Closest Pair of Points Algorithms

Shashidhar Rameshwaram

Z23751768

Srameshwaram2024@fau.edu

1. Defining the Problem:

The Closest Pair of Points problem is a well-known challenge in computational geometry, focusing on identifying the pair of points within a given set that are closest to each other in a two-dimensional plane. Given a list of n points P1, P2, ..., Pn, the objective is to pinpoint the pair of points (Pi, Pj) where the Euclidean distance between them is lowest. And later measure and compare the runtime complexities of the algorithms used to find the closest pair of points.

Input: We're provided with a set of n points P = {(x1, y1), (x2, y2), ..., (xn, yn)} in a 2D plane. Our task is to determine the indices i and j ($1 \leq i, j \leq n$) satisfying the condition:

$d\_min = \min(\text{sqrt}((xi-xj)^2 + (yi-yj)^2))$ for all $i \neq j$, where (xi, yi) and (xj, yj) denote the coordinates of points Pi and Pj, respectively.

Input size: 15000, 20000, 25000, 30000, 35000, 40000, 45000, 50000, 55000, 60000 with 10 iterations each.

Output: The goal is to return the indices i and j of the two closest points, allowing access to the actual points Pi and Pj in the original list and comparing the runtime complexities of algorithms that are used.

Applications in Real-World Scenarios:

- Image Processing: In image analysis, solving the Closest Pair of Points problem assists in identifying similar regions or objects within images, facilitating tasks like pattern recognition and matching.
- Robotics and Navigation: In autonomous robotics and navigation systems, identifying the closest pair of points is crucial for tasks like obstacle avoidance, path planning, and collision detection.
- Computational Biology: Within bioinformatics, this problem is utilized for tasks such as analyzing molecular structures, protein folding, and comparing genetic sequences.
- GIS and Location-Based Services: Geographic Information Systems (GIS) and location-based services frequently encounter numerous geospatial points. Discovering the closest pair of points aids in optimizing routes and locating nearby points of interest.
- Collaborative Filtering: In recommendation systems, tackling the Closest Pair of Points problem helps identify users or items with similar preferences, enabling personalized recommendations.
- VLSI Design: In Very Large-Scale Integration (VLSI) chip design, this problem plays a role in optimizing placement and routing to minimize wire lengths and enhance overall chip performance.

2. Algorithms and RT analysis:

In this project, we will be implementing and analyzing the two algorithms i.e., ALG1:Divide and Conquer and ALG2: Brute Force.

Algorithm 1: Divide and Conquer Algorithm

The algorithm for Divide and Conquer for the closest pair of points is as follows:

// set of points P = {p 1 , p 2 , ..., p n }

- p i has coordinates (x i , y i)
- d(p i, p j) – Euclidean distance between p i and p j
  For any set of points P
- let Px denote the points sorted by increasing the x-coordinate
- let Py denote the points sorted by increasing the y-coordinate
- First level of recursion:
- Q is the “left half” of P – the first [n/2] points in Px
- R is the “right half” of P – the last [n/2] points in Px
- one pass through each of P x and Py in O(n) can create Qx, Qy, Rx, and Ry
- Q x, Rx – points in Q and R sorted in increasing x – coordinate
- Q y, Ry– points in Q and R sorted in increasing y – coordinate
- recursively find the closest pair of points in Q and R
- Let q 0* and q 1* be the closest pair of points in Q
- Let r 0* and r 1* be a closest pair of points in R//

Pseudocode:

```
Closest-Pair(P)
construct Px and Py // O(nlog2 n)
(p0 *, p1 *) = Closest-Pair-Rec(Px, Py)
Closest-Pair-Rec(P x , Py)
if |P| ≤ 3
find the closest pair by measuring all pairwise distances
construct Q x, Q y, Rx, Ry // O(n)
(q0 *, q1 *) = Closest-Pair-Rec(Q x, Q y)
(r 0 *, r 1 *) = Closest-Pair-Rec(Rx, Ry)
δ = min(d(q0 *, q1 *), d(r0 *, 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 Sy // 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(q0 *, q1 *) < d(r0 *, r 1 *)
return (q0 *, q1 *)
else
return (r 0 *, r 1 *)
```

Running time analysis RT1:

$T(n) = 2T(n/2) + cn$

$RT(1) = O(n\log_2 n)$

Algorithm 2: Brute-Force

The algorithm for Brute Force for Closest Points(P)

// P is a list of n points, n ≥ 2, P1 = (x 1 , y 1),..., Pn = (x n , y n)

