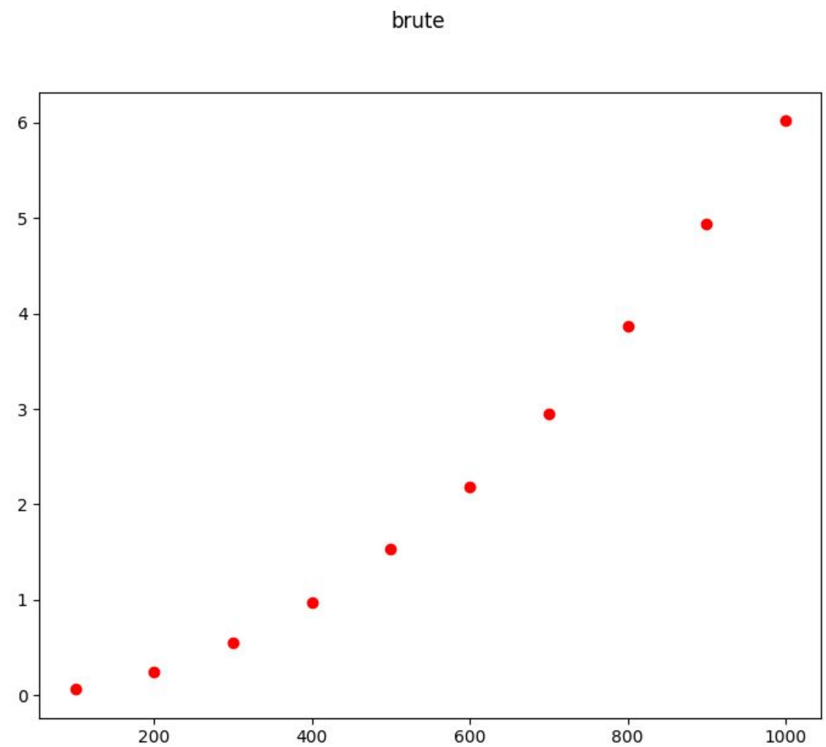
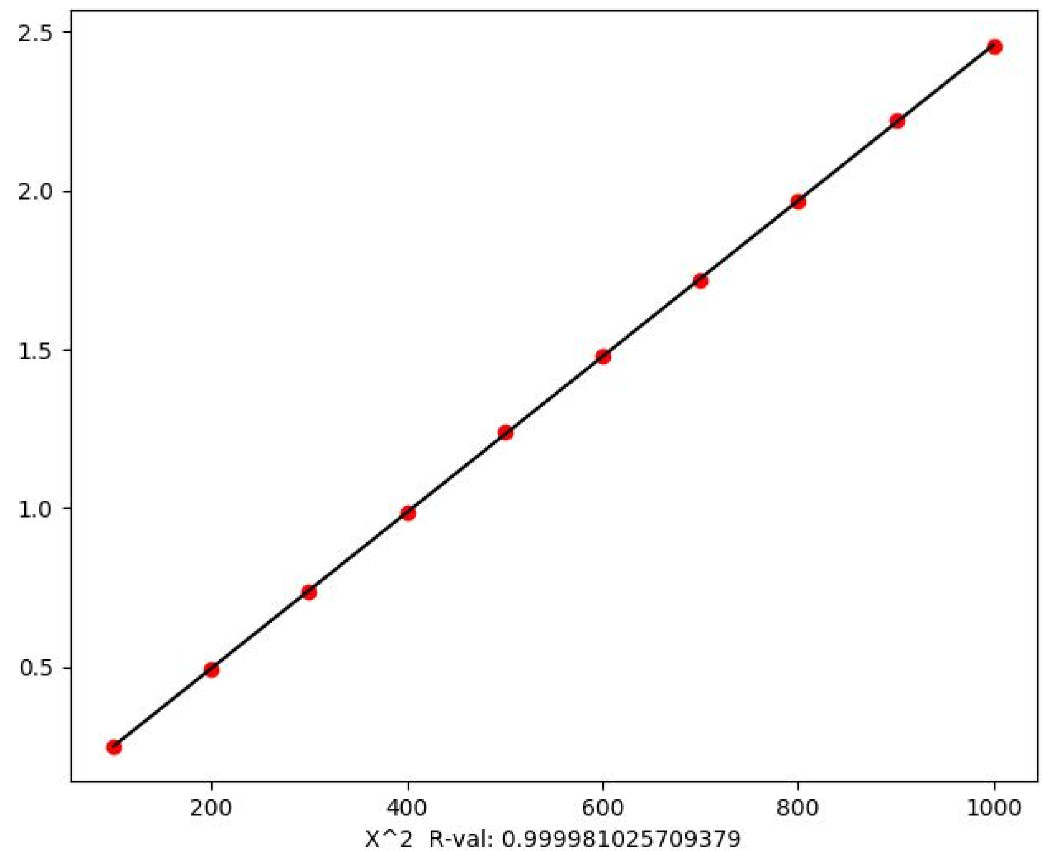
Scalability

