Karthi Sankar

kasankar@ucsc.edu

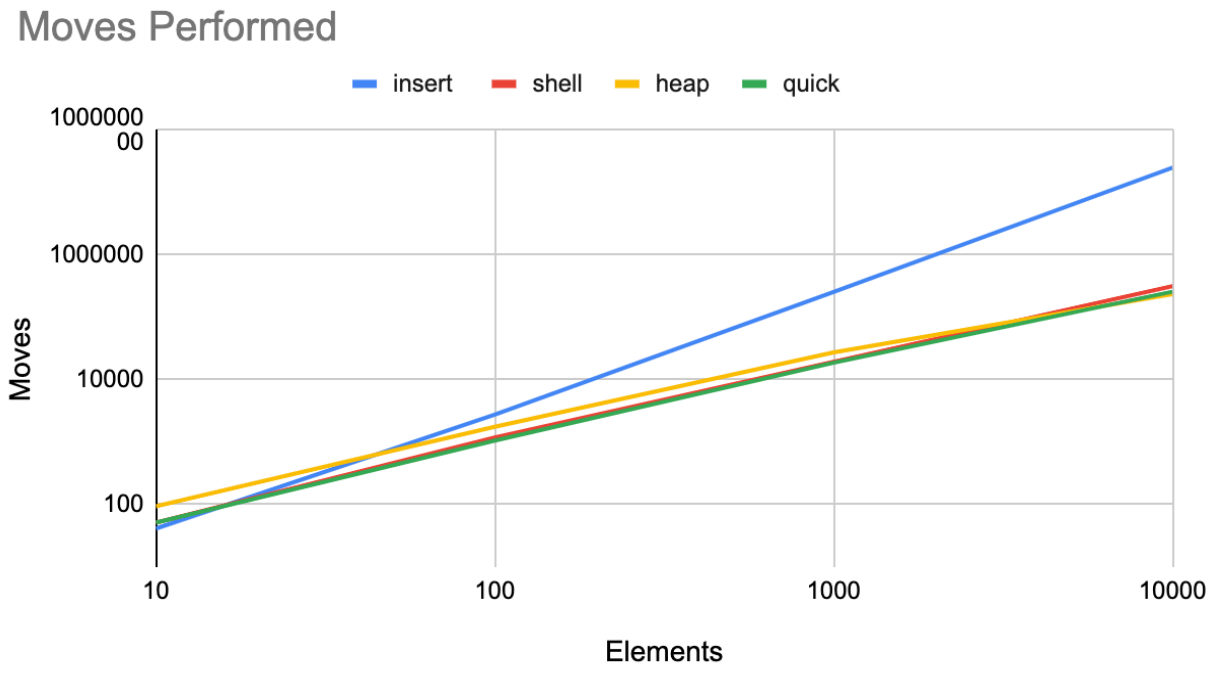
10/18/21

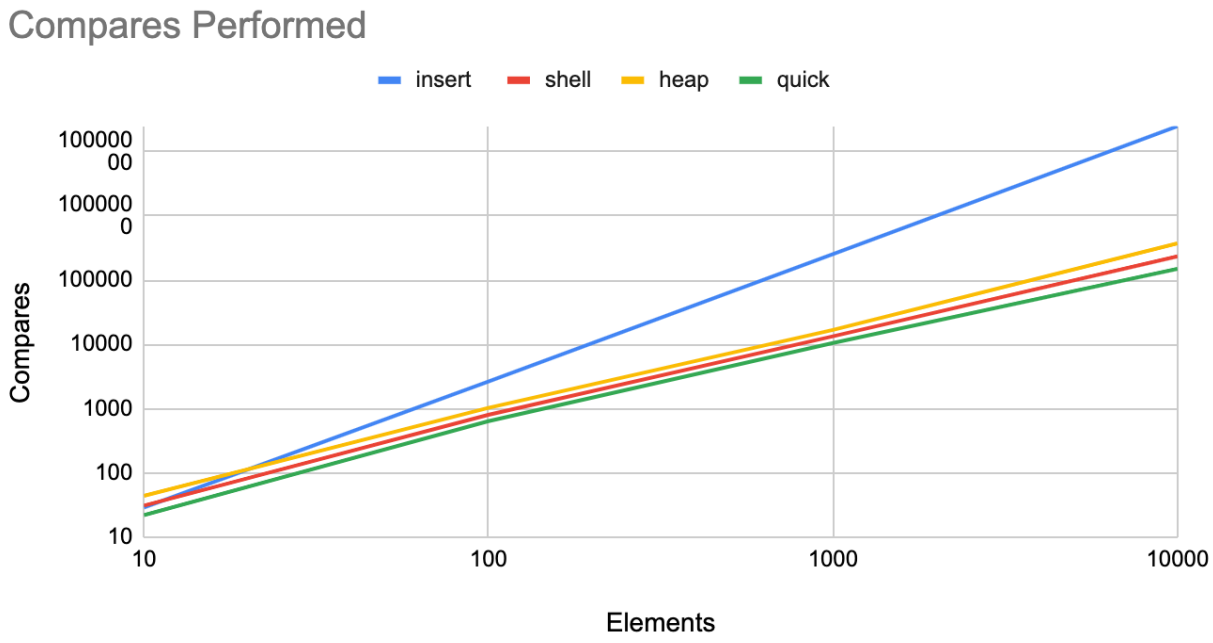
CSE 13S Assignment 3 Write-up

What I learned:

Generally, I learned about the sorts I had not heard of before. Insertion sort was really the only one that I had had experience with, so it was interesting to learn about more and understand how they work. I learned about time complexity for the first time. I obviously had heard of execution time, but not in any way of analyzing it. The time complexity of each of the sorts was different. Insertion sort has an O(n^2) time complexity. Shell sort has an O(n^(5/3)) time complexity. Heap sort has an O(n*log(n)) time complexity. Finally, quick sort has different cases. The average case for quick sort is O(n*log(n)), while the worst case is O(n^2). In terms of each sort, I would say the main things I learned are what a heap is (specifically max heap for this assignment) and how to recursively be able to sort an array, using quick sort. I learned how to represent a heap with an array and the rules about the children of parent nodes. In quick sort, I used recursion for sorting, which I hadn't before. The concept of splitting up the array and making the array you sort as you go smaller and smaller was really key to how the recursion in the algorithm actually worked, and it made it more efficient than the other algorithms in the end. I would say that the main way I experimented with the sorts was finding the best way to implement a generator in shell sort. In the end, I used a dynamically allocated array to contain the gaps that were used in the algorithm.

Graphs:





Analysis:

Both graphs generally show the same trend. The trend is that insertion sort is the most inefficient algorithm. Shell sort is slightly less efficient than heap and quick sort, which

are very similar in efficiency, but ultimately, quick sort has the best efficiency. It makes sense that insertion sort would not be as efficient because of the way it iterates through the array. It uses both a for and a while loop, leading to a time complexity of O(n^2). Because for an unsorted array, the algorithm would iterate through the array multiple times, it is not as fast for a larger number of elements. For shell sort, the gaps make the algorithm faster, though it is similar to insertion sort. For heap sort, the fixing of the heap in different stages throughout makes the overall sort faster for larger elements, while quick sort, as well, due to its use of recursion and partitioning the array, is fast for larger elements. What is interesting is that insertion sort works best for a small number of elements. At a certain point, the advantages goes to quick and heap sort, but in the beginning, insertion sort is more efficient. This is due to the fact that with a smaller number of elements, heap and quick sort have more overhead due to calling of other functions, and in the case of quick sort, using recursion, while insertion sort more quickly sorts with a for and while loop.