CS 325 Analysis of Algorithms

Implementation Assignment 1: Divide and Conquer

Daniel Green, Grayland Lunn, and Alex Tovar

Pseudo Code:

The code uses an object distObj that has a list starting with smallest distance and the following pairs sharing that smallest distance.

**Brute Force:**

brute\_force(list of points)

point1 = first point in list

point2 = second point in list

min = the distance of these first pair of points

ln = length of list

if ln is equal to 2

if min is less than current objects distance

set min as the objects current smallest distance

add pair of points to list of smallest distance points

return object

else

for i in range of ln – 1

for j in range of i+1 to ln

dist = get distance between point i and point j

if dist is less than distObj

distObj = set dist as new min distance

clear list of points and add new points to list

elif dist is equal to current min distance

add points to list

return distObj

**Naïve Divide and Conquer:**

closest\_pair(list , object list of smallest distance and points)

xsorted = sort list of points by x

length = length of xlist

index = 0

if length <= 3 #base case

for point in xlist

if index + 1 is greater then length of xlist

then add the two points

increase index

return distObj

mid = middle of list

leftArrayX = split the left half of xsorted

rightArrayX = split the right half xsorted

distObj1 = recursive call closest\_pair(leftArrayX, distObj)

distObj2 = recursive call closest\_pair(rightArrayX, distObj)

merge distObj1 and distObj2

get delta by getting distance from combined lists

min\_cross = closest\_cross\_pair(xlist, delta) #find min distance on boundary

return list of merging distObj1 and min\_cross

closest\_cross\_pair( list, delta)

distObj = get current smallest distance

length = length of list

mid = length of list divided by 2

if length is less than 7

strip = list

else

strip = list from range of [mid – delta, mid +delta]

sort strip by y coordinates

index = 0

for point in strip

for i in range of 1 through 8

if index + i < length of strip

then add those points to distObj

increase index by 1

return distObj

**Enhanced Divide and Conquer:**

closest\_pair(list sorted by x, object list of smallest distance and points)

length = length of xlist

index = 0

if length <= 3 #base case

for point in xlist

if index + 1 is greater then length of xlist

then add the two points

increase index

return distObj

mid = middle of list

leftArrayX = split the left half

rightArrayX = split the right half

distObj1 = recursive call closest\_pair(leftArrayX, distObj)

distObj2 = recursive call closest\_pair(rightArrayX, distObj)

merge distObj1 and distObj2

get delta by getting distance from combined lists

min\_cross = closest\_cross\_pair(xlist, delta) #find min distance on boundary

return list of merging distObj1 and min\_cross

closest\_cross\_pair( list, delta)

distObj = get current smallest distance

length = length of list

mid = length of list divided by 2

if length is less than 7

strip = list

else

strip = list from range of [mid – delta, mid +delta]

sort strip by y coordinates

index = 0

for point in strip

for i in range of 1 through 8

if index + i < length of strip

then add those points to distObj

increase index by 1

return distObj

Asymptotic Analysis:

**Brute Force Algorithm:**

In the brute force algorithm, we have a nested for loop. Making this algorthim have a runtime of O(n2). There are constant operations as well but the nested for loop overtakes them.

**Naïve Divide and Conquer:**

In the naïve Divide and Conquer, the elements enter the function unsorted. In each call they are sorted again before either entering the base case or going through the recursive call again. The recursive call splits the array in half. This gives the function logn runtime. But when the function closest\_cross\_pair() is called, the runtime in that is O(n). That is because there is a for loop in the function that loops through each element in the list once. This combined gives a total runtime of O(nlogn). Since there is sorting done in each function call, and the sorting runtime is logn, Then the actual runtime of the whole program is O(nlogn2).

**Enhanced Divide and Conquer:**

In the enhance Divide and Conquer, the elements are pre-sorted when entering the function. In each recursive call we split the array in two halves. Then when we call the closest\_cross\_pair function, it runs a for loop adding points that have the minimal distance. The for loop has an O(n) runtime, and the recursive call has a O(logn) runtime because each call halves the problem. So the total runtime will be O(nlogn).

Plotting and Analysis:

Figure 1: Runtime Plot of Enhanced Divide and Conquer

The figure above shows the run time of the Enhanced Divide and Conquer algorithm. The graph in the first few points look to be logarithmic and then goes to linear once it gets to bigger data size. This matches our asymptotic analysis of a runtime of O(nlogn). But overall we can see the algorithm finding the closest distance in a short amount of time

Figure 2: Runtime Plot of Naïve Divide and Conquer

This figure shows the runtime for the Naïve Divide and conquer algorithm. The difference is noticeable but not that significant. It is a little slower than the enhanced version. It is because this algorithm repeatedly sorts the list of points in each recursive call.

Figure 3: Runtime Plot of Brute Force

This is the plot for the Brute Force algorithm. As expected it has an exponential runtime. It looks similar to n2, which is what we analyzed it to be. As n gets bigger, the time it takes to compute the closest pair gets exponentially bigger. For n = 105, It was cut short because it was taking a large amount of time.