// A divide and conquer program in C++ to find the smallest distance from a

// given set of points.

#include <iostream>

#include <float.h>

#include <stdlib.h>

#include <math.h>

using namespace std;

// A structure to represent a Point in 2D plane

struct Point

{

int x, y;

};

/\* Following two functions are needed for library function qsort().

Refer: http://www.cplusplus.com/reference/clibrary/cstdlib/qsort/ \*/

// Needed to sort array of points according to X coordinate

int compareX(const void\* a, const void\* b)

{

Point \*p1 = (Point \*)a, \*p2 = (Point \*)b;

return (p1->x - p2->x);

}

// Needed to sort array of points according to Y coordinate

int compareY(const void\* a, const void\* b)

{

Point \*p1 = (Point \*)a, \*p2 = (Point \*)b;

return (p1->y - p2->y);

}

// A utility function to find the distance between two points

float dist(Point p1, Point p2)

{

return sqrt( (p1.x - p2.x)\*(p1.x - p2.x) +

(p1.y - p2.y)\*(p1.y - p2.y)

);

}

// A Brute Force method to return the smallest distance between two points

// in P[] of size n

float bruteForce(Point P[], int n)

{

float min = FLT\_MAX;

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

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

if (dist(P[i], P[j]) < min)

min = dist(P[i], P[j]);

return min;

}

// A utility function to find a minimum of two float values

float min(float x, float y)

{

return (x < y)? x : y;

}

// A utility function to find the distance between the closest points of

// strip of a given size. All points in strip[] are sorted according to

// y coordinate. They all have an upper bound on minimum distance as d.

// Note that this method seems to be a O(n^2) method, but it's a O(n)

// method as the inner loop runs at most 6 times

float stripClosest(Point strip[], int size, float d)

{

float min = d; // Initialize the minimum distance as d

// Pick all points one by one and try the next points till the difference

// between y coordinates is smaller than d.

// This is a proven fact that this loop runs at most 6 times

for (int i = 0; i < size; ++i)

for (int j = i+1; j < size && (strip[j].y - strip[i].y) < min; ++j)

if (dist(strip[i],strip[j]) < min)

min = dist(strip[i], strip[j]);

return min;

}

// A recursive function to find the smallest distance. The array Px contains

// all points sorted according to x coordinates and Py contains all points

// sorted according to y coordinates

float closestUtil(Point Px[], Point Py[], int n)

{

// If there are 2 or 3 points, then use brute force

if (n <= 3)

return bruteForce(Px, n);

// Find the middle point

int mid = n/2;

Point midPoint = Px[mid];

// Divide points in y sorted array around the vertical line.

// Assumption: All x coordinates are distinct.

Point Pyl[mid]; // y sorted points on left of vertical line

Point Pyr[n-mid]; // y sorted points on right of vertical line

int li = 0, ri = 0; // indexes of left and right subarrays

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

{

if (Py[i].x <= midPoint.x && li<mid)

Pyl[li++] = Py[i];

else

Pyr[ri++] = Py[i];

}

// Consider the vertical line passing through the middle point

// calculate the smallest distance dl on left of middle point and

// dr on right side

float dl = closestUtil(Px, Pyl, mid);

float dr = closestUtil(Px + mid, Pyr, n-mid);

// Find the smaller of two distances

float d = min(dl, dr);

// Build an array strip[] that contains points close (closer than d)

// to the line passing through the middle point

Point strip[n];

int j = 0;

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

if (abs(Py[i].x - midPoint.x) < d)

strip[j] = Py[i], j++;

// Find the closest points in strip. Return the minimum of d and closest

// distance is strip[]

return stripClosest(strip, j, d);

}

// The main function that finds the smallest distance

// This method mainly uses closestUtil()

float closest(Point P[], int n)

{

Point Px[n];

Point Py[n];

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

{

Px[i] = P[i];

Py[i] = P[i];

}

qsort(Px, n, sizeof(Point), compareX);

qsort(Py, n, sizeof(Point), compareY);

// Use recursive function closestUtil() to find the smallest distance

return closestUtil(Px, Py, n);

}

// Driver program to test above functions

int main()

{

Point P[] = {{2, 3}, {12, 30}, {40, 50}, {5, 1}, {12, 10}, {3, 4}};

int n = sizeof(P) / sizeof(P[0]);

cout << "The smallest distance is " << closest(P, n);

return 0;

}