<u>Finding the Closest Pair of Points</u>
<u>Programming Project</u>

Submitted by

Saivineeth Suram

(Z23704703)

Problem Definition

This project aims to compare a brute-force method (ALG1) and an optimized divide-and-conquer strategy (ALG2) for determining the shortest distance between a set of 2D points. This analysis is focused on the well-known Closest Pair of Points problem. The dataset sizes range from 10,000 to 100,000, increasing in increments of 10,000 for each test.

The brute-force method operates with a quadratic time complexity of (O(n^2)). In contrast, ALG2, which implements a divide-and-conquer tactic, has a time complexity of (O(nlogn)). This approach was pioneered by Michael Shamos and Dan Hoey in 1975, although the concept of divide and conquer in this context dates back to John von Neumann in 1945. In ALG2, merge sort is employed to sort the arrays. The textbook's divide-and-conquer method is followed, which can be enhanced to (O(n)) using randomization techniques.

This foundational problem in computational geometry has broad applications across various fields, including sensor localization, autonomous vehicle navigation, obstacle avoidance, robotic motion planning, molecular structure modeling, and nearest-neighbor classification.

Algorithms and RT Analysis

Let's explore the Running Time Analysis and Pseudocode for both the algorithms.

ALG1

In ALG1, we apply a brute force method, utilizing two nested loops to compute the distance between each pair of points. Below is the pseudocode for this algorithm:

Algorithm Brute Force Closest points(P)

// P is a list of n points, n ≥ 2, P_1= (x_1, y_1) ,….., P_n= (x_n , y_n)

// returns the $index_1$ and $index_2$ of the closest pair of points

d_{min} = ∞

for i = 1 to n-1

    for j = i+1 to n

        d = $\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$

        if d < d_{min}

d_{min}= d;

$index_1$= i;

$index_2$= j;

return $index_1$, $index_2$;

Time complexity is O (n^2)

ALG2

In Algorithm 2, I utilized the divide-and-conquer strategy outlined in the course materials. The accompanying pseudo code provides a high-level overview of the method. However, it is essential to note that this pseudo code is abstract and does not delve into the intricate details of the algorithm's implementation**.**

Closest-Pair(P)

Construct P_x and P_y // O(nlogn)

(P_0^*, P_1^*) = Closest-Pair-Rec (P_x, P_y)

Closest-Pair-Rec (P_x, P_y)

if |P| ≤ 3

find the closest pair by measuring all pairwise distances

construct Q_x, Q_y, R_x, R_y // O(n)

(q_0^* , q_1^*) = Closest-Pair-Rec (Q_x , Q_y)

(r_0^* , r_1^*) = Closest-Pair-Rec (R_x , R_y)

δ = min(d (q_0^* , q_1^*), $d(r_0^*, r_1^*)$)

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 S_y // O(n) time

for each point s ∈ S_y // O(n)

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

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

if d (s, s') < δ

return (s, s')

else if d $(q_0^*, q_1^*) < d(r_0^*, r_1^*)$

return (q_0^*, q_1^*)

else

return (r_0^*, r_1^*)

endif

Time complexity is O(nlogn)

The assumption is that no two points share the same x or y coordinates. For a list of 1D points, sorting the array allows us to calculate the distance between each point and its immediate neighbor, ensuring that these distances are minimized as the points are sorted.

In the case of 2D points, a divide and conquer method is employed where the points are split into two groups, left and right. Within each group, the closest pairs are identified recursively, efficiently solving more minor instances of the problem. Initially, all points in list P are sorted by their x-coordinate and then by their y-coordinate, resulting in two lists, P_x and P_y, respectively. Similarly, the points are divided into two halves, Q (left) and R (right), which are also sorted into lists Q_x, Q_y, R_x, and R_y based on x and y coordinates. The closest pairs within these halves are determined through recursive calculations.

After addressing all sub-problems for the left and right halves, the method calculates the minimum distance between all possible pairs of points across these subsets, ultimately selecting the shortest distance from these calculations.

