Michael Gillett
Drake Seifert
Nathan Shepherd

CS 325 Assignment 1

Pseudo-code

Brute Force

```
bruteForce(points)
coords, dists = []
for i in range (0, length(p))
        for j in range (i + 1, length(p))
                distance = sqrt((x2 - x1)^2 + (y2 - y1)^2)
                coords.append(p[i], p[j])
                dists.append(distance)
pairs = []
indices = the index locations of the minimum distance in dists
for range(0, length(indices))
        append coordinate pairs to pairs list
sort pairs
```

Divide and Conquer

```
divideAndConquer(points sorted by x)
//Base Case:
If len(points) == 1
        Return error
Else if len(points) == 2
        Return distance = sqrt((x2 - x1)^2 + (y2 - y1)^2)
Else if len(points) == 3
        Calculate min distance between the three points using pythagorean theorem
        Return min distance
//Recursive Case:
        Compute x value of separation line
        Split points array into two halves
        Recursively call each half
        Check min distance between the two calls
        Find range of separation line plus and minus min distance, find all points within this
range
        Calculate distances and return overall minimum distance
```

Enhanced DandC

```
enhancedDandC(points sorted by x, points sorted by y)
```

If len(points) == 1:

Return error

Else if len(points) == 2:

distance = $\sqrt{(x2-x1)^2+(y2-y1)^2}$

Return distance and points

Else if len(points) == 3:

abDistance = $\sqrt{(a2-b1)^2+(a2-b1)^2}$

acDistance = $\sqrt{(a2-c1)^2+(a2-c1)^2}$

bcDistance = $\sqrt{(b2-c1)^2+(b2-c1)^2}$

Return min(abDistance, bcDistance, acDistance), point pairs with min

Else:

Calculate Median value and index

Split Points array based on median index

Call enhancedDandC on both left and right

Find side minimums

close Pairs = [points that are within d of the median value]

Compare each point to the points within d on the y-axis:

Calculate distance, as above

If it is less than, replace and store

If it is equal to, store

If it is greater than, continue

Return the minimum distance and points

Asymptotic Analysis of Runtime

Brute Force

(Given): $O(n^2)$

Divide and Conquer

Master Theorem:

$T(n) = aT(n/b)+cn^d$

a = 2, b = 2, c is a constant, and d = 1

Recurrence relation:

$T(n) = 2T(n/2) + cn\log(n)$

a = 2 because the recursive call happens to each side of the set

b = 2 because the size is divided in half each recursive layer

“cnlog(n)” includes both finding the minimum distance and calculating the points around the separation line

The master theorem can now be applied to solve this equation:
$a = b^d \Rightarrow (2 = 2^1)$
Because ß = 1 > -1, $T(n) = O(n^{\log_b(a)} * \log(n)^{(ß+1)})$
$\Rightarrow T(n) = O(n^{\log_2(2)} * \log(n)^{(1+1)}$
$n^{\log_2(2)} = n$
$\Rightarrow T(n) = O(n\log(n)^2)$

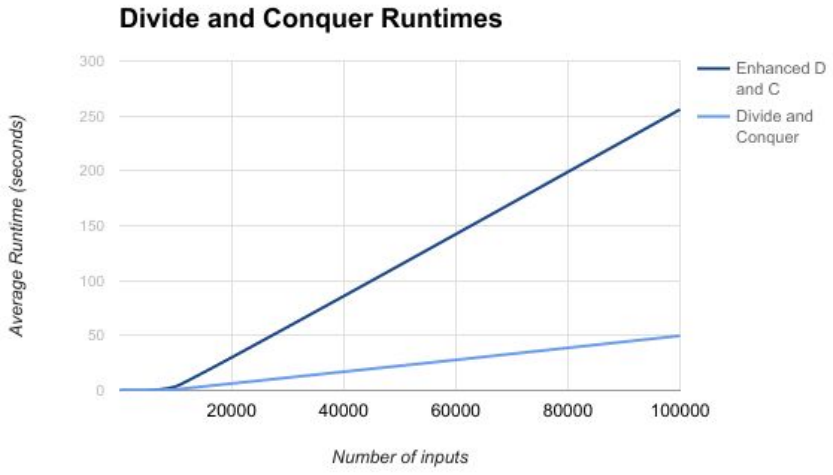
Enhanced DandC
Master Theorem:
$T(n) = aT(n/b)+cn^d$
a = 2, b = 2, c is a constant, and d = 1

