James Eli
CIS-269 Assignment #6
Sort Complexity

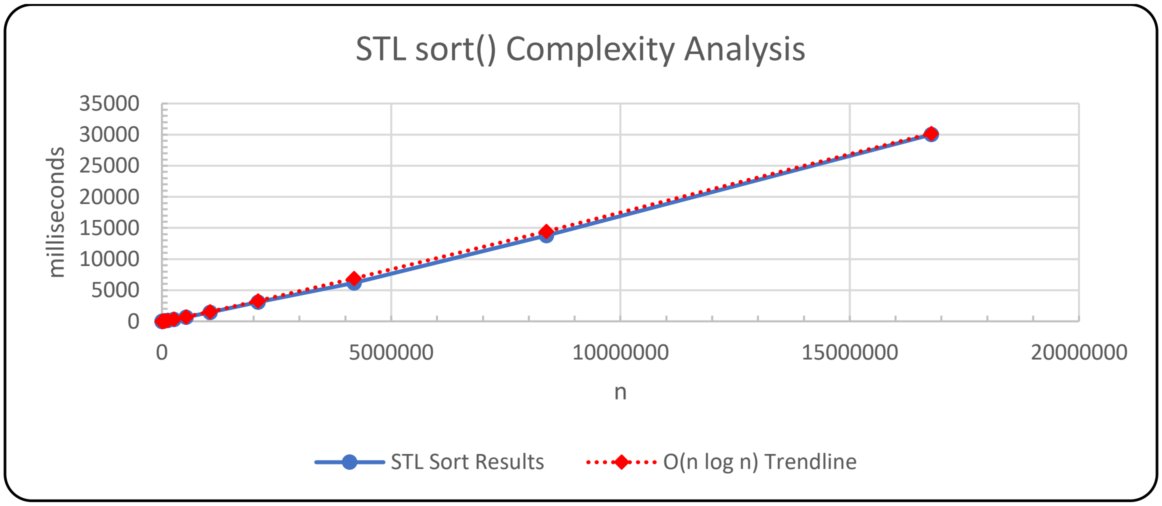
Part 1: Measure times of sort algorithms

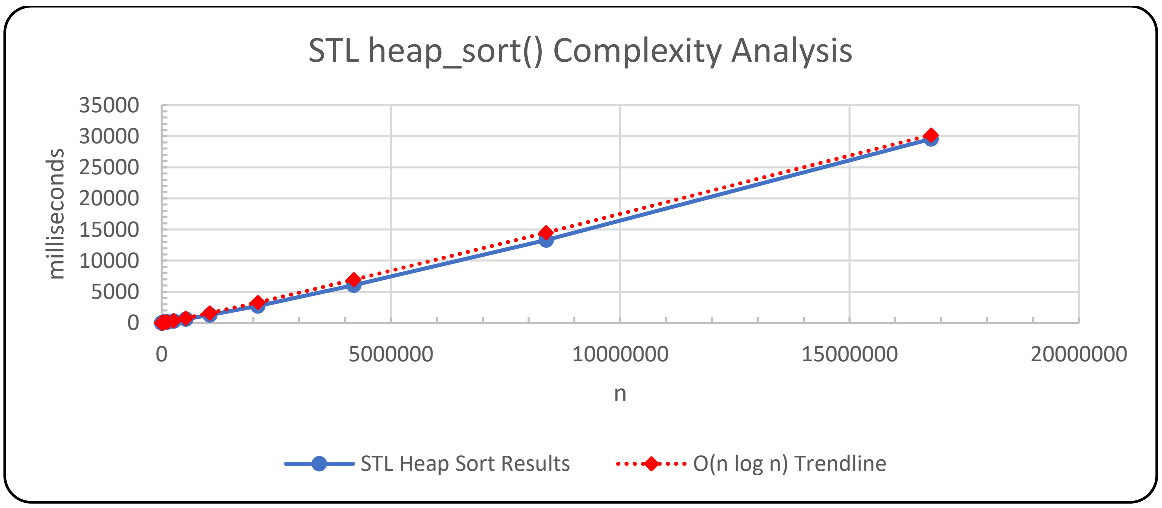
1. Both algorithms should have O(n log n) performance, do they?