Experimental Results

ALG1

n	TheoreticalRT n^2	EmpiricalRT (msec)	Ratio = (EmpiricalRT)/(TheoreticalRT)	Predicted RT
10^4	10^8	255.77	$r_1 = 2.56 * 10^{-6}$	428.13
$2*10^4$	$4*10^8$	1238.44	$r_2 = 3.10 * 10^{-6}$	1712.50
$3*10^4$	$9*10^8$	2897.46	$r_3 = 3.22 * 10^{-6}$	3853.13
$4*10^4$	$16*10^8$	6850.01	$r_4 = 4.28 * 10^{-6}$	6850.01
$5*10^4$	$25*10^8$	10173.92	$r_5 = 4.07 * 10^{-6}$	10703.14
$6*10^4$	$36*10^8$	12446.42	$r_6 = 3.46 * 10^{-6}$	15412.53
$7*10^4$	$49*10^8$	16037.00	$r_7 = 3.27 * 10^{-6}$	20978.16
$8*10^4$	$64*10^8$	21944.12	$r_8 = 3.43 * 10^{-6}$	27400.05
$9*10^4$	$81*10^8$	29666.32	$r_9 = 3.66 * 10^{-6}$	34678.19
$10*10^4$	$100*10^8$	32366.71	$r_{10} = 3.24 * 10^{-6}$	42812.57

Table 1(ALG1 – Brute force)

The table displays the experimental outcomes using the brute force method, and the following formula determines the Predicted RT:

c1 = max (r1, r2, …., r10)
Therefore,
$c_1 = 4.28 * 10^{-6}$
Predicted RT = c1∗TheoreticalRT
Predicted RT = c1∗n2

ALG2

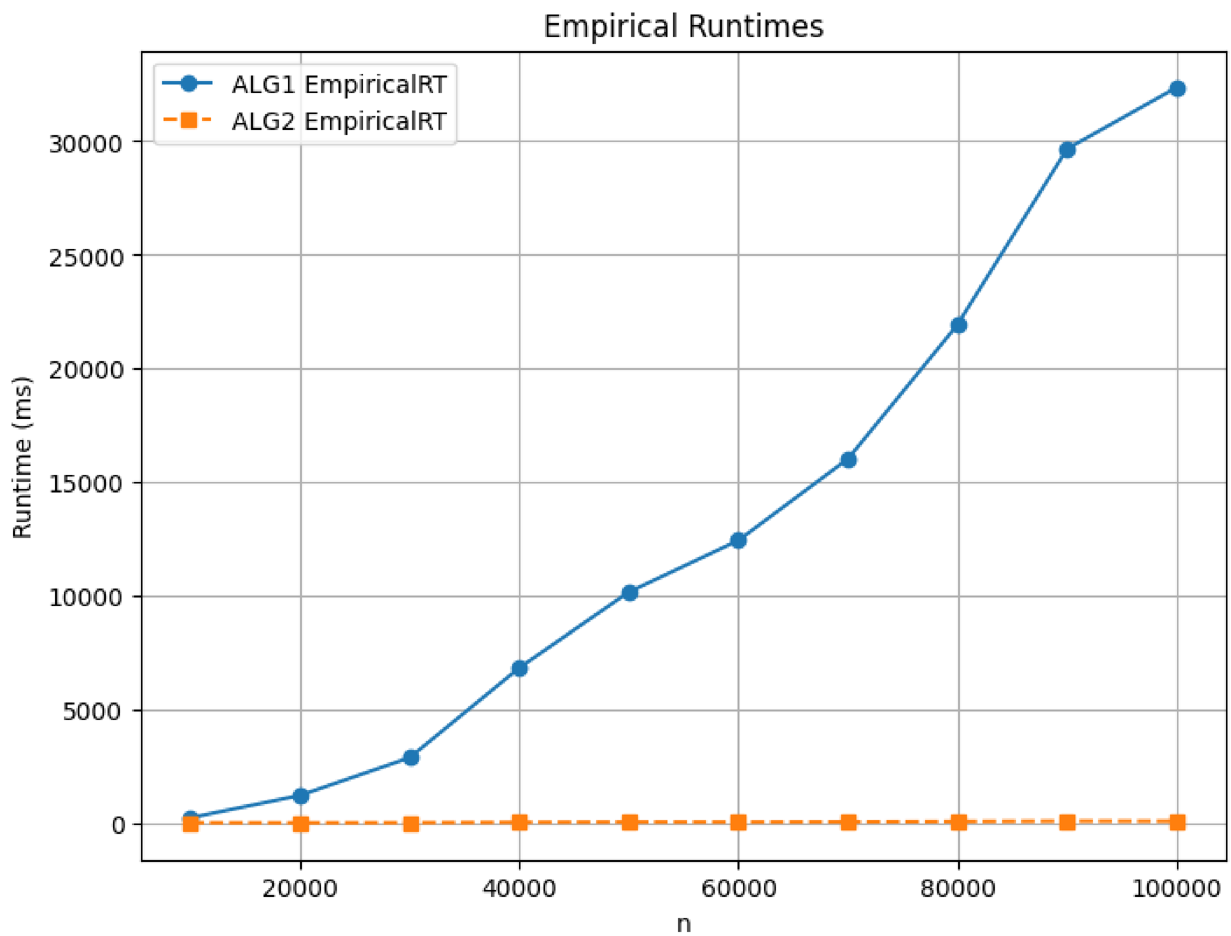
n	TheoreticalRT nlogn	EmpiricalRT (msec)	Ratio = (EmpiricalRT)/(TheoreticalRT)	Predicted RT
10^4	$13.29 * 10^4$	23.43	$r_1 = 1.76 * 10^{-4}$	23.43
$2*10^4$	$28.58 * 10^4$	18.93	$r_2 = 6.62 * 10^{-5}$	50.38
$3*10^4$	$44.62 * 10^4$	25.85	$r_3 = 5.79 * 10^{-5}$	78.66
$4*10^4$	$61.15 * 10^4$	47.89	$r_4 = 7.83 * 10^{-5}$	107.80
$5*10^4$	$78.05 * 10^4$	62.75	$r_5 = 8.04 * 10^{-5}$	137.59
$6*10^4$	$95.24 * 10^4$	60.40	$r_6 = 6.34 * 10^{-5}$	167.89
$7*10^4$	$112.67 * 10^4$	68.09	$r_7 = 6.04 * 10^{-5}$	198.62
$8*10^4$	$130.30 * 10^4$	82.65	$r_8 = 6.34 * 10^{-5}$	229.71
$9*10^4$	$148.12 * 10^4$	102.18	$r_9 = 6.90 * 10^{-5}$	261.12
$10*10^4$	$166.10 * 10^4$	97.76	$r_{10} = 5.89 * 10^{-5}$	292.82

