DIVIDE AND CONQUER ALGORITHMS PROJECT REPORT

Group Members:
Amna (23K-0066)
Layyana Junaid (23K-0056)
Alisha Zaidi (23K-0025)
Laiba Binte Zia (23K-0006)

Abstract

This project explores a set of classic divide-and-conquer algorithms and demonstrates how they behave on real datasets using a custom visualization tool. We implemented the Closest Pair of Points algorithm, Karatsuba's fast multiplication technique. To help users understand the workflow of these algorithms, we developed an interactive interface that displays each recursive step. We also generated large random inputs to evaluate correctness and performance. Our experiments show that divide-and-conquer provides clear efficiency benefits, especially when input sizes are large. Overall, the project combines theory, coding, visualization, and empirical testing to highlight why divide-and-conquer remains one of the most important strategies in algorithm design.

Introduction

Divide-and-conquer is a general problem-solving approach where a task is divided into smaller pieces, solved recursively, and finally merged into a complete answer. Many well-known algorithms, including merge sort, binary search, and fast multiplication, are built on this idea. The strength of this strategy lies in reducing the problem size quickly, which often leads to better time complexity.

The main purpose of this project was to put divide-and-conquer into practice. Instead of only studying the algorithms theoretically, we implemented them, generated our own input datasets, and compared the results with brute force methods. We focused mainly on two algorithms: the Closest Pair of Points and Karatsuba Integer Multiplication. These were selected because they both showcase how recursion and clever splitting can significantly reduce computational effort. Another major goal of the project was to make the algorithms easier to understand by creating a friendly visualization interface. The interface allows users to upload files, choose an algorithm, and watch each recursive call unfold.

Through these steps, the project allowed us to understand not only how divide-and-conquer works but also why it is used so frequently when performance matters.

Proposed System

Our system is divided into four modules: the input handler, the algorithm engine, the visualizer, and the output generator. Together, these form a pipeline that takes user data, applies a divide-and-conquer algorithm, and displays the results clearly.

1. Input Handling

Users can upload text files or paste data directly into the interface. The application supports the expected formats for closest-pair coordinates and for large integers used in Karatsuba multiplication. Basic validation is performed to ensure the file structure is correct.

2. Algorithm Engine

This is the core of the system. We implemented two major algorithms:

Closest Pair of Points:

- The algorithm first sorts the points by their x-coordinate.
- It then recursively splits the dataset into halves.
- After solving both halves, it checks the narrow band around the midpoint, which is where cross-pairs may exist.
- Thanks to geometric properties, only a small number of comparisons are required in the strip.
- This reduces the overall time complexity to $O(n \log n)$, which is a major improvement over the $O(n^2)$ brute-force approach.

Karatsuba Multiplication:

- Each number is split into a high and a low part.
- Three recursive multiplications are performed (z0, z1, z2).
- These results are then combined to form the final product.
- Karatsuba reduces the number of multiplications needed and runs much faster than traditional long multiplication for very large numbers.

