# Find Closest Pair of Points

Given n points on the plane. Each point is defined by its coordinates (xi,yi)(xi,yi). It is required to find among them two such points, such that the distance between them is minimal:

We take the usual Euclidean distances:

$$\rho(pi,pj)=\sqrt{(xi-xj)^2+(yi-yj)^2}.$$

**Algorithm:**

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[]. 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.

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

**Code:**

```
#include <cstdio>

#include <algorithm>

#include <cmath>

#include <limits>

using namespace std;


struct Point {
```

```
    double x, y;
};

Point result1, result2;
double bestDistance;

double euclideanDistance(Point a, Point b)  {
    double X = a.x - b.x, Y = a.y - b.y;
    return sqrt(X*X + Y*Y);
}

// comparison first done by y coordinate, then by x coordinate
bool less(Point a, Point b) {
    if(a.y < b.y) return true;
    if(a.y > b.y) return false;
    return a.x < b.x;
}

void merge(Point* a, Point* aux, int lo, int mid, int hi)   {
    int i, j, k;
    for(k = lo; k <= hi; k++)
        aux[k] = a[k];

    i = lo; j = mid + 1; k = lo;
    while(i <= mid && j <= hi)
        a[k++] = less(aux[i], aux[j]) ? aux[i++] : aux[j++];

    // Copy the rest of the left side of the array into the target array
    while(i <= mid)
        a[k++] = aux[i++];
```

```
}

double closestPair(Point* pointsByX, Point* pointsByY, Point* aux, int lo, int hi)    {
    if(hi <= lo)
        return numeric_limits<double>::infinity();

    int mid = lo + (hi - lo)/2;
    double delta = closestPair(pointsByX, pointsByY, aux, lo, mid);
    double dist = closestPair(pointsByX, pointsByY, aux, mid+1, hi);
    if(dist < delta)
        delta = dist;

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

    int M = 0, i, j;
    for(i = lo; i <= hi; i++)
        if(abs(pointsByY[i].x - pointsByX[mid].x) < delta)
            aux[M++] = pointsByY[i];

    double distance, t;
    for(i = 0; i < M; i++)  {
        for(j = i+1; j < M && (aux[j].y - aux[i].y < delta); j++) {
            distance = euclideanDistance(aux[i], aux[j]);
            if(distance < delta)    {
                delta = distance;
                if(delta < bestDistance) {
                    bestDistance = delta;
                    result1 = aux[i];
                    result2 = aux[j];
                }
```

```
            }
        }
    }
    return delta;
}

bool X_ORDER(Point a, Point b)  {
    return a.x < b.x;
}

int main()  {
    int N, i;
    Point *points, *pointsByY, *aux;

    //freopen("input.txt", "r", stdin);

    scanf("%d", &N);    //Enter the number of points in the plane
    points = new Point[N];
    for(i=0; i<N; i++)  // Enter N points (x, y)
        scanf("%lf %lf", &points[i].x, &points[i].y);

    if(N <= 1) return 0;

    sort(points, points + N, X_ORDER);
    pointsByY = new Point[N];
    for(i=0; i<N; i++)
        pointsByY[i] = points[i];
    aux = new Point[N];

    bestDistance = numeric_limits<double>::infinity();
```

```
    closestPair(points, pointsByY, aux, 0, N-1);

    // print the closest pair of points and their euclidean distance
    printf("%lf %lf\n", result1.x, result1.y);
    printf("%lf %lf\n", result2.x, result2.y);
    printf("%lf\n", bestDistance);

    return 0;
}
```