Adam Hayen
CS-203
Giuseppe Turini

Theoretical Analysis

Exhaustive Search

The strategy of this algorithm is to compare all possible combination pairs of the coordinates on the plane. This is a simple and intuitive solution, but it is not very efficient.

The coordinates plane is an n*15 x n*15 -sized board represented as an array of coordinate points. The algorithm makes use of this global array as its input. The input size is the number of points on the plane, n. The algorithm returns the indices of the two closest points in the coordinate array.

Basic operation

The basic operation in exhaustive search is comparing the distances between two points. The theoretical operation count, C(n), can be setup as shown below:

$$C(n) = [\sum_{j=0}^{n-1} [\sum_{i=0}^{n-1} (1)]] - \sum_{i=0}^{n-1} (1)$$

This can be simplified to:

$$C(n) = n^2 - n$$

The efficiency class of this algorithm is $\Theta(n^2)$.

Exhaustive search is an exponential solution, so it slows down quickly as the input size increases. There is no best or worst case because the basic operation count will be exactly the same for the same size n.

Divide-and-Conquer

The strategy of this algorithm is to use divide-and-conquer to find the closest pair on the coordinate plane. This solution is far more efficient than exhaustive search, because it uses smarter techniques that greatly reduces basic operation count. The coordinates plane is an n*15 x n*15 -sized board represented as an array of coordinate points. The algorithm accepts two coordinate arrays, one sorted by x-position and the other sorted by y-position. These two arrays

need to be sorted before passing them into the algorithm. The given implementation uses quicksort for both arrays. The input size is the size of the arrays, n.

Basic operation

The basic operation in divide-and-conquer is the partitioning of the x and y sorted arrays into PL, QL, PR, QR.

The recursive algorithm's basic operation count can be represented as:

$$\mathrm{DivC(n)} = \begin{cases} 0, & \mathrm{n} \leq 3 \\ 4\mathrm{n} + \mathrm{DivC}(\lceil \mathrm{n}/2 \rceil) + \mathrm{DivC}(\lfloor \mathrm{n}/2 \rfloor), & x > 3 \end{cases}$$

This can be simplified to

$$\mathrm{DivC(n)} = \begin{cases} 0, & \mathrm{n} \leq 3 \\ 4\mathrm{n} + \mathrm{DivC}(\mathrm{n}/2), & x > 3 \end{cases}$$

Using the master theorem, d = 1, a = 2, b = 2, so a = b^d. This makes the overall overall efficiency class of the algorithm:

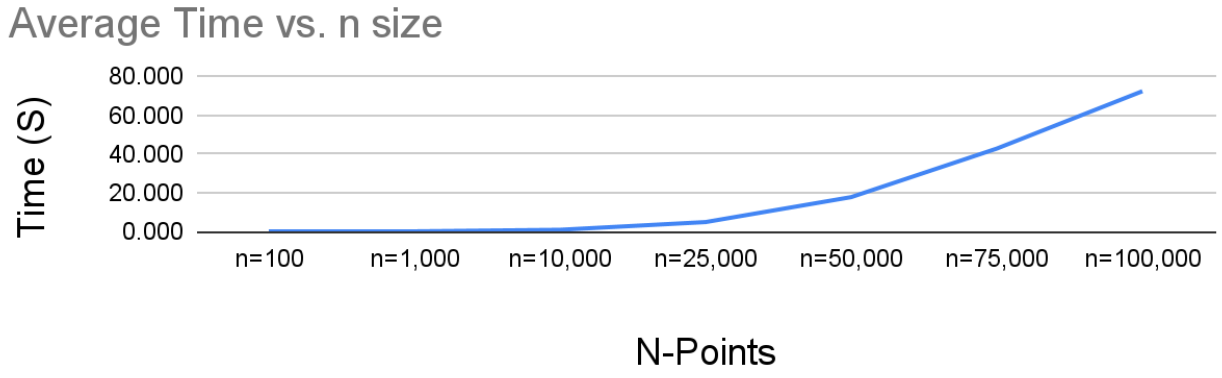
$\Theta(\, n\, log(n)\,)$

The basic operation count depends only on the input size, so there in no need for best or worst case analysis.

Empirical Analysis

Exhaustive Search

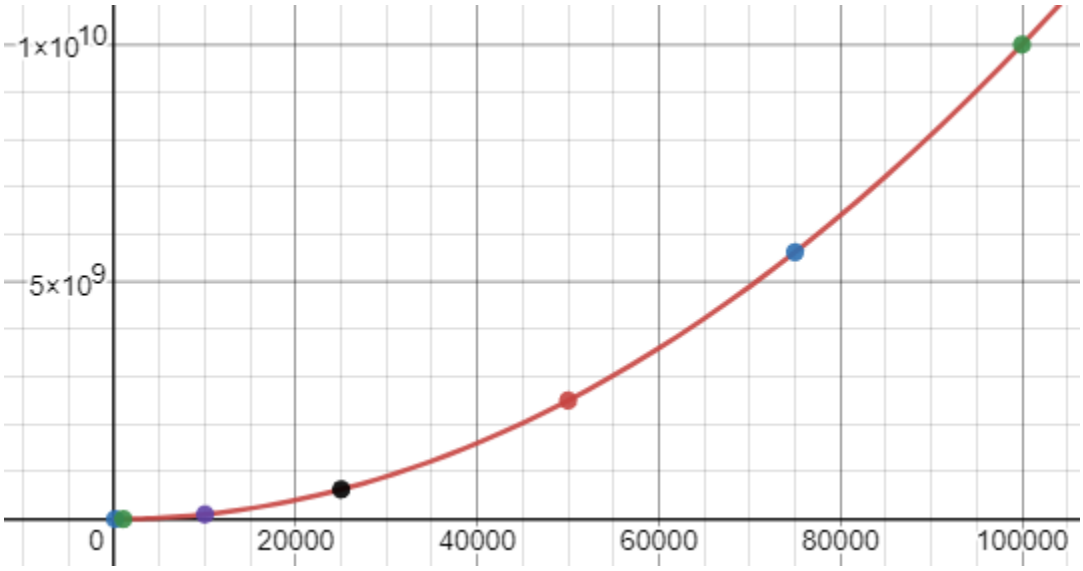
The average running time has been graphed for several significant n values. The results have been graphed as seen below. As expected, it grows exponentially.