   The STL *sort* algorithm had O(n log n) performance while the STL *heap_sort* algorithm had slightly better than O(n log n) performance.

2. How do you know? (Answer these questions in your documentation)

   Plotting the results of the sort benchmarks (see the two graphs below), I empirically solved the equation for the trendlines.





Part 2: Add another sort algorithm

1. What algorithm did you pick?

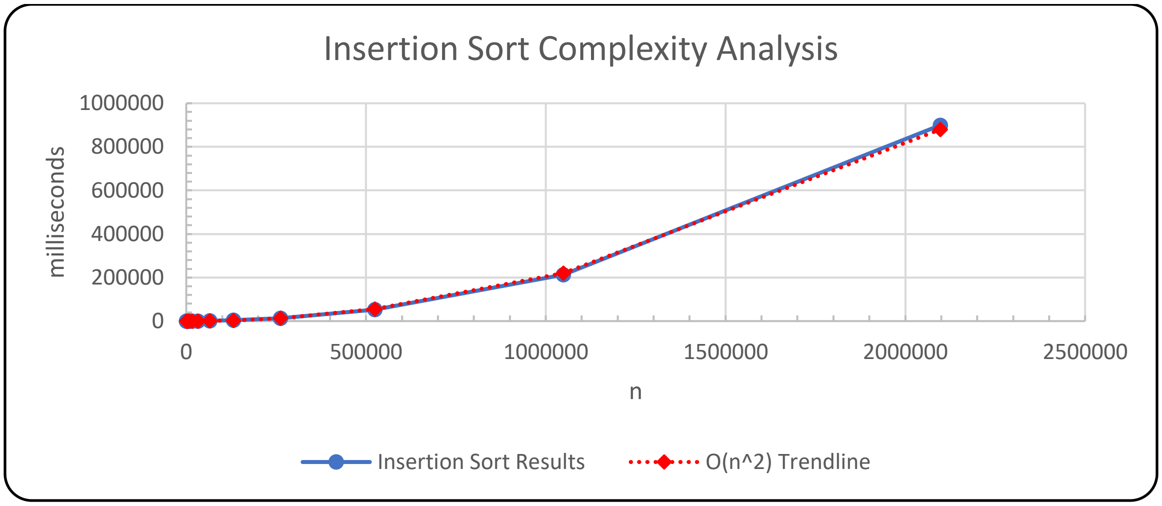
Insertion sort.

2. What is the complexity of the algorithm you picked?

O(n^2) was the expected performance.

3. Does the performance seem to match that performance?

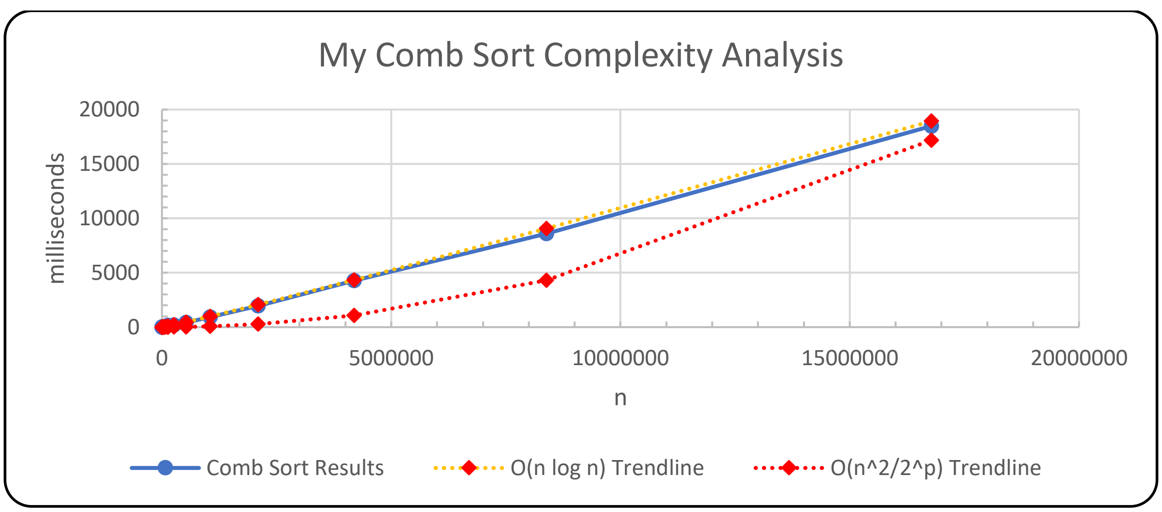
Yes, the algorithm matches O(n^2) performance:



