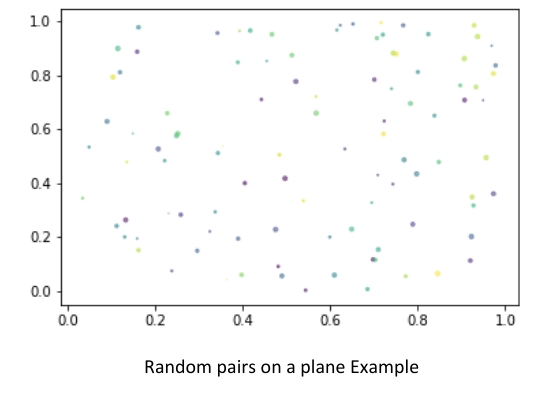
***Problem statement 1***

***G.Tharitha Sririya***

***18bct053***

***Find the closest pair of points:***

***Generate a random pair of 100 values. Apply the closest pair algorithm to find the closest pair (distance). A C++ Implementation from scratch implementation with neat documentation is expected.***

***The time complexity of the algorithm should be O(N log N)***

***Program :***

***#include <bits/stdc++.h>***

***using namespace std;***

***class Point***

***{***

***public:***

***int x, y;***

***};***

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

***{***

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

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

***}***

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

***{***

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

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

***}***

***float dist(Point p1, Point p2)***

***{***

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

***}***

***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;***

***}***

***float min(float x, float y)***

***{***

***return (x < y)? x : y;***

***}***

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

***{***

***float min = d;***

***qsort(strip, size, sizeof(Point), compareY);***

***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;***

***}***

***float closestUtil(Point P[], int n)***

***{***

***if (n <= 3)***

***return bruteForce(P, n);***

***int mid = n/2;***

***Point midPoint = P[mid];***

***float dl = closestUtil(P, mid);***

***float dr = closestUtil(P + mid, n - mid);***

***float d = min(dl, dr);***

***Point strip[n];***

***int j = 0;***

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

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

***strip[j] = P[i], j++;***

***return min(d, stripClosest(strip, j, d) ); }***

***float closest(Point P[], int n)***

***{***

***qsort(P, n, sizeof(Point), compareX);***

***return closestUtil(P, n);***

***}***

***int main()***

***{***

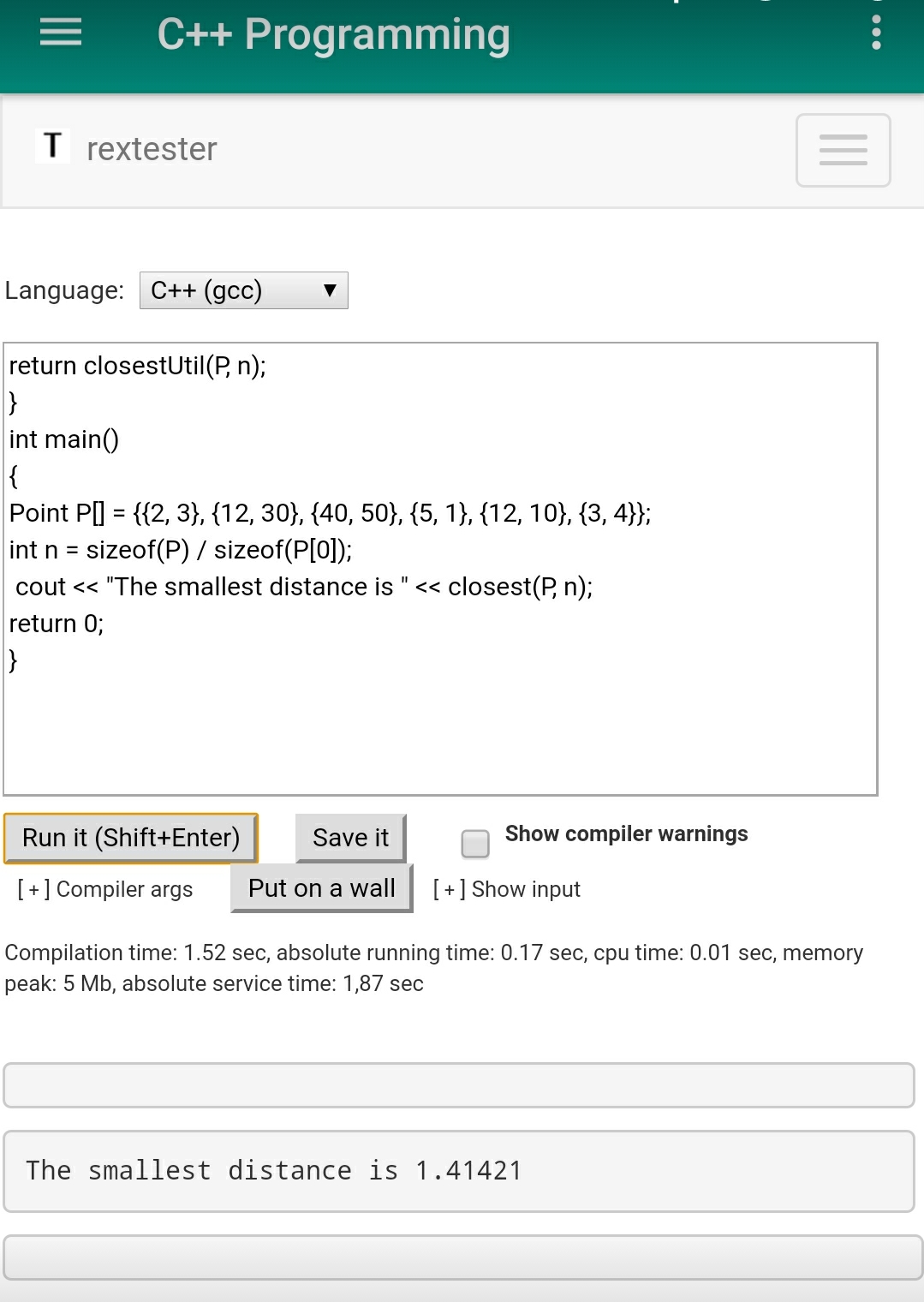
***Point P[] = {{2, 3}, {12, 30}, {40, 50}, {5, 1}, {12, 10}, {3, 4}};***

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

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

***return 0;***

***}***



***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[]. This is tricky. From the first look, 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 (note that strip is sorted according to Y coordinate).***

***7) Finally return the minimum of d and distance calculated in the above step (step 6)***