Table 2(ALG2 – Divide and Conquer)

c_2 = max (r1, r2, …., r10)
$c_2 = 1.76 * 10^{-4}$
so r1 is outliner

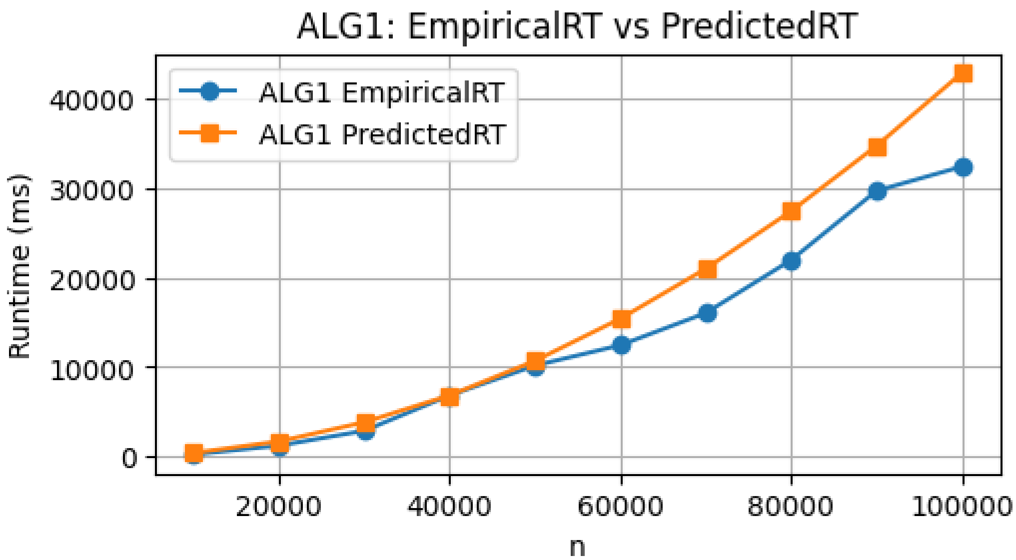
Graphs

A graph where the x-axis represents the variable n with the value ranging from n = 10^4 , 2 * 10^4, 3 * 10^4 ,….,10 * 10^4 . these values denote n = 10000 , 20000, 30000, 40000, 50000, 60000, 70000,80000, 90000, 100000.
Y- axis will display the empirical runtime for both algorithms.
ALG1: Empirical RT ALG2: Empirical RT.