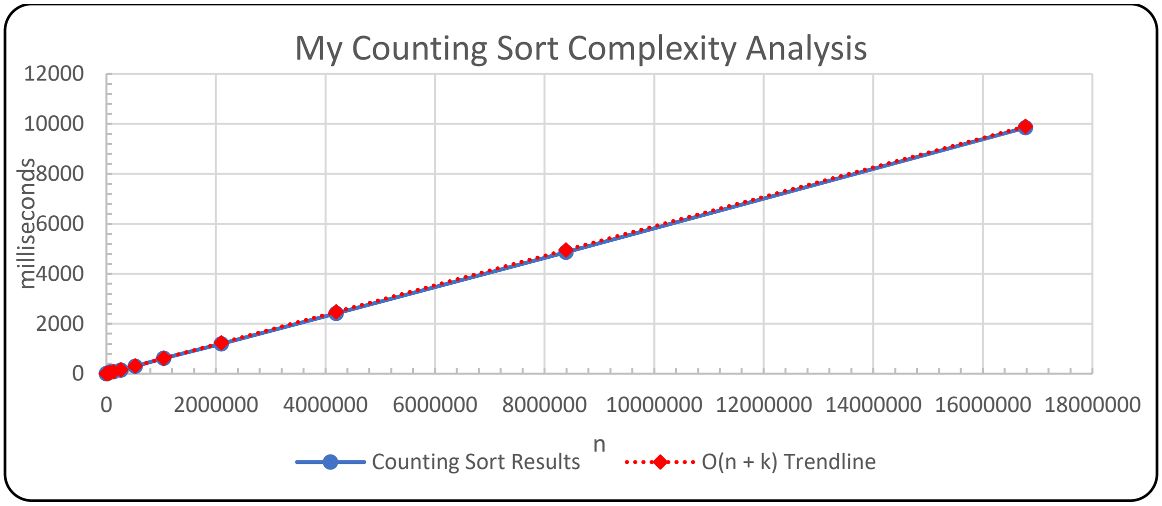
Extra Credit (5 points per algorithm)

I added comb, counting, parallel STL sort, and quick sorts (see included source code) with the following results:

