Brian Huang, Bryce Egley
CS325
2/2/2017

Assignment 1 Write Up

Brute Force

Initialize shortest distance variable
For i to number of points
    For i+1 to number of points
        Check distance between (x_i , y_i) and (x_{i+1} , y_{i+1})
        if(checked distance is shorter than shortest distance)
            Replace shortest distance with the checked distance

Return the shortest distance

Naïve Divide and Conquer

Sort the array by x value
Find the length of the array
if(array size less than 3)
    Find length by using brute force on the small array

Find the median of the array
Split the data by the median into left and right halves and recursively call the divide and conquer function on these two lists.

Take the minimum of the result of the recursive call

Create an array that contains all the points that are in the boundary of (median - d, median + d)
Find the smallest distance in the middle strip of data.

Compare the length of the left and right array to the length of the middle strip.

Return the smaller value

Enhanced Divide and Conquer

Sort the array by x values.
Sort the array by y values.
Find the array length
Find the median

If(array length is less than 3)
Use the brute force method to find the shortest distance.

Divide the data into left and right halves by using the median.
Find the shortest distance of these two halves by recursively calling itself with inputs of the smaller list.
Store the smaller distance of the two halves into a variable. (using d in this example)

Create an array that contains all the points that are in the range of (median - d, median + d)
Find the shortest distance this array by comparing the y values.

Compare the values found in the middle strip and the left and right strip.

Return the smallest value.

Runtime analysis

The brute force algorithm will compare all the points with all of the other points. This is a doubly nested for loop, so the runtime of this algorithm is $O(n^2)$

The regular divide and conquer method requires sorting first, and then it requires sorting of the y values multiple times. It also had recursive calls to itself so the recurrence relation is more complex. For our program we used python's built in sorted() function, which uses the time sort algorithm. This algorithm has a worst case runtime of O(n log n). The recurrence relation of our implementation of divide and conquer can be written as:

T(n) = 2T(n/2) + nlog n + c
$T(n) = 2T(\frac{n}{2}) + nlog(n) + c$

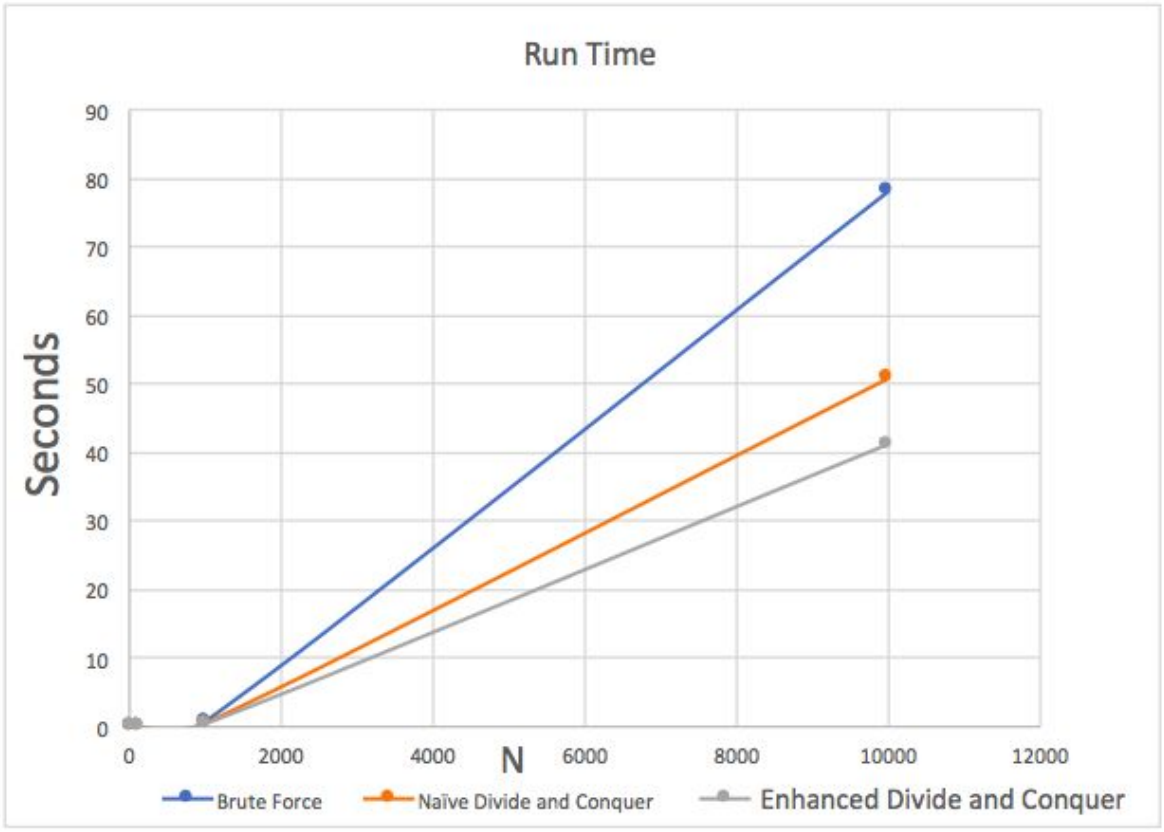
Solving the recurrence relation by using the recursion tree we get:
$T(n) = T(\frac{n}{2^i}) + \frac{n}{2^i}log(\frac{n}{2^i}) + ic$
$T(n) = O(nlog^2(n))$

The Enhanced Divide and Conquer recurrence relation is $T(n) = 2T(\frac{n}{2}) + cn$

because we eliminated sorting the array every time by presorting the points. At each level the y list is split based on the way the x list is split with the run time being O(n) and then half of it will go down to the next x list. The strip uses the y list since it is presorted. This allows it to skip sorting at each y level so each level is O(n). Solving the recurrence relation we get $T(n) = O(n \log n)$

Plotting the runtime

n	Brute Force	Naïve Divide and Conquer	Enhanced Divide and Conquer
10	0.000339031	0.000332117	0.000283957
100	0.01053381	0.006680012	0.004333973
1000	0.76055789	0.493371964	0.424195051
10000	78.09054184	50.81849694	41.23361301



Interpretation and discussion

The results of our runtime analysis were what we expected. The growth curves match what we expected. The brute force method should be the slowest, and become increasingly slow as the data set gets larger. The difference between the enhanced version and the native version should be noticeable, but not too large. The difference between the two is the constant sorting of data in

the regular version. Since the enhanced version starts with sorted lists there the time dedicated to sorting is constant.