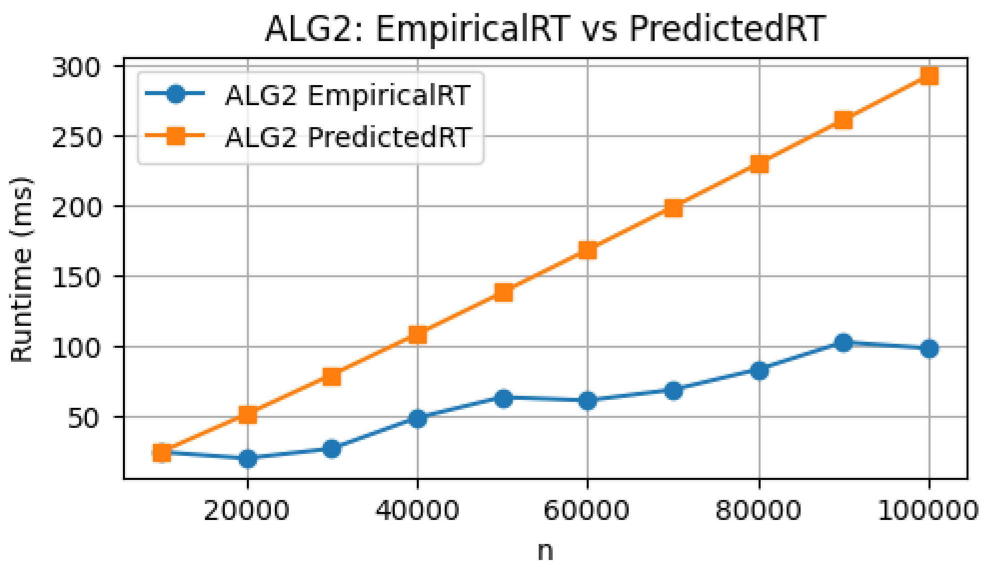
As time complexity of Brute force is O(n^2) , the graph grows exponentially compared to divide-and-conquer algorithm.

A graph were x-axis has values of n = 10^4 , $2 * 10^4$, $3 * 10^4$,….,$10 * 10^4$. Below graph plotted values for ALG1: Empirical RT and ALG2: Predicted RT



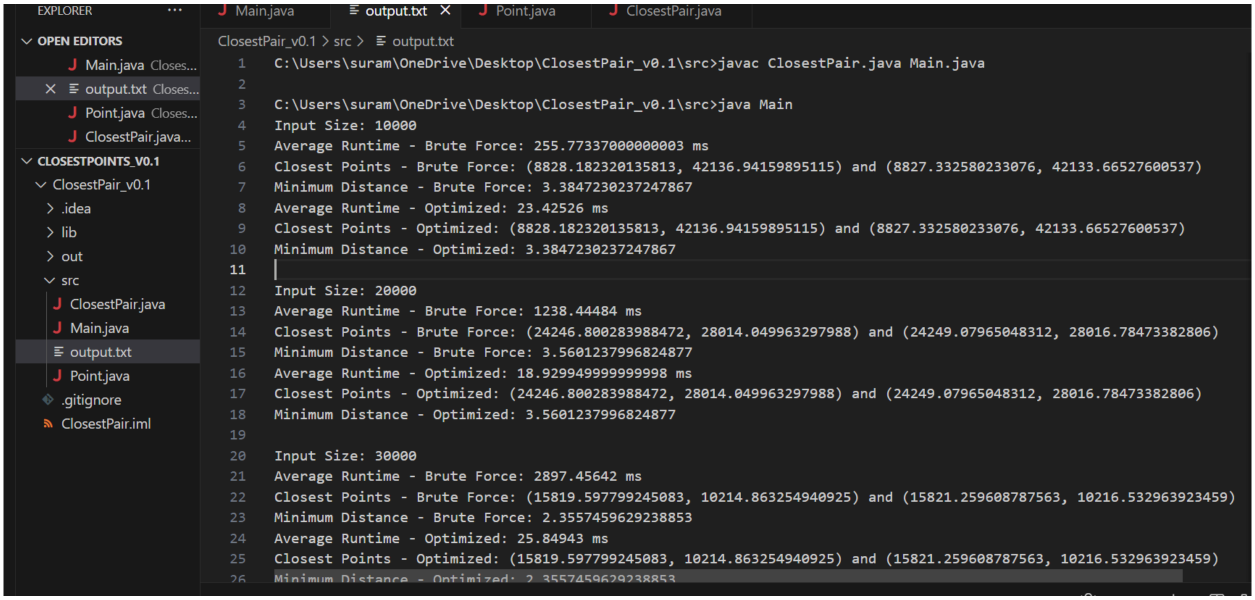
The graph displayed compares the empirical and predicted values of ALG1. It shows that the empirical runtime closely matches the expected runtime. In the graph, time increases exponentially, and the algorithm's complexity is O(n^2).

