CS 310
Assignment 321

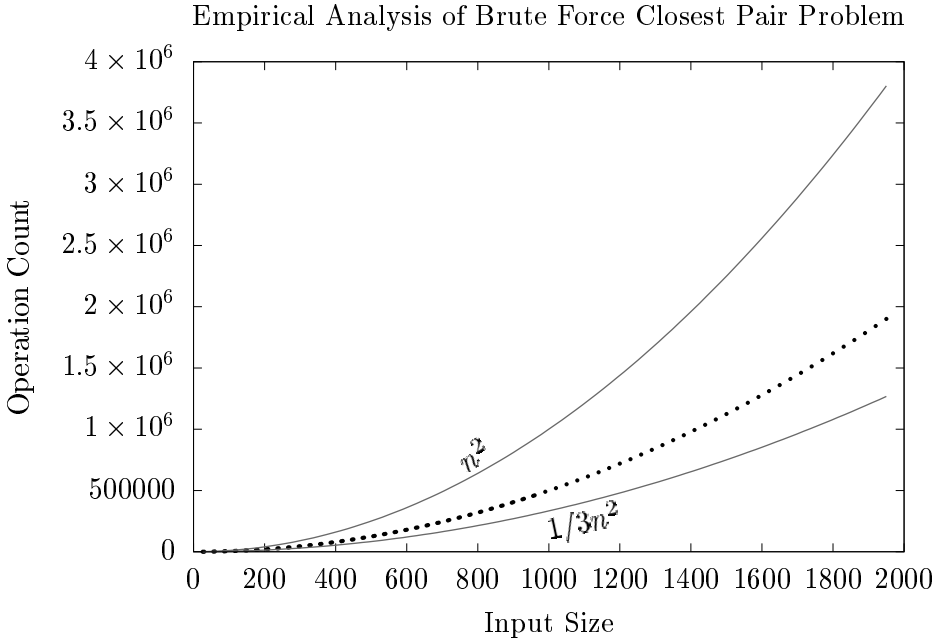
Cameron Moberg

There are several ways to solve the closest-pair problem, the two done in this analysis are the brute-force method, where you compare every single pair and find the smallest, and the divide-and-conquer method. The purpose of this study is to perform an empirical analysis of the two methods.

For both methods, the value for n is taken to be the number of points in the plane.

To analyze the brute-force algorithm, we chose calculating the distance on line 67 as the basic operation, because this happens most frequently, is the most expensive, and is the philosophical heart of this algorithm.

An empirical analysis of running the algorithm for multiple values of n produces the results shown below. Standard function $f(n) = n^2$ with constant multipliers, has been added to illustrate the analysis.



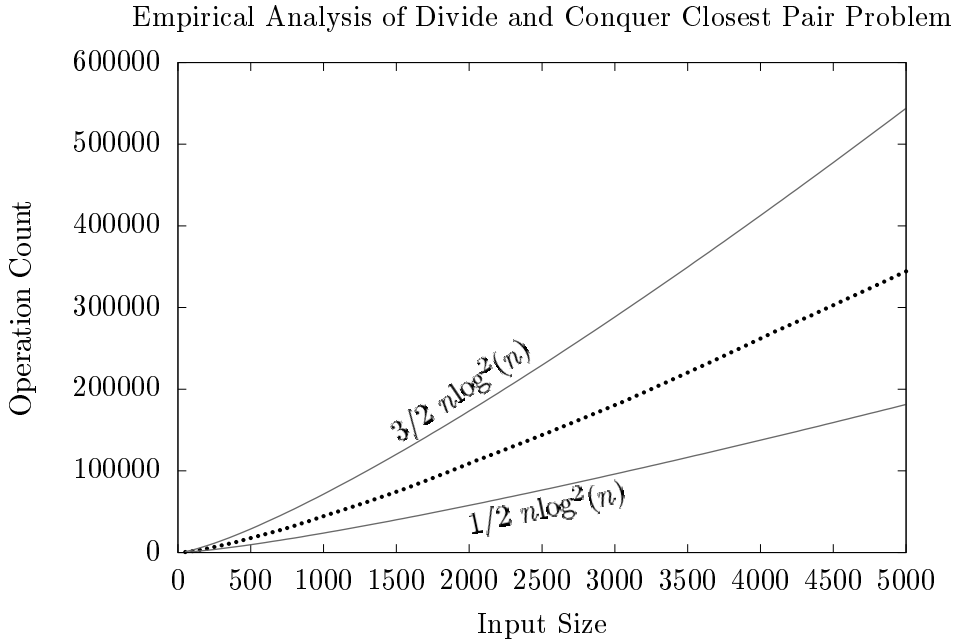
An examination of the code itself explains the empirical results when we observe that there is a doubly-nested for loop that iterates over the input size.

Therefore, since the brute-force algorithm always runs to completion we conclude that the algorithm is described by

$$T(n) \in \Theta(n^2)$$

To analyze the divide-and-conquer algorithm, we chose the sort method on lines 126–128 as the basic operation since it is by far the most expensive operation compared to all other possible choices. After research into the std::sort function that C++11 uses, it has been determined that the sorting algorithm used is of runtime complexity $\Theta(n \log(n))$[1].

An empirical analysis of running the algorithm for multiple values of n produces the results shown below. The standard function $f(n) = n \log^2(n)$ with constant multipliers, has been added to illustrate the analysis.



An examination of the code itself explains the empirical results when we observe that this algorithm is recursive, lending us the ability to use the Master Theorem, which is of the form

$$T(n) = aT(\tfrac{n}{b}) + f(n) \text{ where } a \geq 1, b > 1, \text{ and } d \geq 0$$

Now, this algorithm has two recursive calls, each recursive call operates on half of the input, and $f(n) \in \Theta(n \log(n))$ where $f(n)$ is the local work, in this case, std::sort. However, with $f(n)$ of the form $n \log n$, we cannot use the Master Theorem traditionally, and must use a special case where[2]

If it is true for some constant, $k \geq 0$, that:

$$f(n) = \Theta(n^c \log^k n) \text{ where } c = \log_b a$$

then,

$$T(n) = \Theta(n^c \log^{k+1} n)$$

Thus, after applying the Master Theorem with $c = \log_2 2 = 1$ and $k = 1$ (since $f(n) = n \log n$) we get $T(n) = \Theta(n \log^2 n)$. Therefore, since the divide-and-conquer algorithm always runs to completion we conclude, from using the Master Theorem, that the algorithm is described by

$$T(n) \in \Theta(n \log(n))$$

References