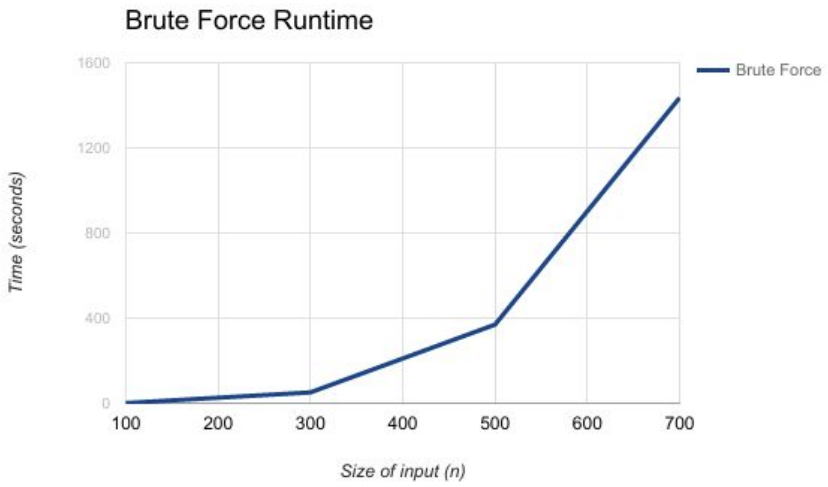
Recurrence Relation:
T(n) = 2T(n/2) + cn

a = 2 because the recursive call happens to each side of the set
b = 2 because the size is divided in half each recursive layer
cn because the presorted y list allows for constant comparison in the middle strip, comparing at most 7 other points, as no more can be within the minimum of the sides.

The master theorem can now be applied to solve this equation:
$a/b^d = 2/2^1 = 1$
$\Rightarrow T(n) = O(nlog(n))$

Plotting the Runtime





Average time in seconds for Brute Force:
100:
0.69621

300:
50.82234917

500:
369.4047839641571

700:
1435.4049019813538

Average time in seconds for DandC:
100:
0.001029205322

1,000:
0.02120804787

10,000:
0.9207650185

100,000:
49.75536542

Average time in seconds for EnhancedDandC:
100:
0.00113

1,000:
0.03829578

10,000:
3.4894064

100,000:
255.87766

Interpretation and Discussion

The experimental runtimes of each algorithm seems to follow a general trend that is to be expected, but there are some slight variations within the programs that cause the enhanced divide and conquer to be a bit slower than the naive divide and conquer. This could be due to two different programmers styles in writing; in the enhanced divide and conquer there are multiple instances where data is sorted before it is needed, which could slow down the algorithm after a large number of inputs. Overall however, the experimental runtimes follow the general trend of the asymptotic runtimes; brute force took a very large amount of time. An example of the expenditure of brute force, it was left running with 10,000 inputs, for four hours, and it didn't complete in this time. Other factors to discrepancies seen could include the quirks of our language, Python. Python is known to be slower, while allowing greater amounts of abstraction for the programmer. We chose Python for its ease, but if we were making an algorithm with speed in mind we would most likely use C. As mentioned before one factor could be programming styles. If all of the codes were programmed using the same functions and methods perhaps we would better see the relationships that we expected. Also, given more time, we might have been able to analyze each algorithm and how much time each line took, as although each individual line takes a constant time, some lines take far longer, and there is probably room for improvement.