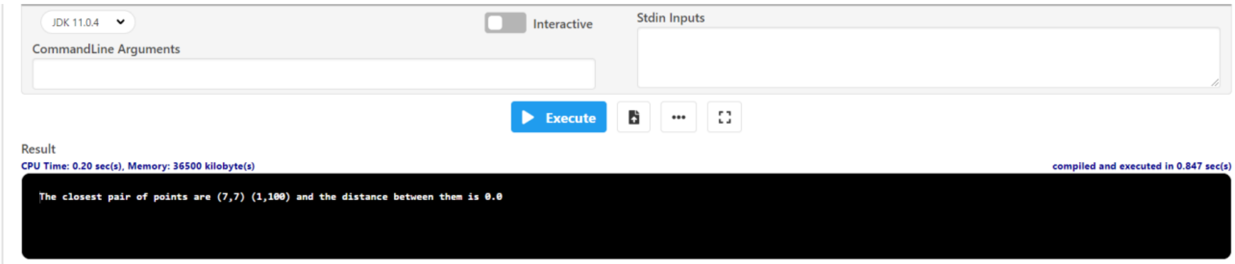
HARSHITHA
19BCE7582

DAA ASSIGNMENT

NEAREST NEIGHBOURING ALGO

```
import java.util.*;


public class Main {
        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];


    double delta1 = closest(pointsByX, pointsByY, aux, lo, mid);
    double delta2 = closest(pointsByX, pointsByY, aux, mid+1, hi);
    double delta = Math.min(delta1, delta2);


    merge(pointsByY, aux, lo, mid, hi);


    int m = 0;
    for (int i = lo; i <= hi; i++) {
            if (Math.abs(pointsByY[i].x - median.x) < delta)
                aux[m++] = pointsByY[i];
        }


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

            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];

                    }
                }
            }
        }
        return delta;
    }

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



        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(7,7);

        Point p2 = new Point(1,100);

        Point p3 = new Point(4,8);

        Point p4 = new Point(7,7);

        /*

                Point p1 = new Point(0,0);

        Point p2 = new Point(3,4);*/

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

        list.add(p1);  list.add(p2);list.add(p3);  list.add(p4);

        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);


    }

}
```



JDK 11.0.4

CPU Time: 0.20 sec(s), Memory: 36500 kilobyte(s)

compiled and executed in 0.847 sec(s)

```
The closest pair of points are (7,7) (1,100) and the distance between them is 0.0
```

MergeSort: O(n log n) time.

```
def closest(P, n):
P.sort(key=lambda point: point.x)
   Q = copy.deepcopy(P)
Q.sort(key=lambda point: point.y)
   return closestUtil(P, Q, n)
```

Therefore, Total Running Time: n + n log n + n log n = n(1+ log n + log n)

= 2n log n

Which gives O(n log n)