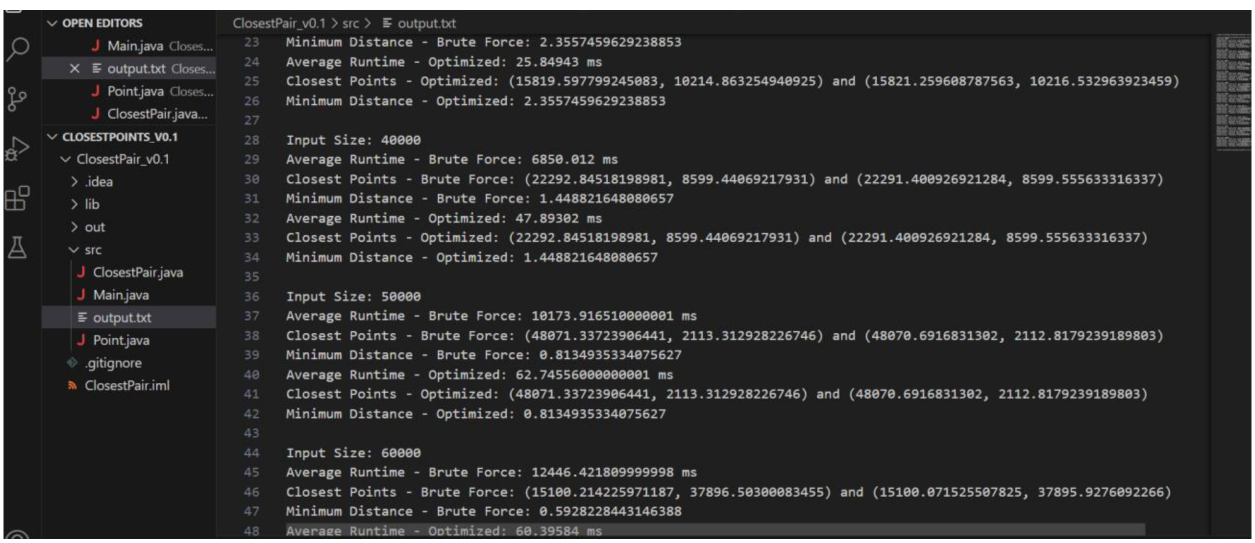
The graph features an x-axis labeled with values n = 10^4, 2 * 10^4, 3 * 10^4 ,….,10 * 10^4. It plots the data points for the empirical runtime (Empirical RT) and the predicted runtime (Predicted RT) of ALG2.

