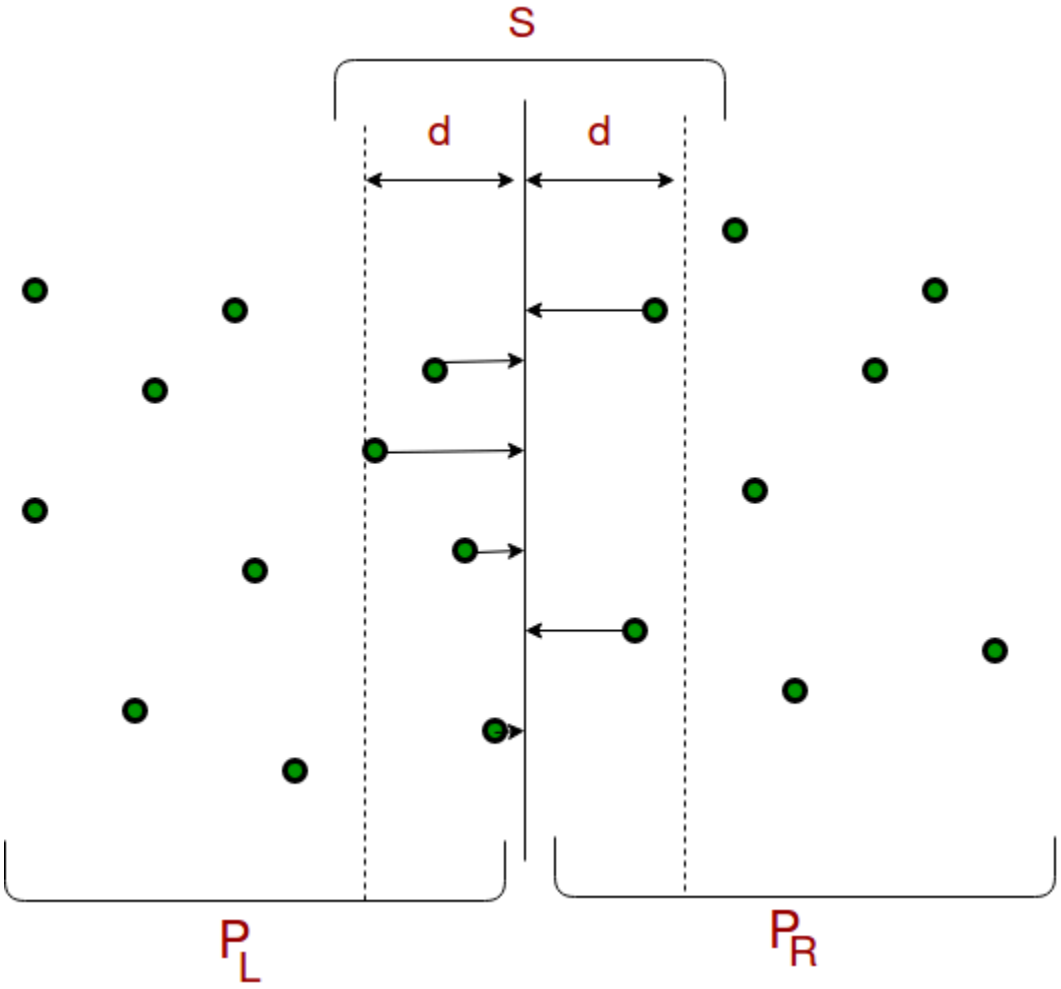
Closest Pair Problem

Brute Force VS. Divide-and-Conquer



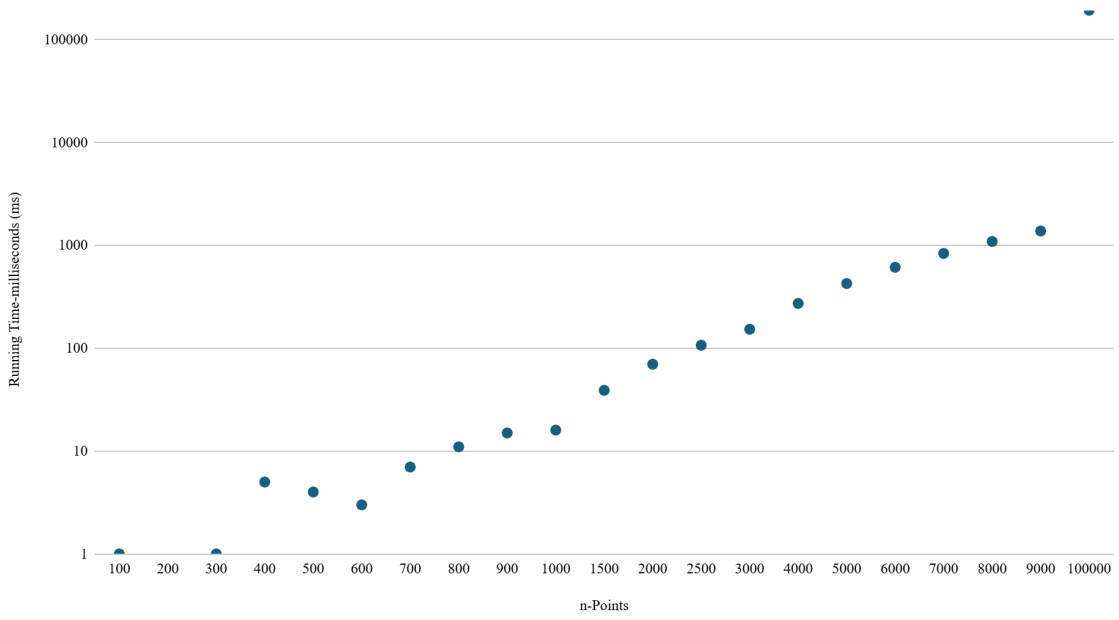
Adrian Lachowski-Patrick Pajda-Hashem Alkadri

04.12.2024

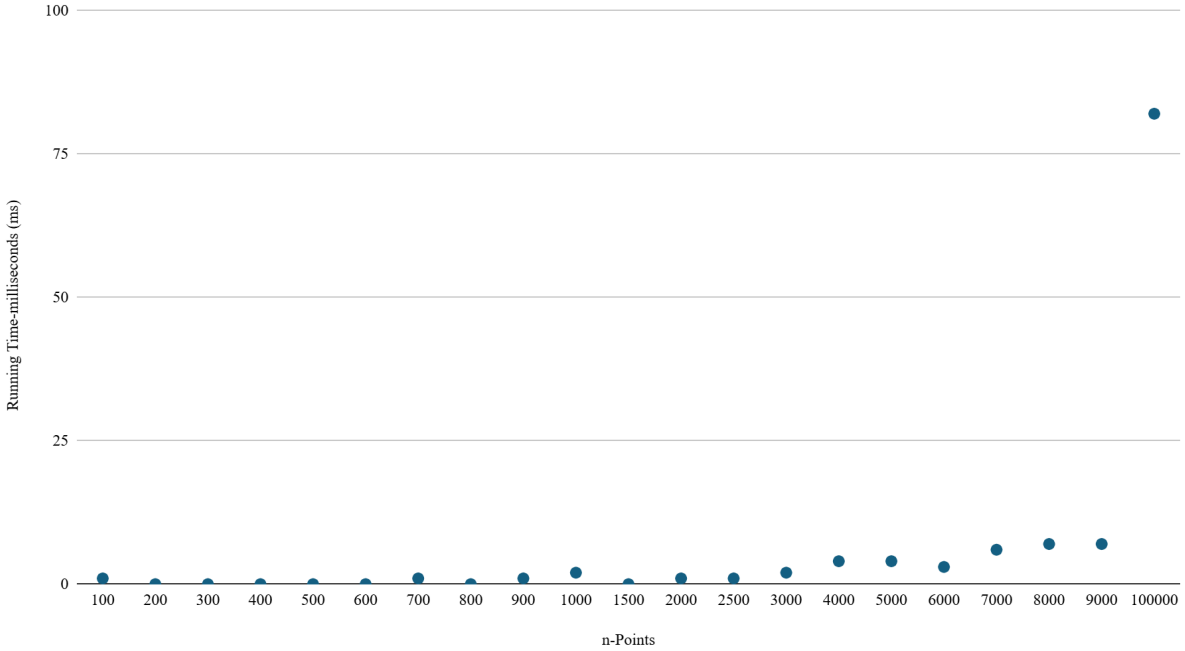
Data Structures and Algorithms

(2) Charts

Time vs. n-Points (Brute Force)



Time vs. n-Points (Divide and Conquer)



(3)Comparison of Implementations and Theoretical Time Complexity

Brute Force

Observed Performance:

The runtime chart for the brute force implementation shows a clear quadratic growth pattern. As the number of points n increases, the runtime increases steeply. For instance:

- At n=1,000, the runtime is manageable.
- At n=100,000, the runtime spikes significantly.

Comparison to Theory:

The brute force method theoretically operates at O(n^2), where the number of distance calculations(basic operation) grows quadratically with n.

The observed results align with this theoretical complexity. This makes the brute force algorithm inefficient for large datasets.

Divide-and-Conquer Approach

Observed Performance:

The runtime chart for divide-and-conquer shows a much slower growth rate compared to brute force. The execution time remains low for most values of n, increasing noticeably only for very large datasets. For example:

- At n=1,000, the runtime is minimal.
- Even at n=100,000, the runtime is significantly lower than brute force.