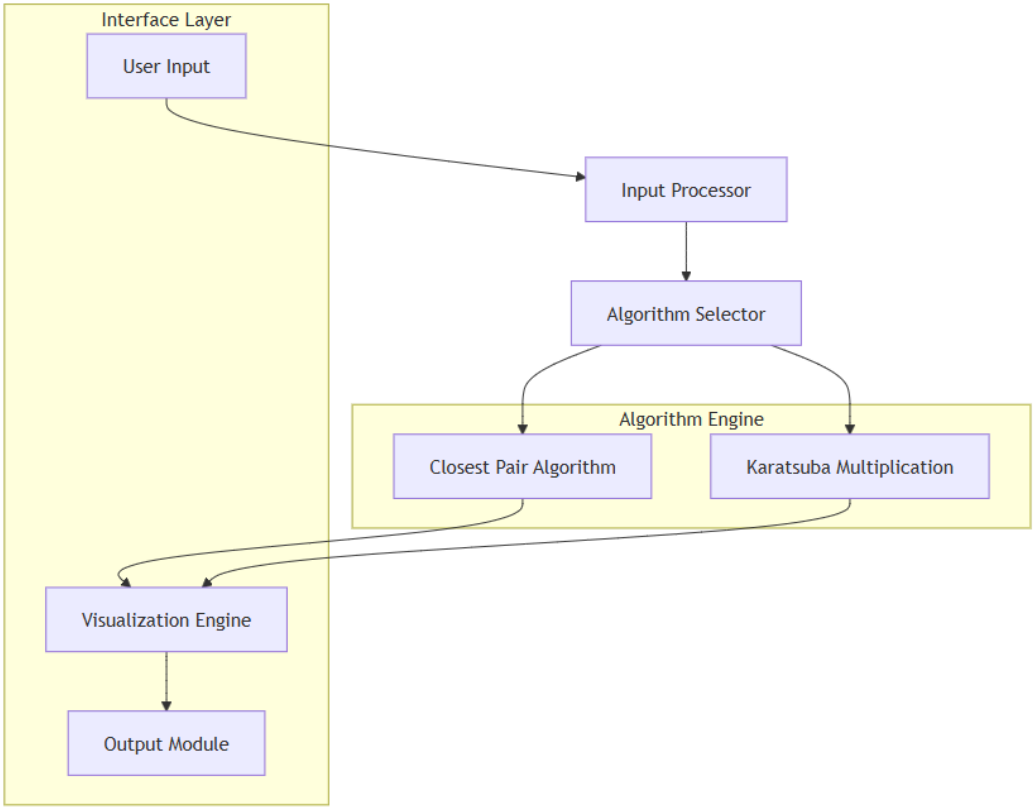
3. Visualization Module

To make the algorithms easier to follow, we built an interactive Streamlit interface. It highlights each recursive split, shows strip comparisons for the closest-pair algorithm, and displays the intermediate computations for Karatsuba. This turned out to be a helpful learning tool because it allows users to see how the algorithm reaches the final answer step by step.

4. Output Generation

After the algorithm finishes, the system displays the final answer, along with optional intermediate steps. For large integer multiplication, the result is summarized rather than printed in full because the outputs can exceed 500 digits.

System Diagram



Experimental Setup

To test our system properly, we generated two sets of input:

1. Closest Pair of Points

- 10 datasets created
- Each file contained 100–300 random (x, y) coordinates
- Values ranged between 0 and 10,000
- The first line stored the number of points, followed by coordinate pairs
- These datasets helped us test both small and medium-sized point sets

2. Integer Multiplication

- 10 datasets created
- Each file contained two integers, each with 100–300 digits
- Digits were generated randomly so that no patterns simplified the multiplication
- These large numbers allowed the Karatsuba algorithm to show its performance advantage

We also used Python's built-in arbitrary-precision integer arithmetic to confirm correctness.

Environment

- Python
- Streamlit
- Matplotlib
- Standard math utilities

The program ran smoothly on a normal laptop without needing special hardware.