Next, I recorded the basic operations for each size of n.

Trial	n=100	n=1,000	n=10,000	n=25,000	n=50,000	n=75,000	n=100,000
Basic Operations	9,900	999,000	99,990,000	624,975,000	2,499,950,000	5,624,925,000	9,999,900,000

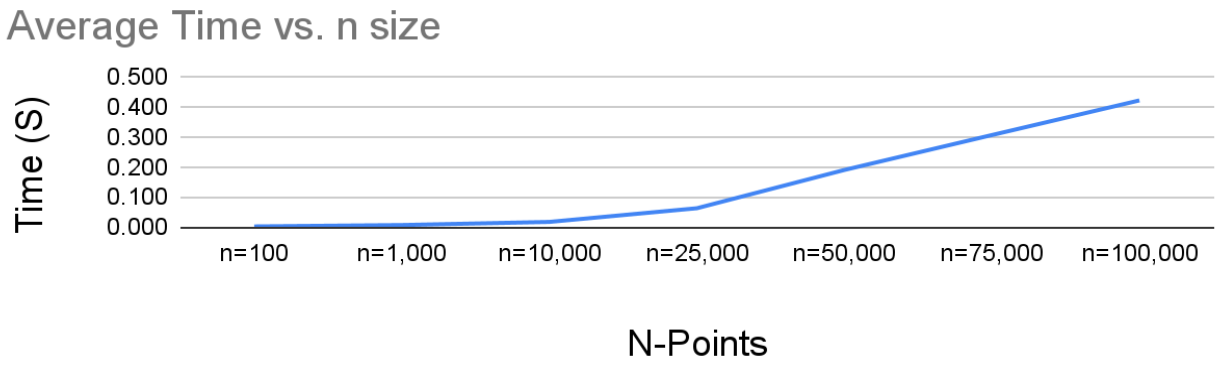
Basic operation count (y-axis) vs. n-size (x-axis) - The points represent the basic operations, the red line is the efficiency class line.



In short, the operation count almost exactly matches the expected result from the theoretical analysis. The efficiency class of this algorithm shows it will start to increase operations quickly as the input grows in magnitude.

Divide-and-Conquer

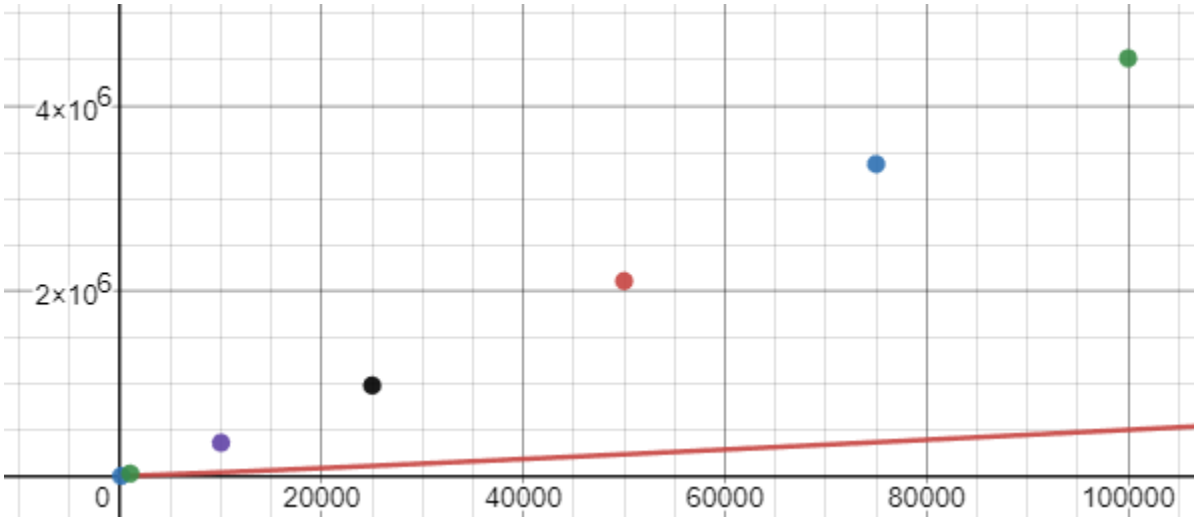
The average running time was recorded for a few different significant n values. The results are close to what is expected from theoretical analysis.



Next, I recorded the basic operations for each size of n.

Trial	n=100	n=1,000	n=10,000	n=25,000	n=50,000	n=75,000	n=100,000
Basic Operations	1548	26,784	360,000	980,088	2,110,176	3,375,000	4,520,352

Basic operation count (y-axis) vs. n-size (x-axis) - The points represent the basic operations, the red line is the function for the efficiency class.



The empirical results show that basic operation count grows logarithmically with n and is much more efficient than exhaustive search.

Conclusion

To summarize, the theoretical analysis and empirical results show that divide-and-conquer is significantly more efficient than exhaustive search. The empirical data shows that divide-and-conquer can solve for 100,000 points faster than exhaustive search can solve for 10,000. The empirical data for both solutions also closely matches the results from their respective theoretical analysis. This assignment teaches the usefulness of divide-and-conquer algorithms for solving these types of problems.