Comparison to Theory:

The divide-and-conquer approach has a theoretical time complexity of O(nlogn), combining a sorting step and recursive merging.

The observed results validate this complexity. The algorithm efficiently handles larger datasets, demonstrating the expected logarithmic growth in runtime as n increases.

Overall Comparison

For Small Datasets: The performance difference between the two algorithms is minimal. This is

because the overhead of divide-and-conquer (ex: sorting and recursion) is not significant at small scales.

For Large Datasets: As n grows, the divide-and-conquer approach demonstrates its efficiency, while the brute force method becomes computationally expensive. This scalability difference reflects the transition from quadratic O(n^2) growth in brute force to the more efficient O(nlogn) growth of divide-and-conquer.

(4)Implementation and Discussion

Brute-Force

1. Main Variables:
    - points: A vector containing all the input points.
    - minDistance: Tracks the minimum distance found so far.
    - closestPoint1 and closestPoint2: Store the closest pair of points.
2. Data Structures:
    - Vector of Points (std::vector<Point>): Used to store all the points.
    - Each Point structure holds two integers: x and y coordinates.
3. Algorithm:
    - The brute-force approach computes the distance between all pairs of points:
        a) Nested loops iterate through all combinations of two points in the input vector.
        b) The Euclidean distance is calculated using the calculateDistance function.
        c) If a smaller distance is found, the minDistance and the corresponding points are updated.
    - Time Complexity: O(n^2), where n is the number of points. This is due to the double loop comparing all pairs of points.

- Strengths: Easy to implement and works well for small datasets.
- Weaknesses: Becomes prohibitively slow for large datasets due to its quadratic complexity.

Divide-and-Conquer

1. Main Variables:
    - points: A vector containing all the input points, initially sorted by the x-coordinate.
    - closest1 and closest2: Track the closest pair of points found.
    - strip: A vector to store points near the dividing line for the "strip" computation.
2. Data Structures:
    - Vector of Points (std::vector<Point>): Used for both the input and intermediate results (e.g., strip).
    - Sorting by x-coordinate and y-coordinate:
        a) Sorting by x is required to enable divide-and-conquer.
        b) Sorting by y in the strip helps efficiently find the closest points within the strip.

Algorithm:

- The divide-and-conquer algorithm divides the points into two halves:
    a) Left and Right Halves: Recursively compute the closest pair for each half.
    b) Merge Step: Consider points near the dividing line (within distance dd, the smaller distance from the two halves).
- The closest pair in the strip is found using a nested loop, but it’s optimized

to consider only points within a certain vertical distance.

- Time Complexity: O(nlogn):
    a) Sorting by x-coordinate takes O(nlogn).
    b) The recursive step and the strip computation each take O(n).
- Strengths: Significantly faster for large datasets due to its logarithmic scaling.
- Weaknesses: More complex to implement and involves additional steps like sorting and merging.

(5)What We Learned

1) Algorithmic Complexity:
    - The brute force method reinforced the importance of understanding time complexity and why quadratic growth becomes infeasible for large datasets.
    - Implementing the divide-and-conquer approach highlighted the value of optimizing problems using recursive techniques and sorting as a preprocessing step.
2) Practical Insights:
    - Working with large datasets revealed real-world challenges like memory management and the importance of efficient data structures.
    - Using timing functions demonstrated how theoretical complexity translates into real-world performance, providing a practical sense of scalability.
3) How Can We Get More Efficient?:
    - Visualization: Adding a graphical representation of points and their closest pair could make the results more intuitive and engaging.

(6)Specific Issues

1. Edge Cases and Error Handling:
    - Ensuring robustness required addressing scenarios like files with less than two points or corrupted data. We implemented error messages to guide users effectively.
    - Handling duplicate points and ensuring accurate distance calculations were critical to maintaining the correctness of results.

2. Balancing Efficiency and Complexity:
    - The divide-and-conquer algorithm, while efficient, involved significant overhead in sorting and recursive calls. Debugging these steps required careful attention to maintain correctness while improving performance.

3. Dataset Challenges:
    - Generating realistic random datasets was essential to simulate real-world scenarios. We also ensured that our generator avoided overly clustered points, which could bias performance results.

4. Effort in Optimization:
    - In the divide-and-conquer approach, we spent extra time optimizing the "strip" logic to minimize unnecessary distance calculations, which significantly improved runtime.

5. Learning Curve:
    - While the brute force algorithm was straightforward to implement, the divide-and-conquer approach posed a steep learning curve. Understanding how to split the problem space, merge results, and manage intermediate data structures required considerable effort.