Results and Discussion

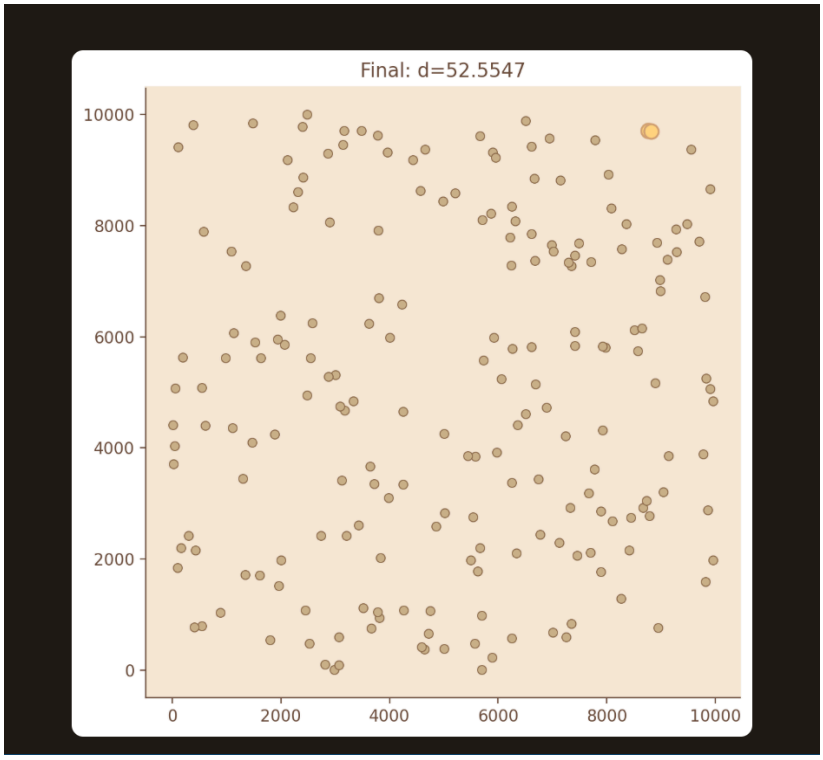
A. Closest Pair of Points (Real Output Values)

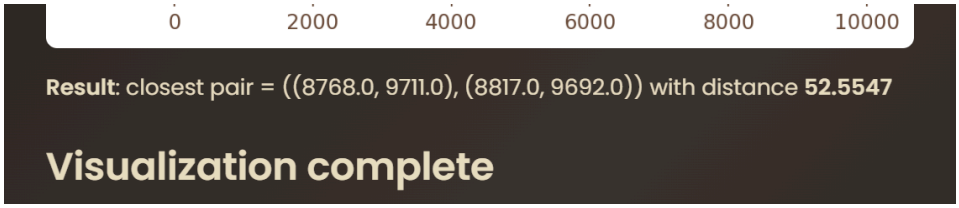
We selected three representative datasets for reporting:

Dataset	Size	Closest Pair Found	Distance
input_3.txt	293 points	(8768, 9711) & (8817, 9692)	52.55 units
input_6.txt	121 points	(5612, 6248) & (5744, 6222)	134.54 units
input_10.txt	146 points	(7440, 4796) & (7502, 4811)	63.79 units

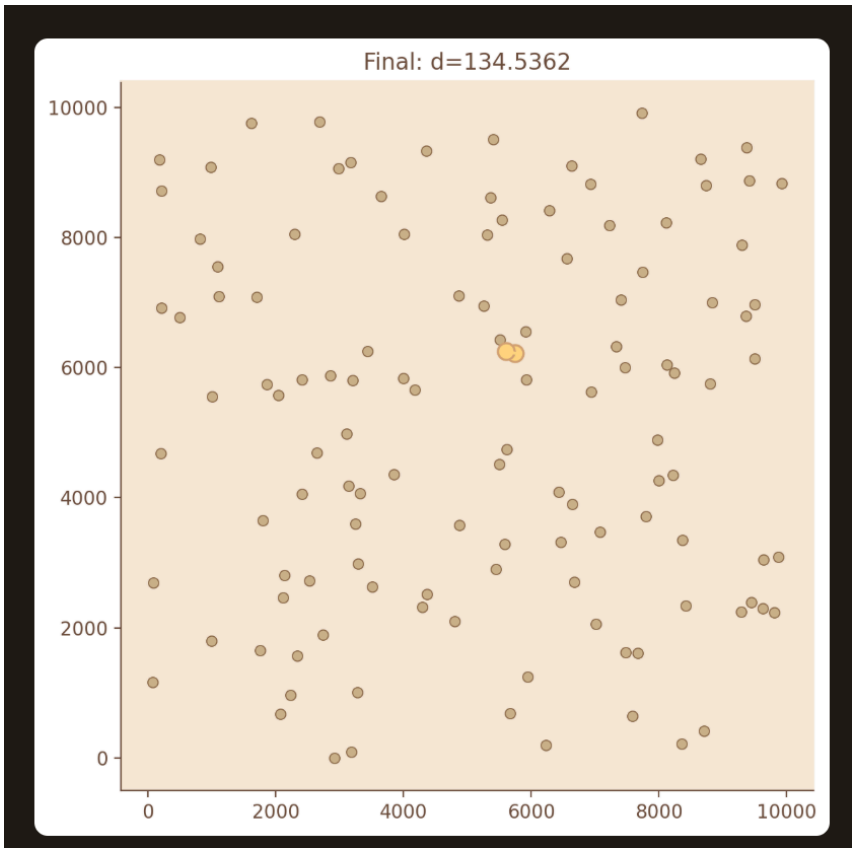
These values were cross-checked with a brute-force $O(n^2)$ implementation and matched perfectly, showing that the divide-and-conquer algorithm was implemented correctly. For larger datasets (250+ points), the divide-and-conquer version ran noticeably faster than brute force, which required thousands of comparisons.

