Jaydin Freeman

CS – 203

Giuseppe Turini

11/30/2022

Assignment Two

**Description**

The method calculates the two closest points when given two arrays of points, one sorted by x values and one sorted by y values. This algorithm splits both lists of points into halves so it ends up with four different arrays of points. It then recursively calls upon itself twice supplying itself those halves as arguments. It then finds the middle point of each half of the original points list and makes that center point. It then creates a strip and adds points if their X value is less than the minimum distance. It then squares that minimum distance stores it as dminsq and loop through the points in the strip and check to see if the distance between the points is less than the smallest distance and if it is it updates dminsq.

**Theoretical Analysis**

**Input size:** The input size of the divide-and-conquer is the points array. This is dependent on the size and contents of the array.

This algorithm can be broken down into five stages. The first stage is the if statement in the beginning of the method. For this stage the basic operation is the exhaustive search method call. it checks to see if the points array is larger than 4 if it’s less than four it runs exhaustive search which has the basic operation count of n squared (Lines 97 – 99). . However, the exhaustive search is only called upon when n < less than 4 times and only called upon at most n times. So, we can say at most this stage of the program would run n times and each iteration the size input would be 3 so 3 square is 9 times the number of times it’s called so n. times so then we can change . The second stage where it creates arrays for the recursive calls (Lines 101-104). . The third stage where it recursively calls itself to find the closest pair in the previously mentioned arrays. This can be . The fourth stage where it finds the minimum distance of the two arrays, gets the middle point then creates and populates said strip this section has linear run time. Finally, the fifth step is the for while loop that finds the closest pair in the strip. For this we will use the addition as the basic operator. The outer most loop runs from 0 to i – nums -2 and the while loop runs at most for 8 iterations. The best case for this will be if all points lie outside of the strip. Thus, never satisfying conditions for the while loop to run making the basic operation count for this loop . The worst case for this loop is if all the points lie within d so it would run for 8 iterations. The amount of work this while loop is using is constant so we will use one. for the best case

Smoothness Rule a = 2, b = 2, d = 1

The worst-case equation would be

Smoothness Rule a = 2, b = 2, d = 1

Final Efficiency Class

**Empirical Analysis**

For the empirical analysis of this algorithm, I compared it’s run time in nano seconds to the brute force approach. I did 1000 trials with 0 to 5000 points. With the points ranging from 0 – 1000 I created a unique points array for each trial. I then used Nanotime to log the start time and subtract it from end time then logged that time as a float in an array then I divided that by the number of trials which is 1000 to get the average below are my results. This test was performed on an Intel Core i7-8850U @ 1.8 GHz with 8 GB of Ram.

|  |  |  |  |
| --- | --- | --- | --- |
| Divide by 2 | | Brute Force | |
| **Points** | **Seconds (ns)** | **Points** | **Seconds (ns)** |
| 0 | 555 | 0 | 365 |
| 10 | 3,461 | 10 | 1,685 |
| 20 | 6,201 | 20 | 5,526 |
| 30 | 10,919 | 30 | 11,253 |
| 40 | 13,671 | 40 | 19,307 |
| 50 | 14,083 | 50 | 20,233 |
| 60 | 16,888 | 60 | 23,064 |
| 70 | 19,382 | 70 | 27,971 |
| 80 | 19,491 | 80 | 36,280 |
| 90 | 21,548 | 90 | 44,893 |
| 100 | 24,803 | 100 | 51,761 |
| 200 | 36,291 | 200 | 198,853 |
| 300 | 51,426 | 300 | 447,269 |
| 400 | 62,769 | 400 | 752,602 |
| 500 | 80,270 | 500 | 1,144,879 |
| 1000 | 128321 | 1000 | 4,575,176 |
| 1500 | 157289 | 1500 | 10,141,913 |
| 2000 | 213896 | 2000 | 17,984,385 |
| 2,500 | 263,727 | 2,500 | 27,822,034 |
| 5,000 | 5,054,300 | 5,000 | 141,891,900 |

These findings match my theoretical analysis of the program with the brute force algorithm having an n^2 runtime which is evident in the brute force algorithm graph. When looking at the Divide-by-Two and conquer algorithm it begins to look logarithmic then towards the end it starts to slope upwards. Therefore, this function falls somewhere between n and n squared like n log n supporting the findings of the theoretical analysis.