// returns the index 1 and index 2 of the closest pair of points

Pseudocode:

```
d min = ∞
for i = 1 to n-1
for j = i + 1 to n
d = (sqrt(xi-xj)^2 + (yi-yj)^2)
if d < d min
 d min = d; index 1 = i; index 2 = j
return index 1, index 2
```

Running time analysis

$RT2=O(n^2)$

3. Experimental Results:

The algorithms were implemented using the C++ programming language since C++ has the fastest compiler compared to Java and Python. The provided code presents an experiment designed to evaluate and contrast the empirical running times of two algorithms: brute force and Divide and Conquer, specifically addressing the closest pair problem. The experiment's objective is to analyze the running times of both algorithms across varied input sizes (n), spanning from 15,000 to 600,000 points. The empirical running time signifies the actual duration each algorithm takes to process randomly generated input points.

Additionally, the theoretical running times for both algorithms were computed as follows:

For DivideAndConquer(), the theoretical running time is $O(n * \log_2(n))$.

For BruteForce(), the theoretical running time is $O(n^2)$.

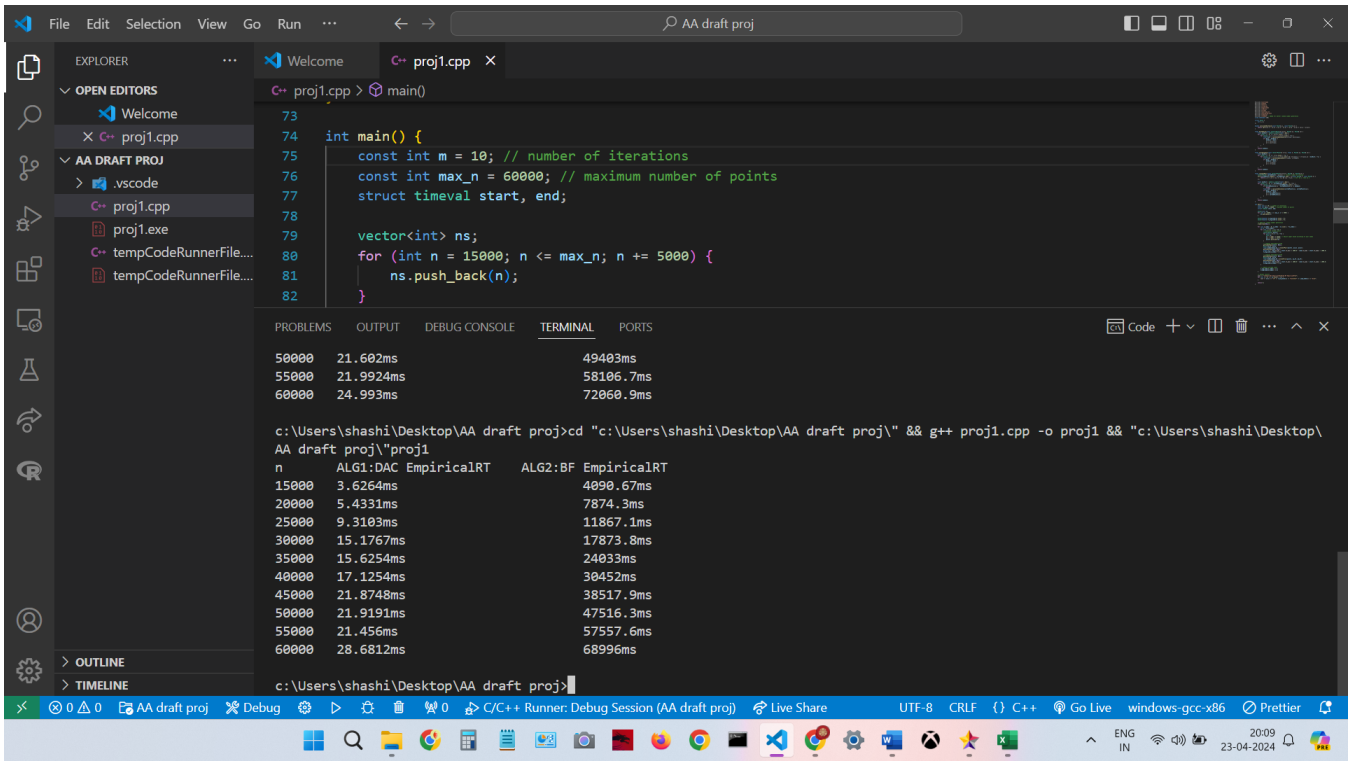
The experiment encompasses arrays of sizes 15,000, 20,000, 25,000, 30,000, 35,000, 40,000, 45,000, 50,000, 55,000, and 60,000 for 10 iterations each.

