**Problem:**

Find the closest pair of points in a set of points.

**Algorithm 1 (no squares):**

Find(){

* Base condition: if there are only two or one point in this array. Return
* Use the median of the points (x coordinate sort) to split the array into left part and right part.
* Find the closest distance distance1 in the left part. Find the closest distance distance2 in the right part. d = minimum (distance1, distance2);
* Create two rectangle areas representing ( median.point.x-d < x < median.point.x) and ( median point.x<x< median.point.x +d). find the points locating in these two areas respectively.
* According to the y coordinate, sort the points in the left part and right part respectively.
* For each point 1 in the left part: find the close points that satisfy the condition ( point.y – d < y < point.y +d).

compare the distance from the point 1 to these points and find the closest distance3.

Return minimum(distance1, distance2, distance3).}

**Algorithm 2 (squares):**

Find(){

* Base condition: if there are only two or one point in this array. Return
* Use the median of the points (x coordinate sort) to split the array into left part and right part.
* Find the closest distance distance1 in the left part. Find the closest distance distance2 in the right part. d = minimum(distance1, distance2);
* Create two rectangle areas representing ( median point.x-d < x < median point.x) and ( median point.x<x< median point.x +d). find the points locating in these two areas respectively.
* Split the right rectangle area to many squares whose sides are d. Put every point in its corresponding square.
* For each point 1 in the left part:

Find the square to which point 1 relating. Mark it middle-square. Mark the square above the

middle-square as up-square, the square below the middle-square as down-square.

Iterate the point in the up-square, middle square, and down square and find the smallest

distance from point 1 to these points.

* Return minimum(distance1,distance2,distance3).}

**Implementation and Analysis:**

**Algorithm1, Key characteristic:**

Divide the problem into left part and right part. 2\*T(n/2)

Use ArrayList to store the points in the left rectangle and right rectangle. n

According to y coordinates, sort the points in the left rectangle and right rectangle respectively. nlog(n/2) = nlogn -n;

For each point 1 in the left rectangle, use binary search to find the one point in the right rectangle that is close to the point 1, then use the finding point to find other close points. (n/2)(Log(n/2)+5) = n/2logn+2n;

Compare the distance and find the smallest one. constant c

T(n) = 2\*T(n/2) +nlogn +n/2 logn +2n;

Such that T(n) = O(nlognlogn).

**Algorithm2, Key characteristic:**

Divide the problem into left part and right part. 2\*T(n/2)

Use LinkedList to store the points in the left rectangle and right rectangle. n

For right rectangle, use the array of LinkedList (corresponding to square) to store the points in each square. n/2

For each point 1 in the left rectangle, find the corresponding three LinkedList of the array, iterate the points in these LinkedLists and find the smallest distance. c

Compare the distance and find the smallest one. c

T(n) = 2\*T(n/2) +3n/2+c

Such that T(n) = O(nlogn).

**Comparison and analysis:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| n | theoretical time(squares) | Experimental time (squares) | theoretical time(no squares) | Experimental time(no squares) |
| 8 | 12974 | 100144 | 213.9072735 | 39845 |
| 16 | 34598 | 115306 | 760.5591946 | 104728 |
| 32 | 86495 | 244717 | 2376.747483 | 240485 |
| 64 | 207587 | 546910 | 6845.032752 | 274336 |
| 128 | 484370 | 1065610 | 18633.70027 | 622369 |
| 256 | 1107131 | 1967253 | 48675.78846 | 1029291 |
| 512 | 2491044 | 3322363 | 123210.5895 | 2340322 |
| 1024 | 5535654 | 3654528 | 304223.6778 | 2514163 |
| 2048 | 12178438 | 8956485 | 736221.3004 | 4241636 |

**Observation 1：** The growth rate of Experimental time(squares) and that of theoretical time(squares) are not matched; The growth rate of Experimental time(no squares) and that of theoretical time(no squares) are not matched, either. Actually, the growth rates of both experimental lines are matched with O(n).

**Here is my explanation:** the Input points are random initialized, it means the points are evenly distributed in the graph. So the number of points in the left rectangle and right rectangle is proportional to sqrt(n) ( I can prove it, but the page is limited) , so the handle time of these points are O(sqrt(n) logn), It means the T(n) = n;

**Observation 2:** The time complexity of algorithm 2 is less than that of algorithm 1, but algorithm1 is faster than algorithm2.

**Reason:** 1. In algorithm 2, we need to handle three close squares for each point in the left part.

But in algorithm 1, we actually handle two close squares for each point in the left part.

2. In algorithm2, splitting the right rectangle to many squares will cause much cost if the d is

very small. Because initialize every square requires much time.

**Optimization**: In the algorithm2, I don’t initialize all squares. I just declare all the squares. Then every time when I need one square, I initialize one. This optimization improves the performance of the algorithm2 up to a little better than algorithm1. It is amazing!