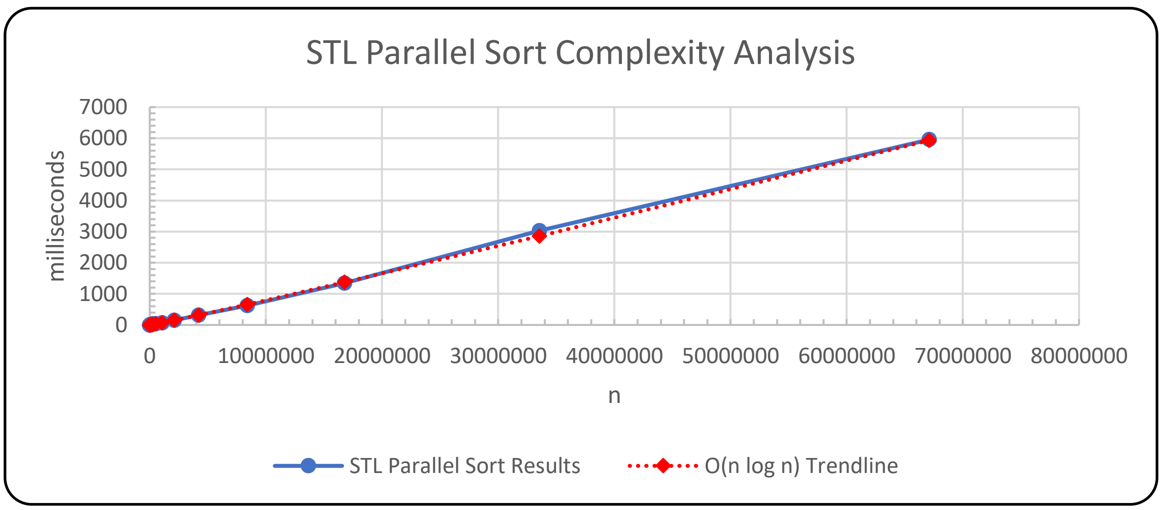
A. Comb sort expected performance is O(n^2/2^p) average, and O(n log n) best case. My function performs closer to the best case rather than average case:

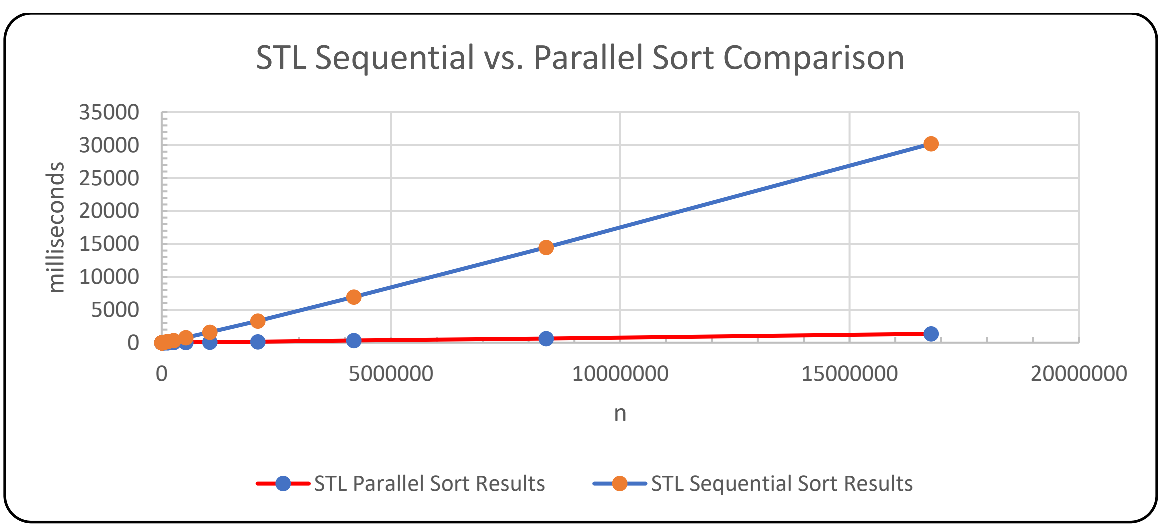


B. My counting sort O(n + k):

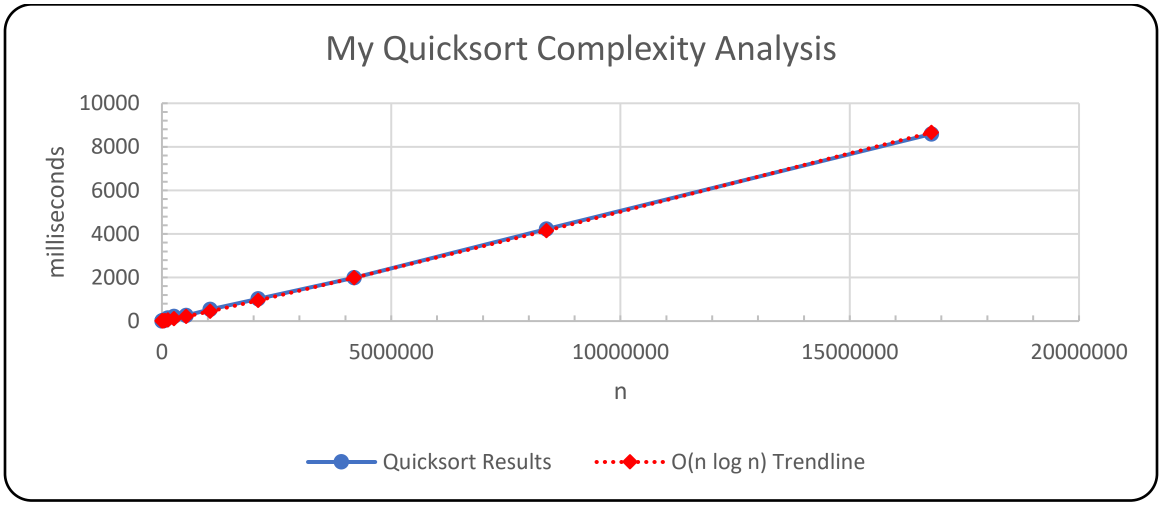


C. STL sort using a parallel (concurrent) algorithm matched expected O(n log n) performance. However, on arrays with more than 1,000 elements, the algorithm performed on average 20x better than the sequential version. My benchmark computer has only 2 cores. Note the comparison chart below.





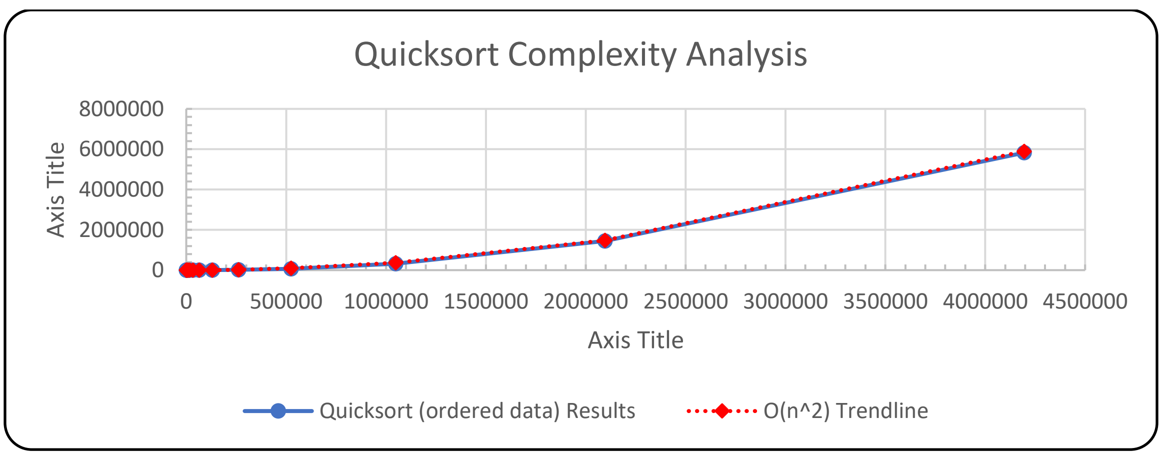
D. My quicksort matched expected O(n log n) performance:



Part 3: Ordered sort values

1. Is the performance significantly different than before?

   I benchmarked my quicksort algorithm with random verses ordered data (see included source code). The quicksort with ordered data was *significantly* slower.



2. Why or why not?

   When the data is ordered, quicksort selects a pivot value of the largest element in the array. When this happens, one of the partitions contains no elements and the other partition contains n−1 elements (all but the pivot). Therefore, quicksort recursively calls the subarray of n−1, which results in the series of O(n-1) + O(n-2) + O(n-3) + … O(1), which yields an O(n^2) run time (big-O, worst case).