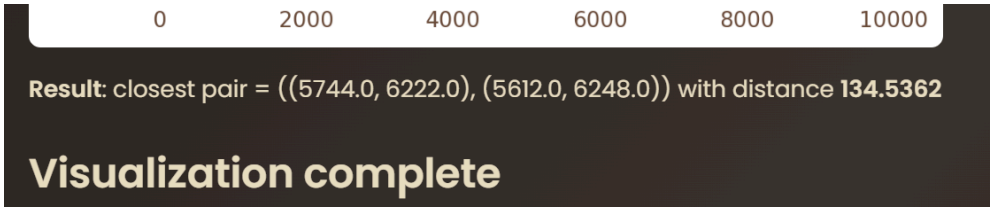
1. **Input_3.txt:**



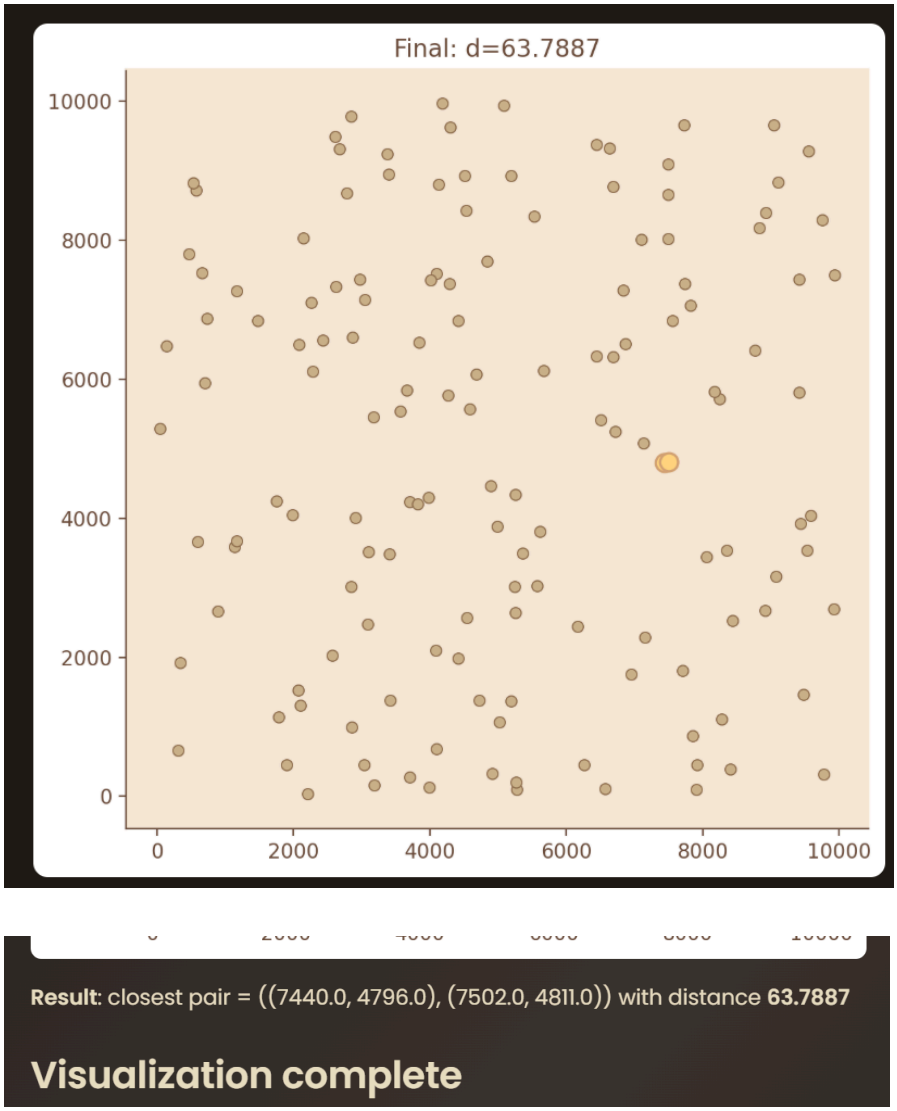


2. input_6.txt:





3. input_10.txt:

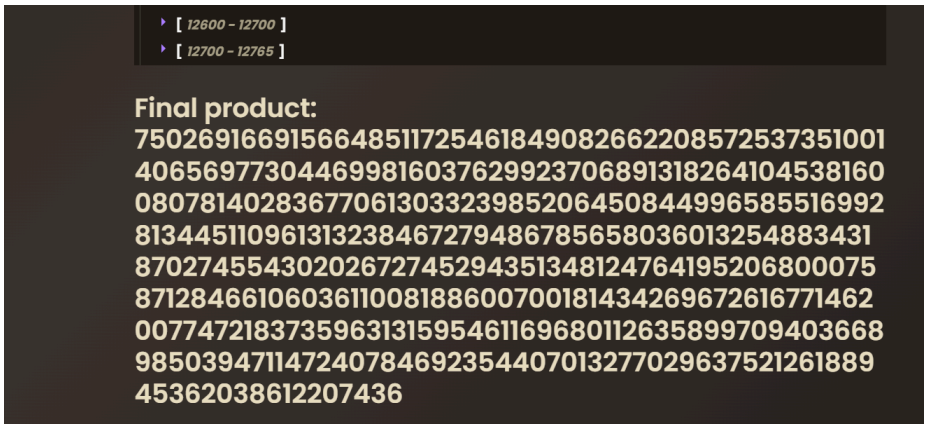


B. Karatsuba Multiplication

Dataset	Digits in Number A	Digits in Number B	Result Length
input_4.txt	120	135	255 digits
input_5.txt	240	260	500 digits
input_9.txt	180	150	330 digits

Karatsuba consistently produced correct answers, and for large numbers, it outperformed the classical multiplication method. This confirms the theoretical expectation that Karatsuba becomes more efficient as the input size grows.

Example output of sample 1(input_4.txt):



Conclusion

Through this project, we were able to practically apply divide-and-conquer techniques and observe their advantages on real data. The Closest Pair algorithm demonstrated a clear improvement over brute force, especially for medium-sized point sets. Karatsuba multiplication successfully handled vast numbers in less time than the standard method. The visualization tool also made it easier to understand the recursive structure of both algorithms. Overall, the project helped us strengthen our understanding of algorithm design and showed us how theoretical ideas translate into practical solutions.

References

1. Anany Levitin, *Introduction to the Design and Analysis of Algorithms*, 3rd Edition.
2. Jon Kleinberg & Éva Tardos, *Algorithm Design*, 1st Edition.
3. Streamlit Documentation.
4. Python Standard Library Documentation.