Closest Pair of Points | O(nlogn) Implementation - GeeksforGeeks

Closest Pair of Points | O(nlogn) Implementation

We are given an array of n points in the plane, and the problem is to find out the closest pair of points in the array. This problem arises in a number of applications. For example, in air-traffic control, you may want to monitor planes that come too close together, since this may indicate a possible collision. Recall the following formula for distance between two points p and q.

$$\|pq\| = \sqrt{(p_x - q_x)^2 + (p_y - q_y)^2}.$$

We have discussed a divide and conquer solution for this problem. The time complexity of the implementation provided in the previous post is O(n (Logn)^2). In this post, we discuss an implementation with time complexity as O(nLogn).

Following is a recap of the algorithm discussed in the previous post.

1) We sort all points according to x coordinates.

2) Divide all points in two halves.

3) Recursively find the smallest distances in both subarrays.

4) Take the minimum of two smallest distances. Let the minimum be d.

5) Create an array strip[] that stores all points which are at most d distance away from the middle line dividing the two sets.

6) Find the smallest distance in strip[].

7) Return the minimum of d and the smallest distance calculated in above step 6.

The great thing about the above approach is, if the array strip[] is sorted according to y coordinate, then we can find the smallest distance in strip[] in O(n) time. In the implementation discussed in previous post, strip[] was explicitly sorted in every recursive call that made the time complexity O(n (Logn)^2), assuming that the sorting step takes O(nLogn) time.
In this post, we discuss an implementation where the time complexity is O(nLogn). The idea is to presort all points according to y coordinates. Let the sorted array be Py[]. When we make recursive calls, we need to divide points of Py[] also according to the vertical line. We can do that by simply processing every point and comparing its x coordinate with x coordinate of middle line.

Following is C++ implementation of O(nLogn) approach.

```
// 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 minimum of two float values
float min(float x, float y)
{
    return (x < y)? x : y;
```

```
}// A utility function to find the distance beween the closest points of// strip of given size. All points in strip[] are sorted accordint 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+1];   // y sorted points on left of vertical line
    Point Pyr[n-mid-1];  // 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)
         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 min(d, stripClosest(strip, j, d) );
}// The main functin 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;
}
```

Output:

```
The smallest distance is 1.41421
```

Time Complexity: Let Time complexity of above algorithm be T(n). Let us assume that we use a O(nLogn) sorting algorithm. The above algorithm divides all points in two sets and recursively calls for two sets. After dividing, it finds the strip in O(n) time. Also, it takes O(n) time to divide the Py array around the mid vertical line. Finally finds the closest points in strip in O(n) time. So T(n) can expressed as follows

$T(n) = 2T(n/2) + O(n) + O(n) + O(n)$

$T(n) = 2T(n/2) + O(n)$

$T(n) = T(nLogn)$

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above