**Closest Pair**

**Algorithm: Closest Pair.**

**Input:** An array of n points P[].

**Output:** The smallest distance between the two points.

1. Sort the array according to x-coordinate. O(nlogn) time.
2. Find the middle point in the sorted array, we can take *P[n/2]* as middle point.
3. 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].
4. 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.
5. Find all points whose x coordinate is closer than d to the middle vertical line. Build an array strip[] of all such points.
6. Sort the array strip[] according to y coordinates. This step is O(nLogn)
7. Find the smallest distance in strip[].It can be proved geometrically that for every point in strip, we only need to check at most 7 points after it.
8. Finally return the minimum of d and distance calculated in above step

#include<stdio.h>

struct Point{ int x, y; };

void DIVIDE(int ARRAY[],int low,int high); // Sorting Function

void MERGEALGO(int ARRAY[],int low,int mid,int high); // Sorting Function

float dist(struct Point p1, struct Point p2){

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

float bruteForce(struct Point P[], int n){

float min = 1000.00f;

int i,j;

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

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

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

return min; }

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

float min = d;

int i,j;

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

for (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; }

float closestUtil(int P1[],int P2[], int n){

struct Point Px[n];

struct Point Py[n];

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

Px[i].x = P1[i]; Px[i].y = P2[i]; Py[i].x = P1[i]; Py[i].y = P2[i]; }

if (n <= 3) return bruteForce(Px, n);

int mid = n/2; struct Point midPoint = Px[mid];

struct Point Pyl[mid+1]; struct Point Pyr[n-mid-1];

int li = 0, ri = 0;

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

if (Py[i].x <= midPoint.x) Pyl[li++] = Py[i];

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

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

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

float d = (dl < dr)? dl : dr;

struct Point strip[n];

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

int temp= Py[i].x - midPoint.x;

if(temp<0) temp=temp\*(-1);

if (temp < d) strip[j] = Py[i], j++; }

float local\_result=stripClosest(strip, j, d);

return (d < local\_result)? d : local\_result;

}

float closestpair(struct Point P[], int n){

int Px[n]; int Py[n]; int i;

for (i = 0; i < n; i++){ Px[i] = P[i].x; Py[i] = P[i].y; }

DIVIDE(Px, 0, n - 1); DIVIDE(Py, 0, n - 1);

return closestUtil(Px, Py, n); }

int main(){

struct Point P[] = {{1,2},{5,3},{2,9},{6,5},{1,7}};

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

printf("%f",res); return 0; }