Starting with the brute force algorithm, we can see that it looks to be polynomial in complexity:



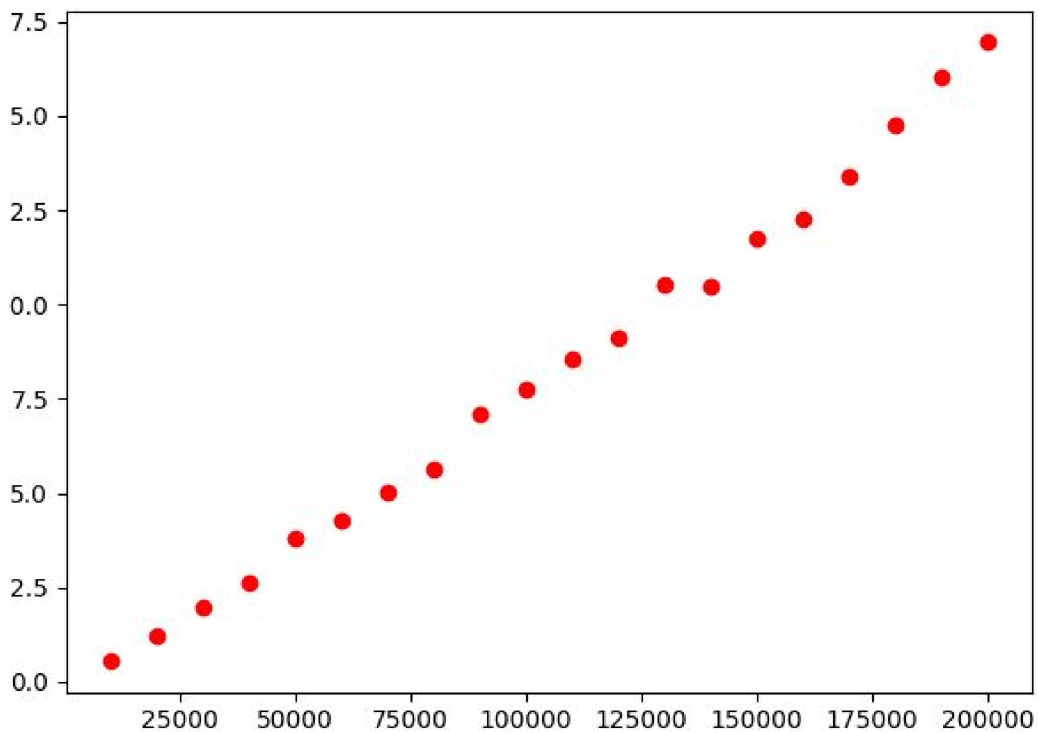
We can verify this more by smoothing the data and plotting it against a straight line (done in plot_data.py). This was done by calculating the sqrt(y) [i.e. sqrt(time)] and plotting it against number of pairs:



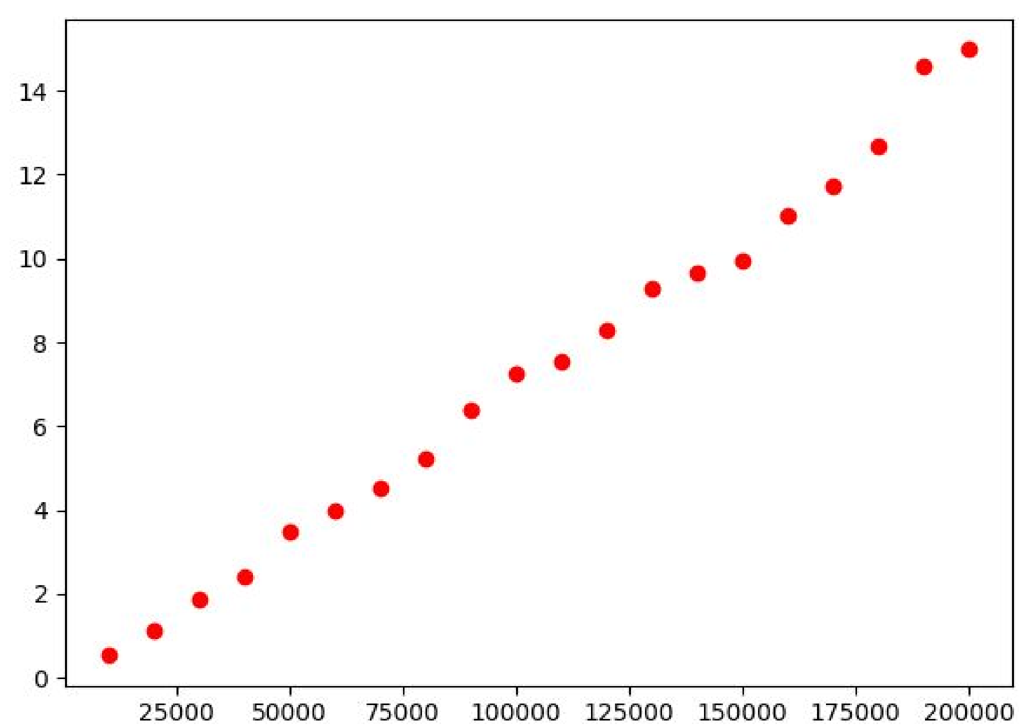
As we can see by the R-value and the exactness of the points, this is growing by n^2. I had trouble getting more than 1000 points to be calculated (it took a while to get these points -- pickle came in handy!)