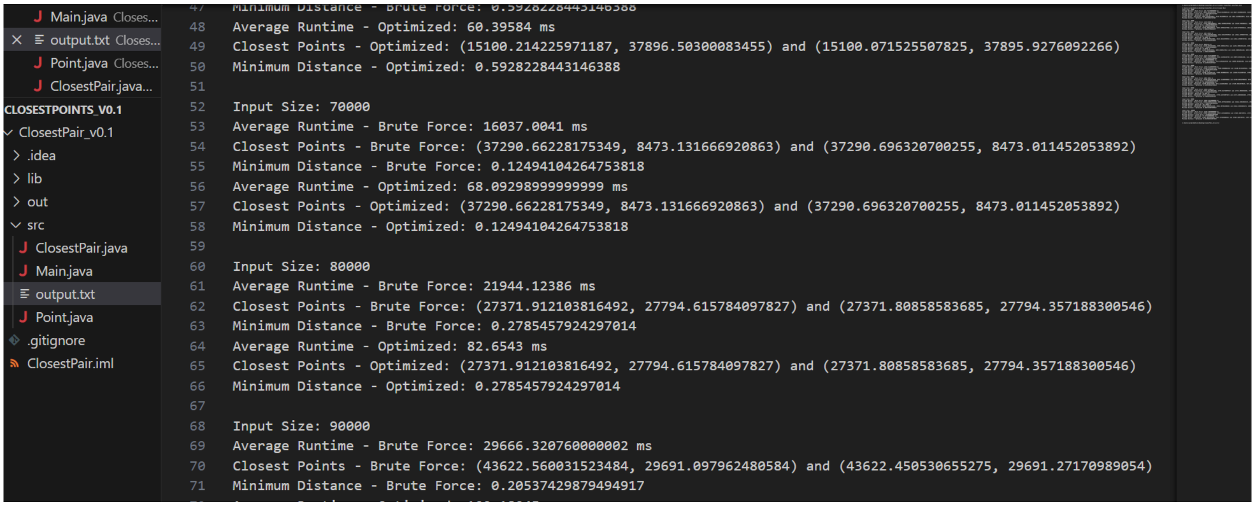


The graph displays the empirical and predicted values of ALG2. It shows that the empirical runtime (Empirical RT) closely traces the predicted runtime (Predicted RT). Notably, the runtime for the input size of 10,000 is viewed as an outlier compared to other input sizes' benchmark results.

Output


EXPLORER
OPEN EDITORS
Main.java
output.txt
Point.java
ClosestPair.java
CLOSESTPOINTS_V0.1
ClosestPair_v0.1
.idea
lib
out
src
ClosestPair.java
Main.java
output.txt
Point.java
.gitignore
ClosestPair.iml
ClosestPair_v0.1 > src > output.txt
C:\Users\suram\OneDrive\Desktop\ClosestPair_v0.1\src>javac ClosestPair.java Main.java
C:\Users\suram\OneDrive\Desktop\ClosestPair_v0.1\src>java Main
Input Size: 10000
Average Runtime - Brute Force: 255.77337000000003 ms
Closest Points - Brute Force: (8828.182320135813, 42136.94159895115) and (8827.332580233076, 42133.66527600537)
Minimum Distance - Brute Force: 3.3847230237247867
Average Runtime - Optimized: 23.42526 ms
Closest Points - Optimized: (8828.182320135813, 42136.94159895115) and (8827.332580233076, 42133.66527600537)
Minimum Distance - Optimized: 3.3847230237247867
Input Size: 20000
Average Runtime - Brute Force: 1238.44484 ms
Closest Points - Brute Force: (24246.800283988472, 28014.049963297988) and (24249.07965048312, 28016.78473382806)
Minimum Distance - Brute Force: 3.5601237996824877
Average Runtime - Optimized: 18.929949999999998 ms
Closest Points - Optimized: (24246.800283988472, 28014.049963297988) and (24249.07965048312, 28016.78473382806)
Minimum Distance - Optimized: 3.5601237996824877
Input Size: 30000
Average Runtime - Brute Force: 2897.45642 ms
Closest Points - Brute Force: (15819.597799245083, 10214.863254940925) and (15821.259608787563, 10216.532963923459)
Minimum Distance - Brute Force: 2.3557459629238853
Average Runtime - Optimized: 25.84943 ms
Closest Points - Optimized: (15819.597799245083, 10214.863254940925) and (15821.259608787563, 10216.532963923459)


