

### 629 HW3

**Name:** Wei-Che Hsu

**UIN:** 826001725

**Programming Exercise.** (30 points) Compare the running time of your implementation of the recursive closest-pair algorithm (from HW 2) with the brute-force algorithm for the problem (computing the distance between all pairs of points). You will need to run both implementations on a range of problem sizes, measure the running times, and plot the results. Compare your experimental results to the theoretical analyses. If you have any discrepancies, discuss possible reasons. Turn in your code and your analysis (does not need to be in LaTeX) on eCampus as one zip file.

I use stopwatch method to measure my code execution time. My main program(Program.cs), which implemented by C#, runs as the following description:

Step1: generate pattern

Step2: create stopwatch object (sw)to measure Brute-Force method

Step3: Execute the Brute-Force method

Step4: measure the time from sw.

Step5: create stopwatch object (sw2)to measure Divide and Conquer method

Step6: Execute the Divide and Conquer method

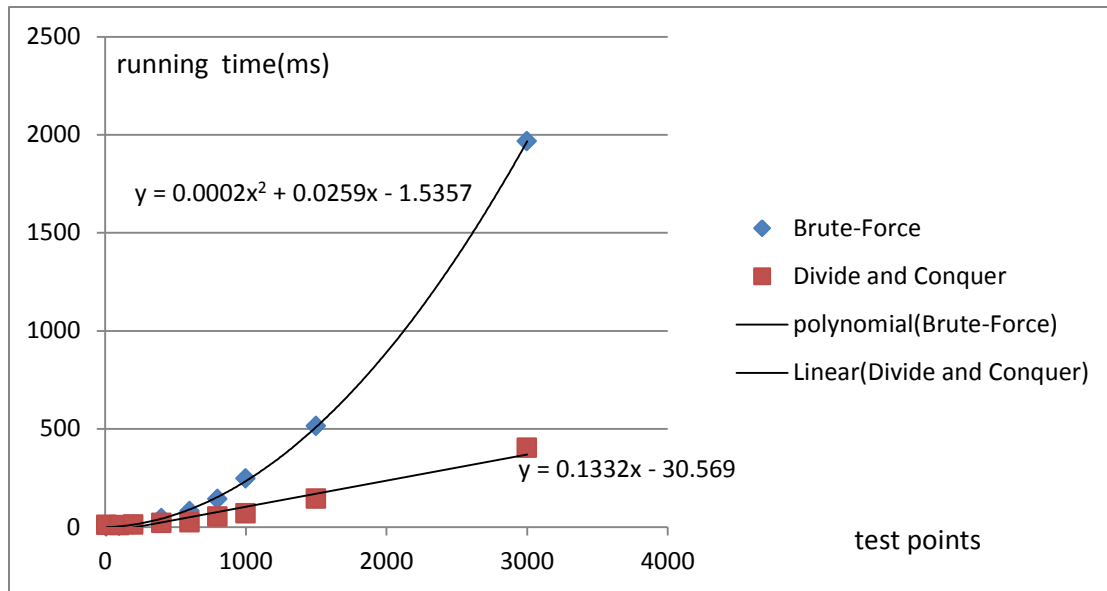
Step7: measure the time from sw2.

I obtain the test result as the below Table:

The number of test points	Brute-Force time (ms)	Divide and Conquer time (ms)
10	1.8781	6.3479
100	4.7543	7.8119
200	12.3031	11.7345
400	44.7902	18.9401
600	80.2221	22.8233
800	141.3202	51.5972
1000	245.7713	68.6091

1500	514.157	144.5153
3000	1965.786	402.965

Also, the plot and the estimated equation is shown as below:



**Brute Force method** is  $O(n^2)$  in theoretical analysis. In my test result,  $O(n^2)$  holds when  $n$  is larger than 200, when  $n$  is small case ( $n < 200$ ), the execution time increasing trend is less than  $O(n^2)$  when  $n$  becomes double. It would be due to optimization at program-execution stage with only small data size input, so the execution commands is not the same as I thought on the paper work. Overall, the trend for Brute Force method is  $O(n^2)$ .

**Divide and Conquer method** is  $O(n \lg n)$  in theoretical analysis. In my test result, the trend is little larger than  $O(n \lg n)$ . For example, when the number of test points from 1500 to 3000, the running time of 3000 points is 2.78 times than that of 1500 points. If  $n = 1500$ ,  $1500 * \lg 1500 = 15826$ . If  $n = 3000$ ,  $3000 * \lg 3000 = 34652$ . The running would be  $34652 / 15826 = 2.19$  in theoretical calculation. That's because my program takes some time during divide process. The  $x$  points set is pre-sorted very well for  $P_x$ , but  $P_y$  in my program should be selected from  $P_x$  when each divide operation.