Next we can look at basic and optimal algorithms, as seen here:

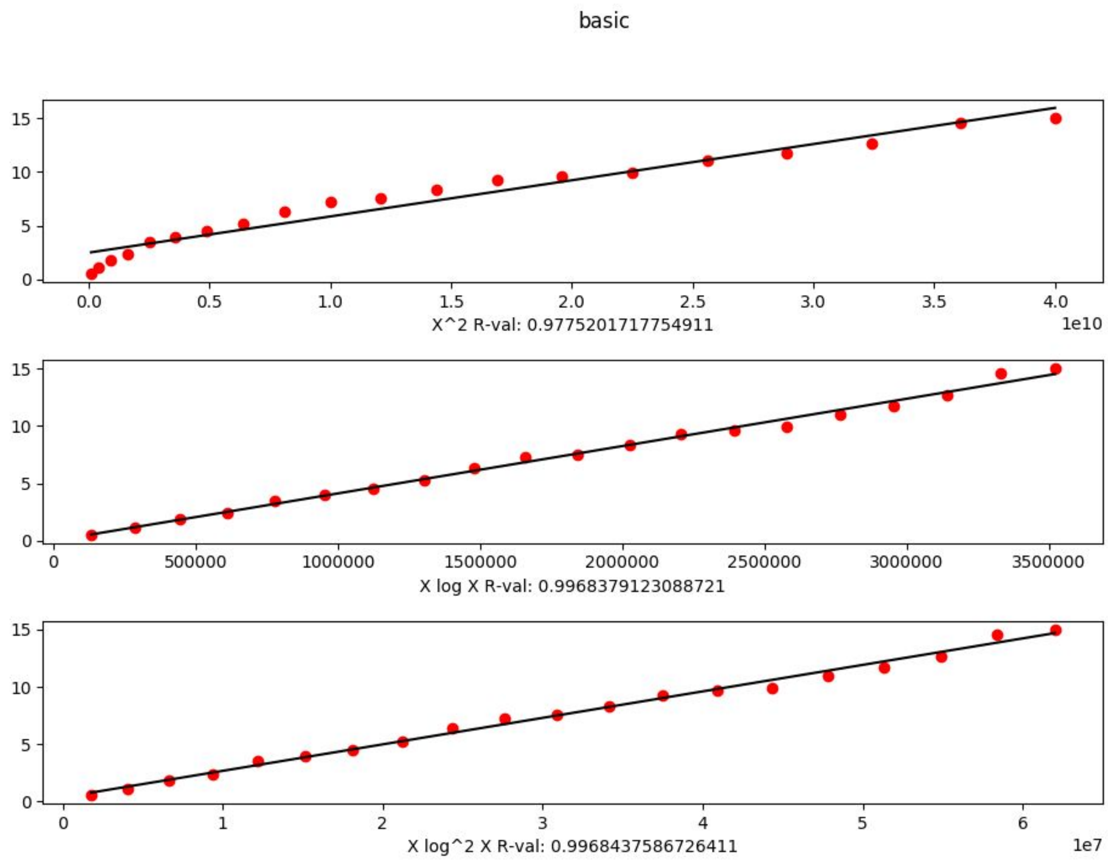
optimal



basic

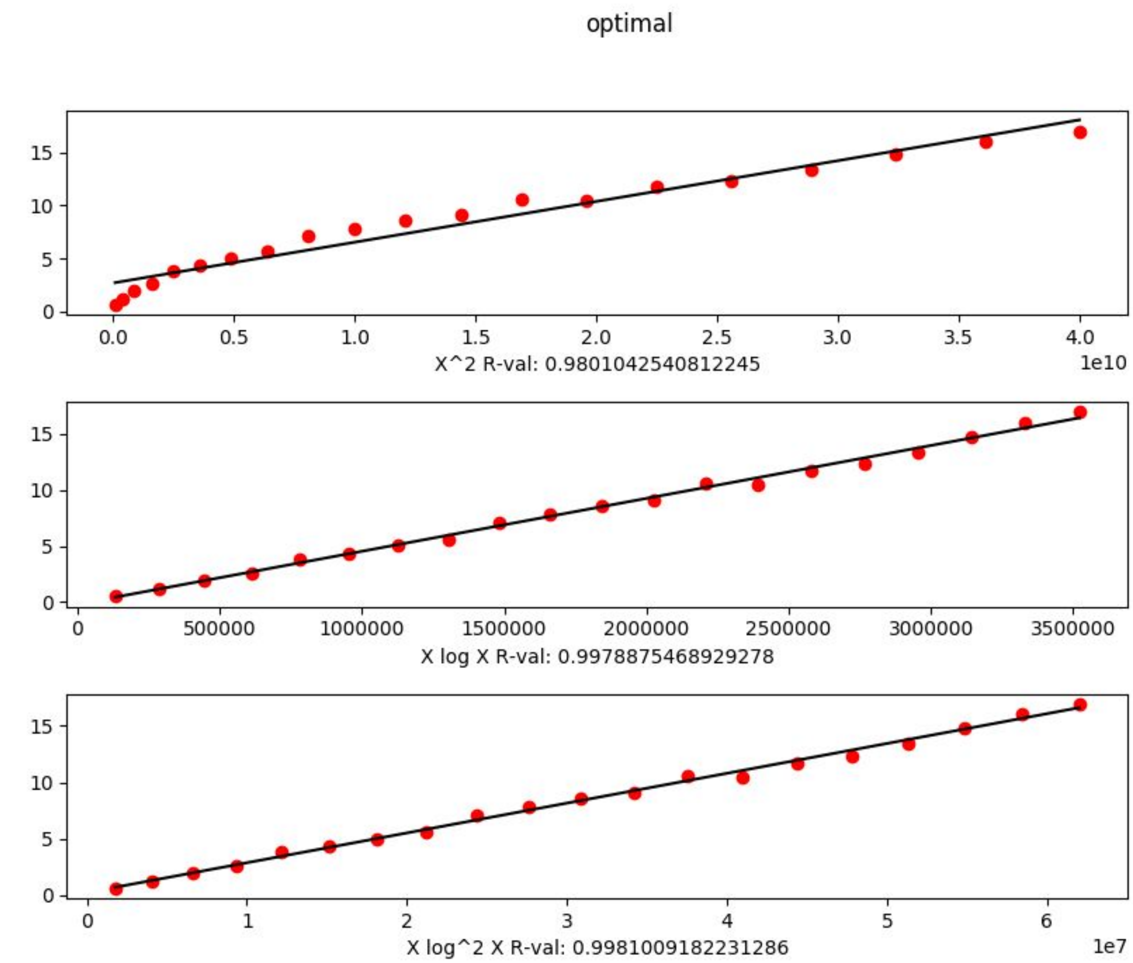


These look relatively linear (at least a lot more linear than brute force), but they both seem to have a slightly greater rate of change as x (number of points) increases. By transforming x (using the functions x^2, x log^2 x, x log x, we can see the resulting linear best-fit lines. Looking at the basic algorithm:



We can see the most correlation with y vs x log^2 x, and the least correlation with y vs x^2. This fits into our narrative that I actually programmed it right oh thank god.

Next, we look at our optimal solution:



Here we can see a very similar set of graphs. We can see it more or less fits for the bottom two graphs. But x log^2 x still has a slightly higher r-value. Why could this be? I don’t know. But I think it has something to do with the amount of discarding (and iterating) my algorithm does when pruning the sorted y-axis list, as can be seen on line 98:

left = solve(collection[:len(collection) // 2], [pair for pair in sorted_y if pair.x < median_x])

Everytime we split we have to iterate twice on sorted_y (in order to ensure it is small). Although this is O(n) instead of sorting (O(n log n)), I still believe this to be the factor that is making it take, in general, longer than the basic solution.

Even though it has a higher r-value for x log^2 x, that doesn’t necessarily mean my algorithm is not O(n log n). It may just have higher constants making it slower (and dragging it towards the appearance of being O(n log^2 n) with our limited input.