For each algorithm and input size, the experiment records the following data:

- n: The input size (number of points).
- Theoretical RT: Theoretical RT is based on complexity analysis.

- Empirical RT: Empirical RT, represents the average time taken by the algorithm during the experiment.
- Ratio: Empirical RT divided by Theoretical RT, offering insights into the algorithm's efficiency.
- Predicted RT: Predicted RT, calculated as the maximum ratio value multiplied by Theoretical RT to estimate the running time for larger input sizes.

Output Screenshot:



The experimental results are visualized using MS Excel to generate graphs:

- The initial two graphs depict the empirical and predicted running times for DivideAndConquer() and BruteForce() individually.
- Subsequently, a comparative graph illustrates the empirical running times of both algorithms corresponding to input size (n).

This experiment aims to compare how well two algorithms perform when solving the closest pair problem and to see if their actual running times match up with what the theory predicts. By looking at both the real and predicted running times, researchers can understand how efficient and scalable these algorithms are, especially with bigger sets of data.

Divide and Conquer Algorithm:(ALG1)

RT1(n) = $0(n\log_2 n)$

RT1(n)=c1* $0(n\log_2 n)$

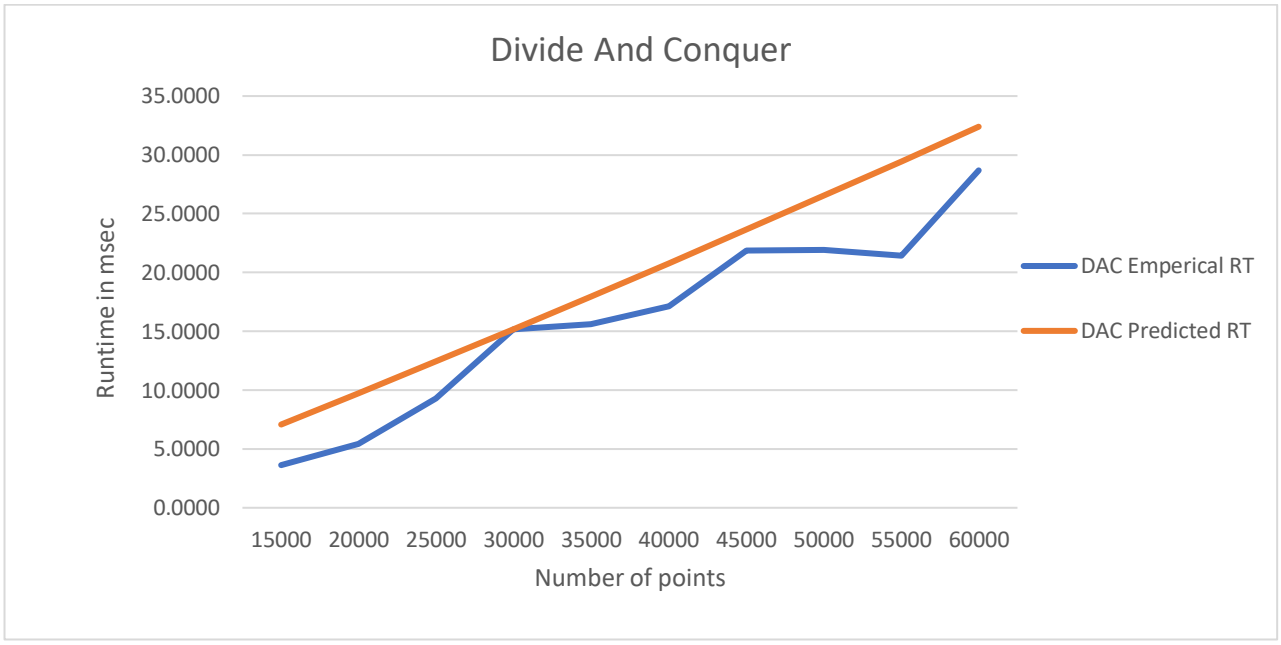
I'm running this program 10 times for every input and calculating the average time taken to execute.

