{67, 45}, {77, 36}, {26, 79}, {28, 8}, {94, 71}, {53, 85}, {12, 44}, {13, 3}, {52, 61}, {25, 20}, {42, 7}, {60, 43},

{82, 26}, {27, 73}, {5, 88}, {25, 67}, {47, 32}, {46, 18}, {45, 9}, {34, 63}, {31, 68}, {80, 69}, {27, 22}, {11, 33},

{63, 20}, {63, 32}, {75, 98}, {78, 32}, {37, 78}, {70, 6}, {58, 38}, {64, 93}, {34, 58}, {24, 94}, {39, 99}, {1, 96},

{6, 45}, {45, 0}, {69, 48}, {79, 92}, {54, 77}, {50, 98}, {62, 37}, {56, 74}, {28, 54}, {76, 15}, {10, 93}, {63, 79},

{51, 70}, {71, 7}, {10, 59}, {6, 46}, {12, 39}, {85, 29}, {93, 0}, {15, 7}, {32, 37}, {100, 100}, {60, 91}, {42, 27},

{73, 18}, {80, 94}, {83, 78}, {73, 11}, {13, 51}, {28, 9}, {77, 54}, {58, 33}, {68, 97}, {55, 76}, {69, 93}, {3, 87},

{58, 8}, {43, 31}, {17, 16}, {44, 62}, {51, 80}, {79, 30}, {45, 45}, {86, 61}, {3, 97}, {4, 62}, {92, 7}, {0, 11}, {4, 26},

{79, 32}, {39, 58}, {18, 66}, {63, 91}, {59, 4}, {36, 21}, {53, 14}, {54, 12}, {83, 65}, {9, 25}, {64, 55}, {79, 53},

{6, 26}, {83, 56}, {14, 33}, {65, 19}, {100, 56}, {23, 42}, {51, 51}, {73, 76}, {40, 9}, {99, 14}, {77, 78}, {70, 5},

{13, 85}, {34, 86}, {65, 4}, {61, 60}, {15, 35}, {60, 50}, {76, 15}, {79, 41}, {5, 53}, {16, 99}, {96, 40}, {79, 77},

{97, 42}, {39, 76}, {91, 78}, {96, 80}, {44, 43}, {1, 40}, {42, 46}, {21, 99}, {53, 88}, {90, 31}, {32, 19}, {50, 37},

{83, 55}, {98, 100}, {49, 8}, {47, 39}, {60, 67}, {87, 11}, {59, 40}, {72, 88}, {27, 54}, {99, 15}, {48, 18}, {72, 26},

{94, 5}, {13, 48}, {53, 38}, {40, 39}, {47, 44}, {30, 59}, {86, 63}, {23, 41}, {25, 19}, {54, 46}, {54, 39}, {40, 34},

{58, 58}, {97, 42}, {97, 34}, {0, 39}, {35, 26}, {44, 14}, {51, 15}, {55, 64}, {34, 55}, {82, 13}, {18, 6}, {4, 17},

{2, 19}, {11, 48}, {65, 4}, {11, 92}, {36, 32}, {20, 45}, {17, 67}, {63, 1}, {73, 2}, {98, 10}, {50, 33}, {23, 79},

{69, 20}, {55, 56}, {30, 36}, {13, 12}, {80, 27}, {12, 90}, {61, 11}, {44, 46}, {82, 27}, {17, 30}, {73, 81}, {74, 76},

{93, 13}, {16, 56}, {54, 97}, {49, 86}, {96, 51}, {16, 77}, {23, 31}, {1, 14}, {20, 95}, {53, 8}, {26, 23}, {3, 12},

{61, 16}, {87, 85}, {65, 5}, {36, 54}, {99, 81}, {72, 53}, {53, 54}, {58, 8}, {82, 50}, {100, 84}, {7, 90}, {80, 80},

{21, 85}, {51, 75}, {71, 23}, {20, 85}, {94, 40}, {87, 55}, {85, 49}, {100, 10}, {74, 56}, {9, 68}, {20, 39}, {16, 97},

{26, 39}, {45, 42}, {33, 17}, {50, 63}, {1, 34}, {15, 11}, {56, 16}, {84, 26}, {100, 75}, {99, 81}, {18, 40}, {39, 31},

{89, 14}, {4, 24}};

for(int i=0;i<a.length;i++)

p.add(new Point(a[i][0],a[i][1]));

return p;

}

public static void main(String[] args) {

List<Point> points = createpoints();

long startTime, endTime;

startTime = System.currentTimeMillis();

Point[] closestPairBruteForce = bruteForceClosestPair(points);

endTime = System.currentTimeMillis();

System.out.println("Closest Pair (Brute-Force): (" + closestPairBruteForce[0].x + ", " + closestPairBruteForce[0].y +

"), (" + closestPairBruteForce[1].x + ", " + closestPairBruteForce[1].y + ")");

System.out.println("Brute-Force Execution Time: " + (endTime - startTime) + " ms");

simulateDroneDelivery(points, closestPairBruteForce);

startTime = System.currentTimeMillis();

Point[] closestPairDivideAndConquer = closestPairDivideAndConquer(points);

endTime = System.currentTimeMillis();

System.out.println("Closest Pair (Divide-and-Conquer): (" + closestPairDivideAndConquer[0].x + ", " + closestPairDivideAndConquer[0].y +

"), (" + closestPairDivideAndConquer[1].x + ", " + closestPairDivideAndConquer[1].y + ")");

System.out.println("Divide-and-Conquer Execution Time: " + (endTime - startTime) + " ms");

simulateDroneDelivery(points, closestPairDivideAndConquer);

}

}

**Output:**

Closest Pair (Brute-Force): (96, 51), (96, 51)

Brute-Force Execution Time: 8 ms

Total Distance Traveled: 17248.791990899794

Closest Pair (Divide-and-Conquer): (99, 81), (99, 81)

Divide-and-Conquer Execution Time: 4 ms

Total Distance Traveled: 15898.89386545681