K. Hohmeier

CSC 380

Homework 5 Written Report

Due October 22, 2020, 11:59 PM

<u>Problem 1: Implement Mergesort and Quicksort</u>

For this problem, the merge sort and quicksort algorithms, as described in the class textbook, were implemented in Python. For quicksort, the Hoare's partitioning algorithm was utilized to partition the first. Further experiments will also be done using a different method of partitioning the list (namely, by using the median-of-three method of choosing the partition from the first element, the last element, and the middle element of the list to be sorted) in order to compare the performance of quicksort under these two sorting methods.

First, I tested merge sort and quicksort (implemented with Hoare's partitioning) on random arrays of varying sizes. Quicksort and merge sort were able to perform quite efficiently, as the graph shown in Figure 1 shows. These algorithms appear to function at an efficiency of O(nlogn), with quicksort generally performing marginally better than merge sort based on its slightly faster runtimes. In this implementation of quicksort, the first element of the list to be sorted is chosen as the pivot.

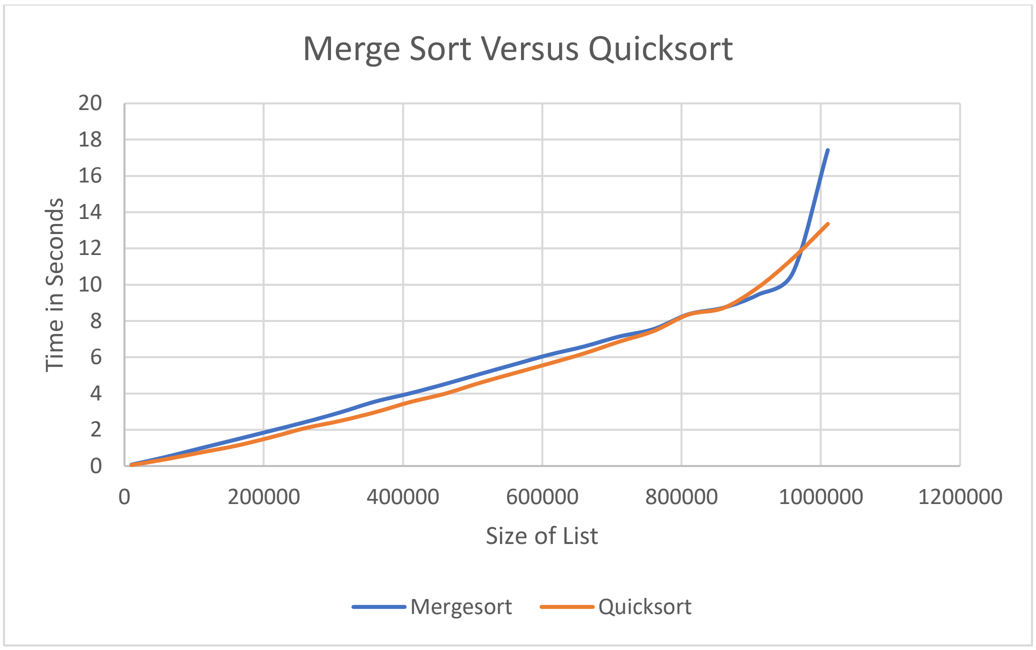
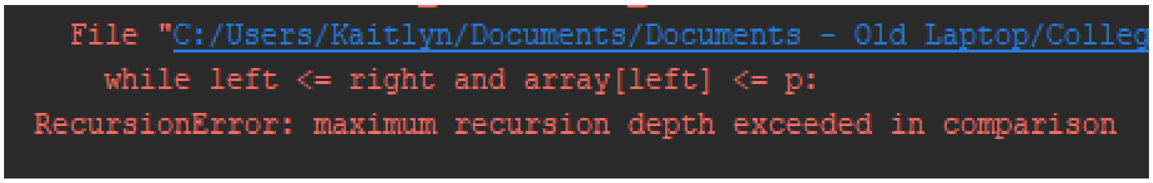


Figure 1. Graph of performance of merge sort versus quicksort (with pivot chosen as first element of the list)

However, quicksort, when implemented in this manner will break down if the list is not random. For example, quicksort will not even be able to operate on a list that is sorted in descending order of size greater than 1,000 elements (see the example output in Figure 2).



```
  File "C:/Users/Kaitlyn/Documents/Documents - Old Laptop/Colleg
    while left <= right and array[left] <= p:
RecursionError: maximum recursion depth exceeded in comparison
```

Figure 2. Error obtained if quicksort is run on a sorted list of a size larger than 1,000 elements.

Quicksort was implemented alternatively using a different pivot selection method: median-of-three. With this method, the pivot is selected based on the first element of the list, the last element of the list, and the element in the middle of the list. The median of these three items is chosen, and then that element. To test this version of quicksort with mergesort, two presorted lists were created and then concatenated together into one list. Then merge sort and quicksort were run and compared on this concatenated list. The results of the experiments of merge sort versus this alternative implementation of quicksort are shown in Figure 3.