$P_x$  and  $P_y$  as the lecture note's definition in Figure 2.

## D&C Algorithm for Closest Pair: Implementation Notes

- Before calling recursive code, preprocess:
  - sort P into array PX by increasing x-coordinate
  - sort P into array PY by increasing y-coordinate
- Use PX to efficiently divide P into half w.r.t. x-coordinates
- Use PY to efficiently scan up the 2d-wide center strip

CSC6 629, Spring 2017, Set 3

23

Figure 2

**Now I need to analyze my program in DCFinder.cs, which operates Divide and Conquer method, for detailed discussion:**

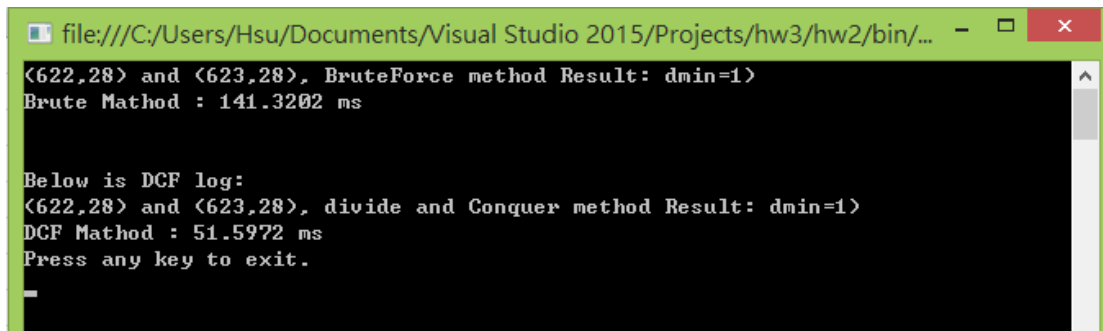
When the program splits Py into Py\_left or Py\_right, I would do two things for all Py points, first check this point exist in Px\_left, and then counting for this point number in Px\_left to make sure that the repeated points(with same x,y value) can be assign to the correct group. However, it would cost near  $O(n^2)$ , because of “in the for loop work” and “do the list count calculation”. But C# have smart decision feature, if the first condition which this point exist in Px\_left is false, it would go else case directly, so it has half chance only taking  $O(n \lg n)$ , here I assume the “Exist” method using binary search taking  $O(\lg n)$ . It should be noticed that n in the last sentence is list’s element count, which would become smaller when divide progress works in the deeper depth. Therefore, **the execution time would between  $O(n \lg n)$  and  $O(n^2)$** . Although this range is loose, it could explain why the running time of Divide and Conquer method is larger than  $O(n \lg n)$  but smaller than  $O(n^2)$ , when n is large enough. ( $n > 200$ )

```
private void write_Y_PointsToSubPoints(List<xyPairs> leftSubOrderY_Li, List<xyPairs> rightSubOrderY_Li, List<xyPairs> leftSub
{
    for (int i = 0; i < this.orderY_Li.Count; i++)
    {
        //condition2 decides the same (x,y) points location.
        // # points in leftSubOrderX_Li = # points in leftSubOrderY_Li
        if ((leftSubOrderX_Li.Exists(point => (point.x == orderY_Li[i].x) && (point.y == orderY_Li[i].y)))
            && (leftSubOrderX_Li.Count(point => (point.x == orderY_Li[i].x) && (point.y == orderY_Li[i].y)) >
                (leftSubOrderY_Li.Count(point => (point.x == orderY_Li[i].x) && (point.y == orderY_Li[i].y)))) )
            leftSubOrderY_Li.Add(orderY_Li[i]);
        else
            rightSubOrderY_Li.Add(orderY_Li[i]);
    }
    return;
}
```

Finally, **comparing these two methods** when the input size  $< 200$ , the Brute-Force method can take advantage because Divide and Conquer takes much time on divide and

combine process, which seems not easily optimized at program-execution stage. However, when the  $n$  becomes larger than 200, the benefit of Divide and Conquer method appears, and it can save much time from distance calculation part when  $n$  becomes larger and larger.

Below picture is my code execution result at 800 test-point case:



```
file:///C:/Users/Hsu/Documents/Visual Studio 2015/Projects/hw3/hw2/bin/... - □ ×
<622,28> and <623,28>, BruteForce method Result: dmin=1>
Brute Method : 141.3202 ms

Below is DCF log:
<622,28> and <623,28>, divide and Conquer method Result: dmin=1>
DCF Method : 51.5972 ms
Press any key to exit.
_
```

The program is almost the same as the program in hw2, I only add the time measurement in Program.cs. If user wants to change points, just modify the variable “`const int testPointsNum = 100;`” in PatternGenerate.cs.

**[Reference]** Figure2 is from 629 lecture notes PPT.