Zehao Li
CS130B
project01

Run:

make
./prog1 < (…txt)

a,
Design a brute-force algorithm to accomplish ClosestPair(n, X, Y), where n is the number of points, X and Y are arrays of size n which store the x and y coordinates, respectively, of the n points. Analyze the complexity of your algorithm.

For the brute-force algorithm, I simply compare each point with other point. Record the pair of points with mini-distance.
When I go through all the points, I will calculate the distance of current two points and compare this number to mini-distance. If it's smaller than the mini-distance, then I change all the records to current pair of points' information.
At last, I will get the closest pair of points.

```
create the record of closest pair of points;
initial the mini-distance;
for (int i=0; i< n; i++){
        for (int j=0; j<n ; j++){
                i is not j;
                calculate point i and point j distance;
                compare;
                if smaller,
                        update the record of closest pair of points and update the mini-distance;

        }
}
```

For the time complexity, it will go through n-1 points for n points. O(n*(n-1)) = O(n^2)

b,
Design a divide-and-conquer algorithm for ClosestPair(n, X, Y). Your algorithm should have a better performance than the brute-force one. Analyze the complexity of your algorithm.

For the divide-and-conquer algorithm, I process the points array as the following steps.
First, I will sort the array of points base on x. Then divide then in two equal number arrays of points. Find the two min-distance info from two arrays (recursive).
Secondly, find the smaller min-distance info between the known two info structs. Get the d. Get the medium of the original array.

Then, get the mid-part array of points through set the x range from (medium.x - d) to (medium.x+d). Sort this array based on y. Compare each point with 7 points after itself and get the mid-distance info.

Final, see if the mid part min-distance smaller then the min-distance of two divided arrays. And we can have the closest pair.

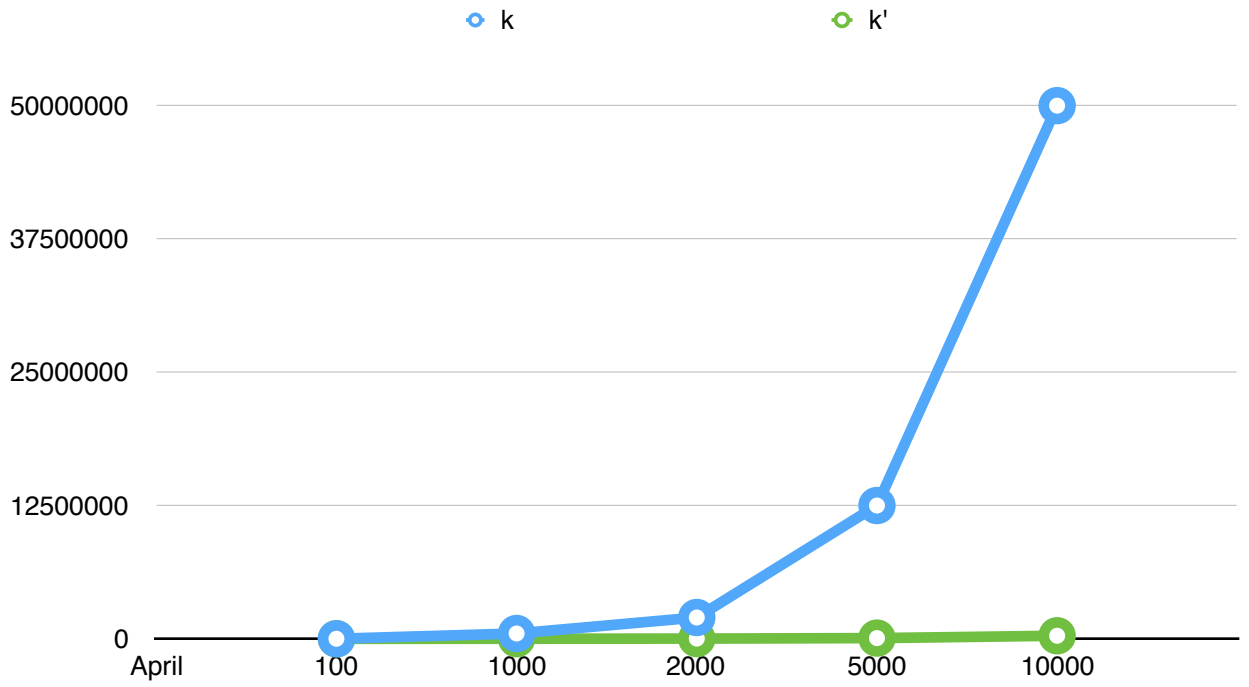
```
sort(vector<Point> original)
        sort original based on x;

        if num of points < 3
                brute force to find the min-distance info;
                return the min-distance info;
        else
                divide the original into two vectors: v1, v2;
                min-distance info1=sort(v1);
                min-distance info2=sort(v2);

                find the smaller one to be the min-distance info_all;
                find the min-distance d, get the medium.x;

                set the x range from (medium.x - d) to (medium.x+d) and get mid-part array m;

                for Point in m
                        compare it with 7 points after itself;

                Get the min-distance info_m of array m;

                Compare info_m with info_all to get the final info_all;
                return info_all;
```

For the time complexity, it will divide in to arrays until reach the base case. After each divide there is a compare. So the time complexity is O (n log n).

After count the k and k' , we could see divide and conquer method k' is much smaller than the brute force method k. Their difference is just similar to their time complexity's difference.

Divide and conquer could save a lot of time comparing to the brute force, especially for large size of data.

Number	**100**	**1000**	**2000**	**5000**	**10000**	**100,000**
k	4950	499500	1999000	12497500	49995000	704982704



Number	100	1000	2000	5000	10000	100,000
k’	318	8828	21298	65901	285929	2244901