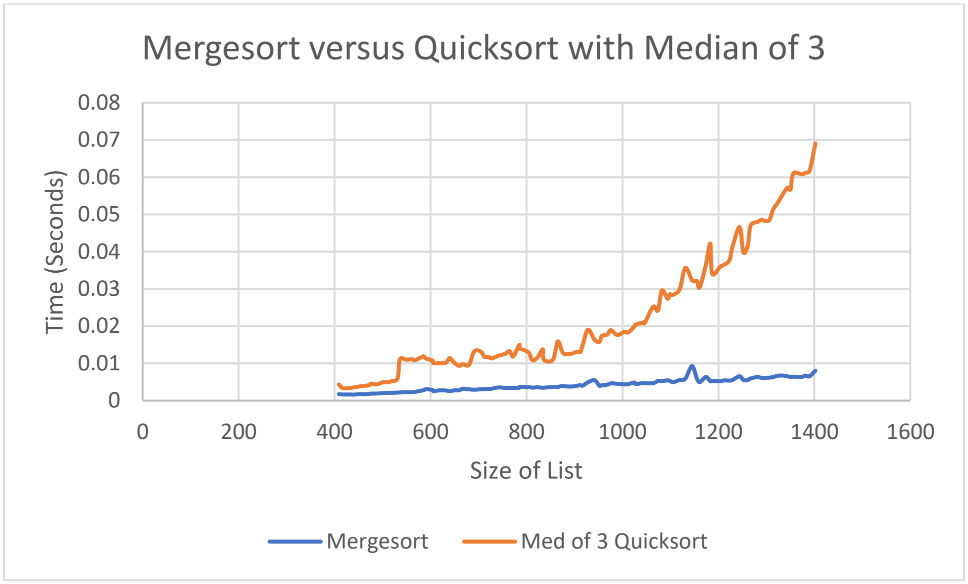


Figure 3. Comparison of mergesort versus implementation of quicksort that uses median-of-three pivot selection.

As can be seen in the graph above, when sorting the concatenation of two presorted lists, quicksort with the median-of-three pivot selection method appears to have O(n^2) efficiency, whereas merge sort has O(nlogn) efficiency.

Problem 2: Implement Efficient Closest Pairs Algorithm

In this section, an efficient algorithm for finding the closest pair of points in a list of points was implemented and compared to the brute-force closest pair algorithm implemented in a previous homework. The “efficient” version of the closest pair algorithm utilizes a divide-and-conquer approach to finding. We wish to compare the time efficiencies of these two algorithms via timing experiments. The results of these experiments are shown below in Figure 4.

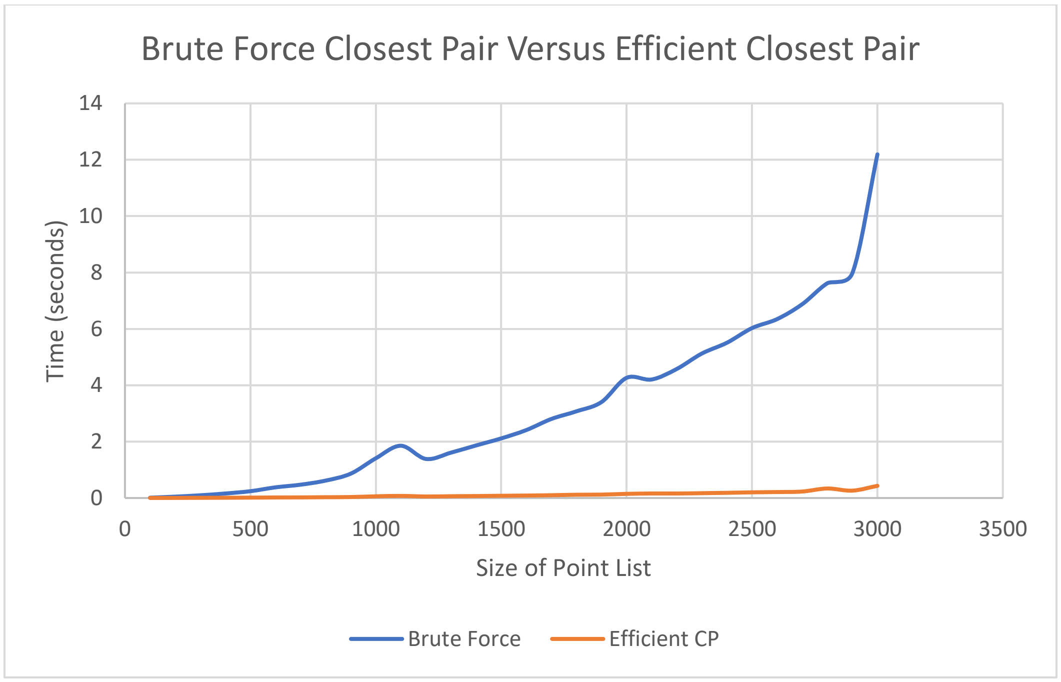


Figure 4. Comparison of efficiency of the brute force closest pair algorithm versus the "efficient closest pair" algorithm

Clearly, the divide-and-conquer algorithm for finding the closest pair of points is much more efficient than the brute force algorithm. The brute-force algorithm appears to be O(n^2), while the divide-and-conquer algorithm appears to be O(nlogn).