Hanqing Wang
CS 130B
9594458

PA 1

1a. My algorithm for brute force checks every single pair of points. Since there are *n* points, there are n^2 pairs of points, so my algorithm for brute force is O(n^2).

1b. My algorithm for divide and conquer:

1. Make two lists: one with all the points sorted based on ascending x value, the other with ascending y value. Pass both into our divide and conquer step.
2. Divide and Conquer
    a. If there are 3 or fewer points in any recursion, we use brute force to find the shortest point.
    b. Splits the list with x ascending into two smaller lists: one list contains all the points with an x value less than the median, and the other contains all the points with an x value above or equal to the median.
    c. Splits the list with y ascending into two smaller lists: one list contains all the points with an x value less than the median, and the other contains all the points with an x value above or equal to the median.
    d. We recursively find the shortest pair: for the points less than the median, and the points greater than or equal to the x median, by passing the respective lists we made in the two previous steps.
    e. We find the smaller value, which will be called smaller of the two shortest pairs from our two recursive calls. We also save the pair of points from each of the recursive calls.
    f. We find the center: defined by the average x value of the median.
    g. We add all the points from the y ascending sorted list that are no further than the smaller value (from step e) from the center to a new list.
    h. For each point A in this new list, do this: for each point B to the right of point A, starting from the point directly to the right of A, if the distance between points A and B are smaller than the smaller value from step e, set the smaller value as the distance between points A and B. Then, save the two points we are comparing.

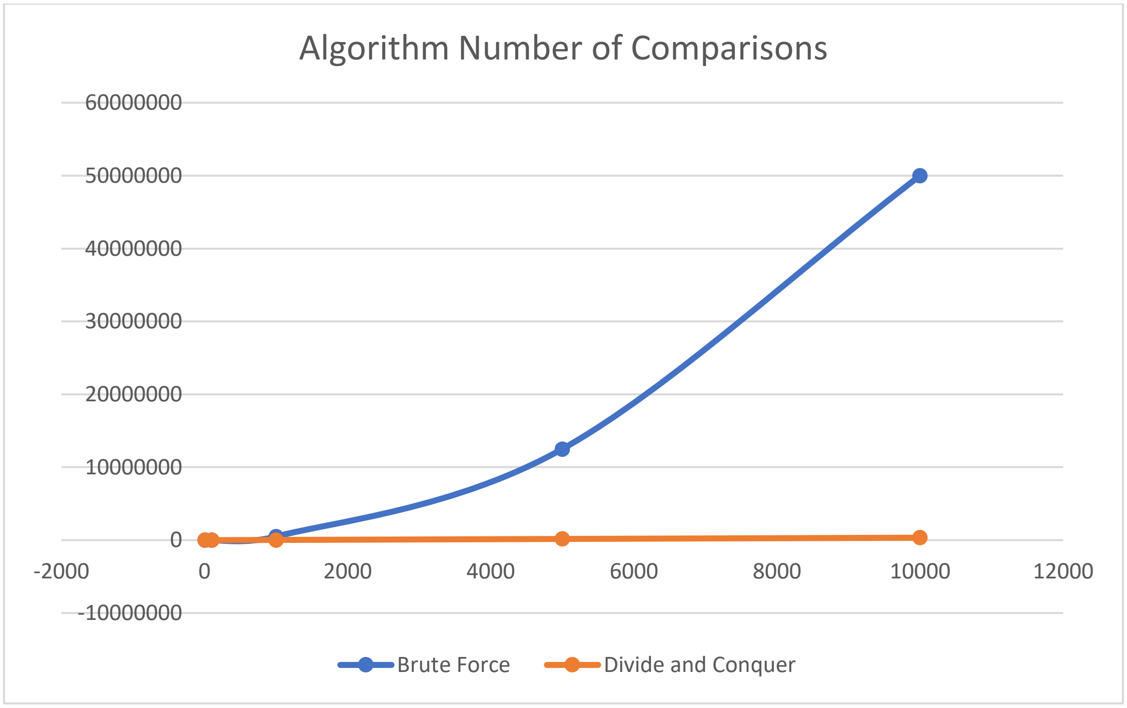
Complexity:
Step 1 is O(log(*n*)) because of the sorting.
Step 2a. makes at most 3^2=9 comparisons, and is called at most n/2 times, so is essentially O(*n*).
Step 2b. looks at each point a constant number of times, and is O(*n*).
Step 2c. looks at each point a constant number of times, and is O(*n*).
Step 2d. See *Overall Complexity*
Step 2e. is O(1).
Step 2f. is O(1).
Step 2g. looks at each point a constant number of times, and is O(*n*).
Step 2h. The inner for loop runs at most 6 times because each pair of the potential shorter points must be at least the value in step 2e. from each other, otherwise we would have found such a shorter distance already. Therefore, since there are *n* points, this process is O(*n*).

Overall Complexity:
The recursion is O(*log*(*n*)) layers deep, because each step has n/2 size of the last one. For each layer, the sum of all the *n* values is n because we run a recursion of size m/2 twice within a step of size m. Therefore, the overall runtime is O(*n*) for each layer. We also do O(nlog(*n*)) of sorting. The overall process is therefore O(log(*n*)).

Comparisons:

Number of Points	Brute Force Comparisons	Divide and Conquer Comparisons
10	47	87
100	4952	1880
1000	499502	28619
5000	12497502	164233
10000	49995002	337356

Graph:


Algorithm Number of Comparisons
Brute Force
Divide and Conquer