**Programming Project**

**Finding the Closest Pair of Points Problem: Analysis of Two Algorithms**

Amparo Godoy Pastore

July 21, 2024

**Introduction**

This project covers the implementation of two algorithms (ALG1 and ALG2) solving the problem of Finding the Closest Pair of Points. ALG1 is a Brute-Force algorithm that has RT = Θ(n2), while ALG2 is a Divide-and-Conquer algorithm with RT = Θ(nlog2n).

**Problem definition**

Given *n* distinct points P1, P2, …, Pn in a two-dimensional Cartesian plane, the task is to find the pair of points that is closest together. For this project, we will measure the RT for the following input sizes: n=5000, n=10000, n=15000, n=20000, …, n=50000. More specifically, n takes 10 values from 5k to 50k with increment of 5k.

This problem is crucial in several real-world applications, including computer graphics, where it can optimize rendering algorithms, in geographic information systems for spatial analysis, and in molecular modeling to find closest contacts between atoms (Kleinberg and Tardos, 2006). Historically, while a brute-force solution with a time complexity of Θ(n2) was straightforward, more sophisticated algorithms with time complexity of Θ(nlog2n) have been developed to handle larger datasets more efficiently. In this project we analyze two solutions. The brute-force algorithm computes the distance between every pair of distinct points and returns the indexes of the points for which the distance is the smallest. The divide-and-conquer algorithm divides the point set P evenly into Q and R by the line L and recursively finds the closest pair among the points in Q and among the points in R until the overall solution is found from the subproblems.

**Algorithm and RT Analysis**

ALG1 Pseudocode

**BruteForceClosestPoints(P)**

// P is a list of n points, n ≥ 2, P1 = (x1, y1), …, Pn = (xn, yn)

// returns the index1 and index2 of the closest pair of points

dmin = ∞

**for** i = 1 to n – 1 **//O(n)**

**for** j = i + 1 to n **//O(n)**

**if** d < dmin

dmin = d; index1 = i; index2 = j

return index1, index2

**RT = O(n2)**

ALG2 Pseudocode

**DivideAndConqClosestPair (P)**

construct Px and Py **//O(nlogn)**

(px, py) = DivideAndConqClosestPair-Rec(Px, Py)

**DivideAndConqClosestPair**-**Rec(Px, Py)**

**if** |P| ≤ 3

find the closest pair by measuring all pairwise distances

construct Qx, Qy, Rx, Ry **//O(n)**

(q0, q1) = DivideAndConqClosestPair-Rec(Qx, Qy)

(r0, r1) = DivideAndConqClosestPair-Rec(Rx, Ry)

δ = min(d(q0, q1), d(r0, r1))

x\* = maximum x-coordinate of a point in set Q

L = {(x, y): x = x\*}

S = points in P within distance δ of L

construct Sy **//O(n)**

**for** each point s ∈ Sy **//O(n)**

compute the distance from s to each of the next 15 points in Sy

let s, s’ be the pair with the minimum distance

**if** d(s, s’) < δ

return (s, s’)

**else if** d(q0, q1) < d(r0, r1)

return (q0, q1)

**else**

return (r0, r1)

**RT = O(nlogn)**

**Experimental Results**

The experiment consisted of measuring the RT for the input sizes specified above. The RT was measured for each algorithm on *m* = 10 different arrays for each *n* and the average of each is reported in the EmpiricalRT column of *Table 1* and *Table 2.* The theoretical complexity is computed looking at the pseudocode. Below the hidden constant *c* of each algorithm is computed using a “dynamic” analysis.

*Table 1. Computing constant c1 for the ALG 1*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Table ALG1** | | | | | | |
| n | Theoretical RT n^2 | | EmpiricalRT (msec) | | Ratio = (EmpiricalRT)/(TheoreticalRT) | Predicted RT |
| 5000 | 25000000 | | 8032.9 | | 0.000321316 | 9236.039 |
| 10000 | 100000000 | | 28991.6 | | 0.000289916 | 36944.156 |
| 15000 | 225000000 | | 62223.9 | | 0.000276551 | 83124.351 |
| 20000 | 400000000 | | 112645 | | 0.000281613 | 147776.624 |
| 25000 | 625000000 | | 206260.5 | | 0.000330017 | 230900.975 |
| 30000 | 900000000 | | 327841.3 | | 0.000364268 | 332497.404 |
| 35000 | 1225000000 | | 448560.2 | | 0.000366172 | 452565.911 |
| 40000 | 1600000000 | | 583760.6 | | 0.00036485 | 591106.496 |
| 45000 | 2025000000 | | 748028.3 | | 0.000369397 | 748119.159 |
| 50000 | 2500000000 | | 923603.9 | | 0.000369442 | 923603.9 |
| **c1 =** | | 0.000369442 | |