Divide and Conquer				
n	Theoretical RT(msec)= n*Log(n)	Emperical RT(msec)	Ratio=(Emp RT/ Theor RT)	Predicted RT(msec)
15000	208090	3.6264	0.0000174271	7.0781290706
20000	285754	5.4331	0.0000190132	9.7198531856
25000	365241	9.3103	0.0000254908	12.4235739017
30000	446180	15.1767	0.0000340147	15.1767000000
35000	528327	15.6254	0.0000295752	17.9709116251
40000	611508	17.1254	0.0000280052	20.8002955163
45000	695594	21.8748	0.0000314477	23.6604303323
50000	780482	21.9191	0.0000280841	26.5478842346
55000	866093	21.4560	0.0000247733	29.4599156356
60000	952360	28.6812	0.0000301159	32.3942837176

c1=max(r1,r2,….r10)= 0.0000340147

Graph:

ALG1: EmpiricalRT vs PredictedRT



Brute Force Algorithm(ALG2):

RT2(n)=O(n^2)

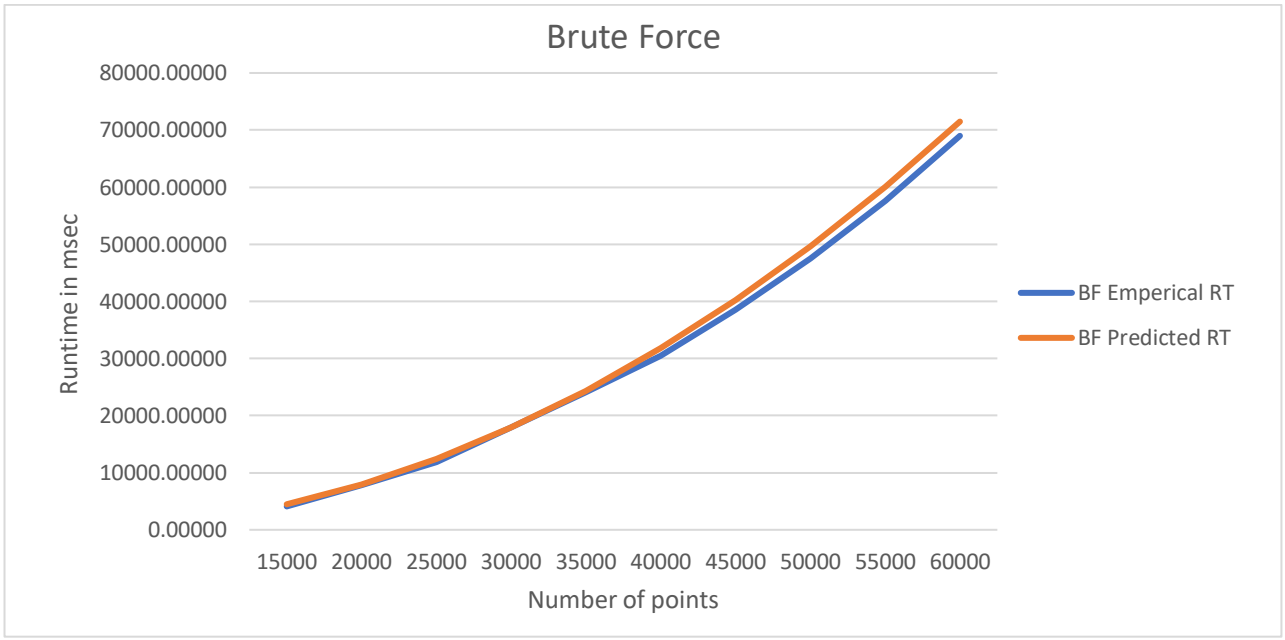
RT2(n)=c2*O(n^2)

Brute Force				
n	Theoretical RT(msec)= n*n	Emperical RT(msec)	Ratio=(Emp RT/ Theor RT)	Predicted RT(msec)
15000	225000000	4090.67000	0.0000181808	4468.2500000000
20000	400000000	7874.30000	0.0000196858	7943.5555555556
25000	625000000	11867.10000	0.0000189874	12411.8055555556
30000	900000000	17873.00000	0.0000198589	17873.0000000000
35000	1225000000	24033.00000	0.0000196188	24327.1388888889
40000	1600000000	30452.00000	0.0000190325	31774.2222222222
45000	2025000000	38517.90000	0.0000190212	40214.2500000000
50000	2500000000	47516.30000	0.0000190065	49647.2222222222
55000	3025000000	57557.60000	0.0000190273	60073.1388888889
60000	3600000000	68996.00000	0.0000191656	71492.0000000000