OPEN EDITORS
Main.java
output.txt
Point.java
ClosestPair.java
CLOSESTPOINTS_V0.1
ClosestPair_v0.1
.idea
lib
out
src
ClosestPair.java
Main.java
output.txt
Point.java
.gitignore
ClosestPair.iml
ClosestPair_v0.1 > src > output.txt
Minimum Distance - Brute Force: 2.3557459629238853
Average Runtime - Optimized: 25.84943 ms
Closest Points - Optimized: (15819.597799245083, 10214.863254940925) and (15821.259608787563, 10216.532963923459)
Minimum Distance - Optimized: 2.3557459629238853
Input Size: 40000
Average Runtime - Brute Force: 6850.012 ms
Closest Points - Brute Force: (22292.84518198981, 8599.44069217931) and (22291.400926921284, 8599.555633316337)
Minimum Distance - Brute Force: 1.448821648080657
Average Runtime - Optimized: 47.89302 ms
Closest Points - Optimized: (22292.84518198981, 8599.44069217931) and (22291.400926921284, 8599.555633316337)
Minimum Distance - Optimized: 1.448821648080657
Input Size: 50000
Average Runtime - Brute Force: 10173.916510000001 ms
Closest Points - Brute Force: (48071.33723906441, 2113.312928226746) and (48070.6916831302, 2112.8179239189803)
Minimum Distance - Brute Force: 0.8134935334075627
Average Runtime - Optimized: 62.74556000000001 ms
Closest Points - Optimized: (48071.33723906441, 2113.312928226746) and (48070.6916831302, 2112.8179239189803)
Minimum Distance - Optimized: 0.8134935334075627
Input Size: 60000
Average Runtime - Brute Force: 12446.421809999998 ms
Closest Points - Brute Force: (15100.214225971187, 37896.50300083455) and (15100.071525507825, 37895.9276092266)
Minimum Distance - Brute Force: 0.5928228443146388
Average Runtime - Optimized: 60.39584 ms