*Table 2. Computing constant c2 for the ALG 2*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Table ALG2** | | | | | | |
| n | Theoretical RT nlgn | | | EmpiricalRT (msec) | Ratio = (EmpiricalRT)/(TheoreticalRT) | Predicted RT |
| 5000 | 61438.5619 | | | 117 | 0.001904341 | 140.5089491 |
| 10000 | 132877.1238 | | | 242 | 0.001821231 | 303.8877287 |
| 15000 | 208090.1232 | | | 366.1 | 0.001759334 | 475.8985828 |
| 20000 | 285754.2476 | | | 516.7 | 0.001808197 | 653.5151181 |
| 25000 | 365241.0119 | | | 835.3 | 0.002286983 | 835.3 |
| 30000 | 446180.2464 | | | 1007.9 | 0.002258953 | 1020.406657 |
| 35000 | 528327.3556 | | | 1199.2 | 0.002269805 | 1208.275702 |
| 40000 | 611508.4952 | | | 1372.3 | 0.002244123 | 1398.509558 |
| 45000 | 695593.6821 | | | 1542 | 0.002216811 | 1590.810954 |
| 50000 | 780482.0237 | | | 1710.4 | 0.002191466 | 1784.949152 |
| **c2 =** | | 0.002286983 |

The following figures compare EmpiricalRT and PredictedRT. In *Figure 1*, we can see how much slower the EmpiricalRT of ALG2 is compared to ALG1. *Figures 2* and *3* show the correspondence of EmpiricalRT and PredictedRT for ALG1 and ALG2, respectively.

*Figure 1. ALG1: EmpiricalRT and ALG2: EmpiricalRT*

*Figure 2. ALG1: EmpiricalRT and ALG1:PredictedRT*

*Figure 3. ALG2: EmpiricalRT and ALG2:PredictedRT*

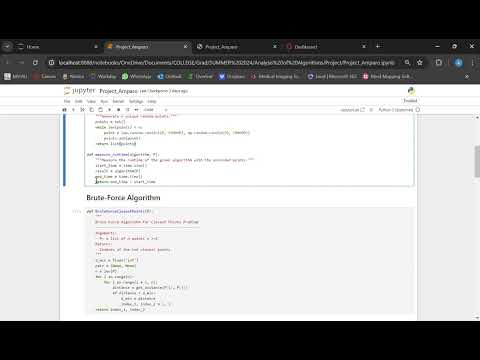
**Conclusions**

The experimental results for both algorithms, Brute Force and Divide and Conquer, align well with their theoretical analyses. For ALG1, the empirical runtime shows a quadratic growth, consistent with its theoretical O(n2) complexity. As *n* increases, the EmpiricalRT of ALG1 aligns more with the PredictedRT. Similarly, for ALG2, the empirical runtime reflects the theoretical O(nlog2n) complexity, aligning better with its prediction as *n* increases. As shown in *Figure 1* this logarithmic growth is much slower than the quadratic growth of ALG1. The close match between EmpiricalRT and PredictedRT, along with the computed constants *c1* and *c2,* confirms the accuracy of the theoretical models. These constants produced accurate predictions of empirical runtimes, further supporting the reliability of the theoretical analysis.

Although the theoretical analysis does not account for practical factors such as hardware (processor speed, system architecture, etc.) and the choice of programming language and compiler, this experiment has shown that the theoretical and empirical results are consistent, underscoring the validity of the runtime complexity assessments for both algorithms.

**Project Demo**

The following is a demo of the source code and test case implementation: <https://youtu.be/KoByPiD2Bdk>

**[](https://www.youtube.com/embed/KoByPiD2Bdk?feature=oembed)**

**References**

*Algorithm Design*, J. Kleinberg and E. Tardos, Addison-Wesley Publishing Company, 2006. [https://theswissbay.ch/pdf/Gentoomen%20Library/Algorithms/Algorithm%20Design%20-%20John%20Kleinberg%20-%20%C3%89va%20Tardos.pdfLinks to an external site.](https://theswissbay.ch/pdf/Gentoomen%20Library/Algorithms/Algorithm%20Design%20-%20John%20Kleinberg%20-%20%C3%89va%20Tardos.pdf)