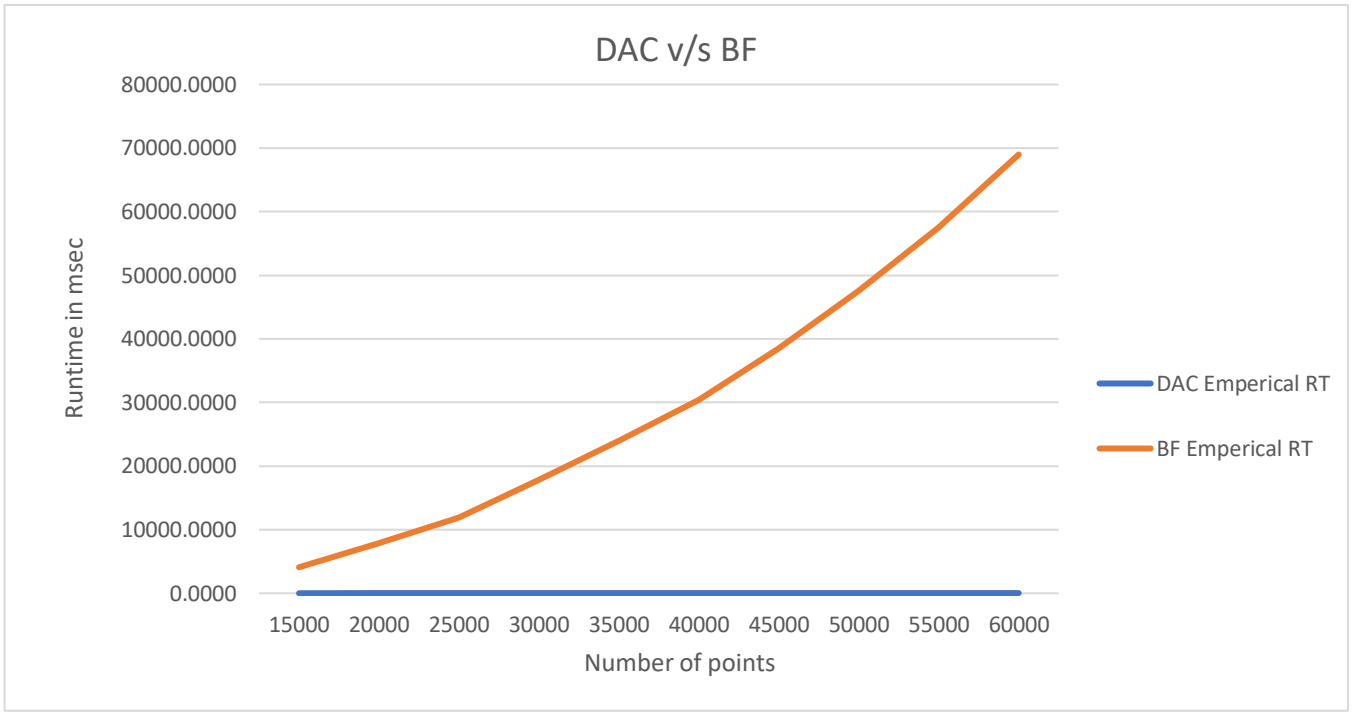
c2=max(r1,r2,….r10)= 0.0000198589

Graph:

ALG2: EmpiricalRT vs PredictedRT



Comparison graph between DAC and BF:



4. Conclusions:

- The graphs clearly demonstrate that the Divide-and-Conquer (ALG1) approach consistently outperforms the Brute-Force (ALG2) approach in terms of time complexity for finding the closest pair of points. ALG1 exhibits a more favorable growth rate as the input size increases.
- Theoretical vs. Empirical Analysis: The theoretical analysis aligns closely with the empirical findings. ALG2 demonstrates a quadratic theoretical running time, while ALG1 shows a logarithmic-linear theoretical running time. These theoretical predictions closely match the actual running times observed.
- Efficiency: Particularly for larger input sizes, the Divide-and-Conquer algorithm (ALG1) proves to be the more efficient choice, especially when compared to the Brute-Force approach (ALG2).
- Consistency of Predictions: The predicted running times are in line with the actual empirical running times, confirming the validity of the theoretical analyses.
- Algorithm Recommendation: Considering both experimental and theoretical results, it is advisable to utilize the Divide-and-Conquer (ALG1) approach for finding the closest pair of points within a set of 2D points to achieve optimal efficiency.

5. Project Demo:

YouTubes link: https://youtu.be/pKsdIZ3c1FI

6. References:

- Algorithm Design, J. Kleinberg and E. Tardos, Addison Wesley, 2006. (Kleinberg&Tardos)
- FAU/COT6405 Analysis of Algorithms (M. Cardei), Module 3 and Module 5.
- Algorithm Design, J. Kleinberg and E. Tardos, Addison Wesley, 2006. (Kleinberg&Tardos)
- Fundamentals of Algorithms, G. Brassard and P. Bratley, Prentice Hall 1996. (Brassard&Bratley)
- The Design & Analysis of Algorithms, 3rd edition, A. Levitin, Addison Wesley, 2007. (Levitin)