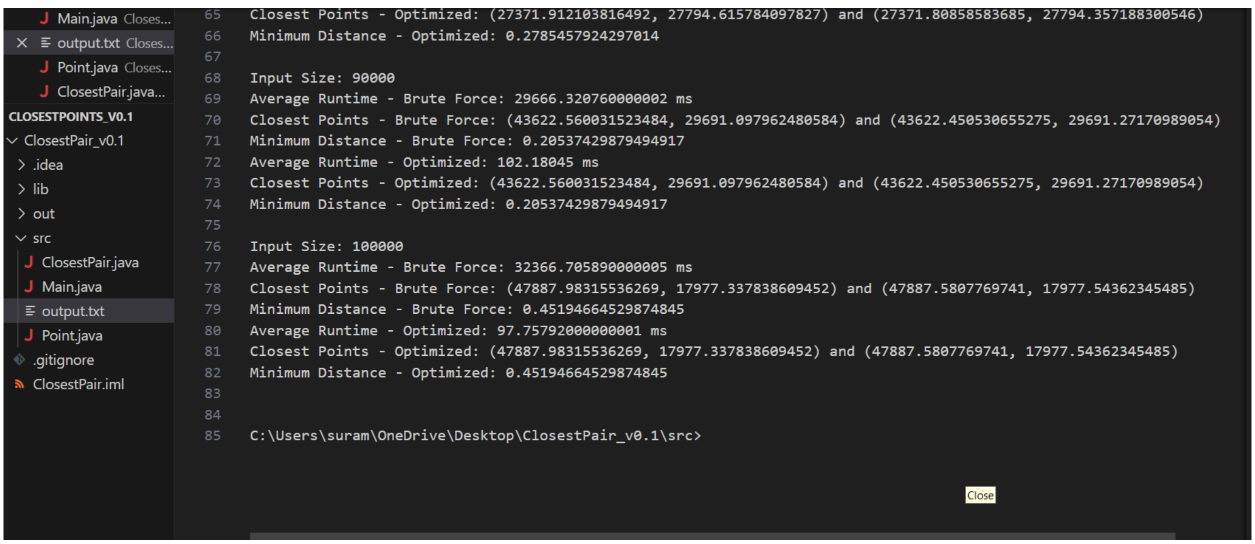



```
Minimum Distance - Brute Force: 0.5928228443146388
Average Runtime - Optimized: 60.39584 ms
Closest Points - Optimized: (15100.214225971187, 37896.50300083455) and (15100.071525507825, 37895.9276092266)
Minimum Distance - Optimized: 0.5928228443146388

Input Size: 70000
Average Runtime - Brute Force: 16037.0041 ms
Closest Points - Brute Force: (37290.66228175349, 8473.131666920863) and (37290.696320700255, 8473.011452053892)
Minimum Distance - Brute Force: 0.12494104264753818
Average Runtime - Optimized: 68.09298999999999 ms
Closest Points - Optimized: (37290.66228175349, 8473.131666920863) and (37290.696320700255, 8473.011452053892)
Minimum Distance - Optimized: 0.12494104264753818

Input Size: 80000
Average Runtime - Brute Force: 21944.12386 ms
Closest Points - Brute Force: (27371.912103816492, 27794.615784097827) and (27371.80858583685, 27794.357188300546)
Minimum Distance - Brute Force: 0.2785457924297014
Average Runtime - Optimized: 82.6543 ms
Closest Points - Optimized: (27371.912103816492, 27794.615784097827) and (27371.80858583685, 27794.357188300546)
Minimum Distance - Optimized: 0.2785457924297014

Input Size: 90000
Average Runtime - Brute Force: 29666.320760000002 ms
Closest Points - Brute Force: (43622.560031523484, 29691.097962480584) and (43622.450530655275, 29691.27170989054)
Minimum Distance - Brute Force: 0.20537429879494917
```




```
Closest Points - Optimized: (27371.912103816492, 27794.615784097827) and (27371.80858583685, 27794.357188300546)
Minimum Distance - Optimized: 0.2785457924297014

Input Size: 90000
Average Runtime - Brute Force: 29666.320760000002 ms
Closest Points - Brute Force: (43622.560031523484, 29691.097962480584) and (43622.450530655275, 29691.27170989054)
Minimum Distance - Brute Force: 0.20537429879494917
Average Runtime - Optimized: 102.18045 ms
Closest Points - Optimized: (43622.560031523484, 29691.097962480584) and (43622.450530655275, 29691.27170989054)
Minimum Distance - Optimized: 0.20537429879494917

Input Size: 100000
Average Runtime - Brute Force: 32366.705890000005 ms
Closest Points - Brute Force: (47887.98315536269, 17977.337838609452) and (47887.5807769741, 17977.54362345485)
Minimum Distance - Brute Force: 0.45194664529874845
Average Runtime - Optimized: 97.75792000000001 ms
Closest Points - Optimized: (47887.98315536269, 17977.337838609452) and (47887.5807769741, 17977.54362345485)
Minimum Distance - Optimized: 0.45194664529874845


C:\Users\suram\OneDrive\Desktop\ClosestPair_v0.1\src>
```

Conclusion

Upon analyzing the experimental results of both algorithms, it is evident that there are only minor variations between the empirical runtime (Empirical RT) and the predicted runtime (Predicted RT) for both algorithms. ALG2 outperforms ALG1 overall, except when dealing with small input sizes, such as less than 150, where ALG1 is quicker. These experiments underscore the significance of both empirical assessment and predictive modeling in evaluating the performance of algorithms. Further investigation into these results can provide deeper insights into the scalability and efficiency of different algorithmic approaches under various conditions. It can aid in optimizing algorithms for specific tasks or operational environments, enhancing their practical utility.

Project Demo Link

https://youtu.be/JPkZqrL0U4Q

References

M. I. Shamos and D. Hoey, "Closest-point problems," *16th Annual Symposium on Foundations of Computer Science (sfcs 1975)*, USA, 1975, pp. 151-162, doi: 10.1109/SFCS.1975.8.

*Algorithm Desig*n, J. Kleinberg and E. Tardos, Addison Wesley, 2005. 209-231.

“Closest Pair Algorithm”, hideoushumpbackfreak ,Internet Blog , https://hideoushumpbackfreak.com/algorithms/algorithms-closest-pair.html

“Closest Pair of Points using Divide and Conquer algorithm”, GeeksForGeeks Blog – Internet, https://www.geeksforgeeks.org/closest-pair-of-points-using-divide-and-conquer-algorithm/ , 13 Feb 2023