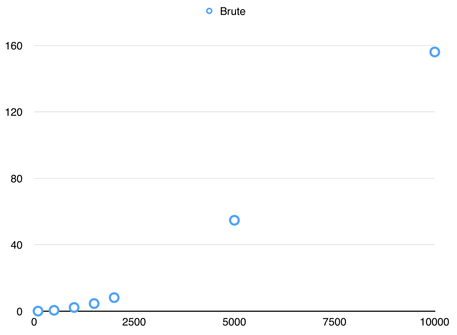
Scalability of Closest-Pair Algorithm

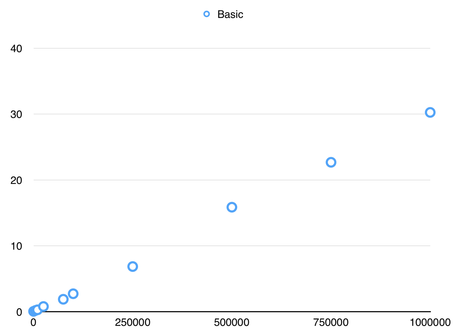
Name: Chang Rey Tang
Perm: 7781693

Items	Brute	Basic	Optimal
10^2	real 0m0.057s user 0m0.054s sys 0m0.006s	real 0m0.053s user 0m0.038s sys 0m0.008s	real 0m0.048s user 0m0.037s sys 0m0.010s
10^3	real 0m1.526s user 0m1.515s sys 0m0.010s	real 0m0.066s user 0m0.055s sys 0m0.011s	real 0m0.066s user 0m0.055s sys 0m0.011s
10^4	real 2m36.050s user 2m27.123s sys 0m0.017s	real 0m0.285s user 0m0.279s sys 0m0.005s	real 0m0.253s user 0m0.232s sys 0m0.006s
10^5	— took too long —	real 0m2.723s user 0m2.681s sys 0m0.016s	real 0m2.334s user 0m2.201s sys 0m0.019s
10^6	— took too long —	real 0m30.245s user 0m26.417s sys 0m0.118s	real 0m24.605s user 0m23.716s sys 0m0.131s



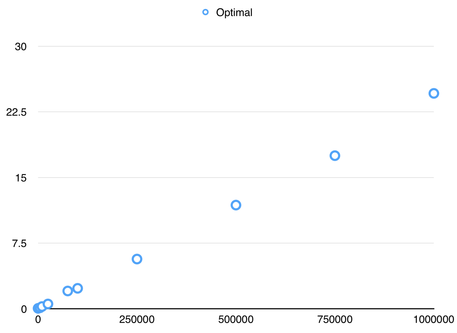
Number of Items	Time (seconds)
100	0.057
500	0.550
1000	2.236
1500	4.621
2000	8.175
5000	54.749
10000	156.05

The brute force algorithm has the time complexity O(n^2) because it checks the distances of all possible pairs of points in the dataset regardless of current minimum distances. There is no sorting done beforehand. The graph shown above definitely matches the typical shape of a O(n^2) algorithm.



Number of Items	Time (seconds)
100	0.053
1000	0.066
5000	0.188
10000	0.285
25000	0.786
75000	1.884
100000	2.723
250000	6.851
500000	15.857
750000	22.673
1000000	30.245

The basic algorithm has the time complexity O(n(logn)^2) because you sort by x-coordinates before entry into the algorithm. The reason it is O(n(logn)^2) is because we sort all “cross-median” points by y-coordinates for every recursive call that splits the data set into left and right based on the position of the median. We are able to significantly improve the run-time compared to the brute force algorithm because we no longer check every possible pair of points. By sorting the data set beforehand, we are able to ignore pairs that are clearly going to be greater distances than the current calculated minimum. The graph shows that the algorithm does indeed run quite close to O(n(logn)^2) time.



Number of Items	Time (seconds)
100	0.048
1000	0.066
5000	0.114
10000	0.253
25000	0.547
75000	2.030
100000	2.334
250000	5.678
500000	11.842
750000	17.489
1000000	24.605

The optimal algorithm has the time complexity O(nlogn) because you sort all input coordinates by x and y respectively before entry into the algorithm. This means that you no longer need to sort by y-coordinates every time you check for “cross-median” distances. My data shows that the algorithm does indeed run quite close to O(nlogn) time. It is definitely better than both the basic and brute force algorithms, though only very slightly when compared to basic. I’m sure if I took the time to run this algorithm on larger data sets we would be able to see a larger disparity between optimal and basic solutions.