Vincent Siu
vsiu2
10/24/22
Asgn4.

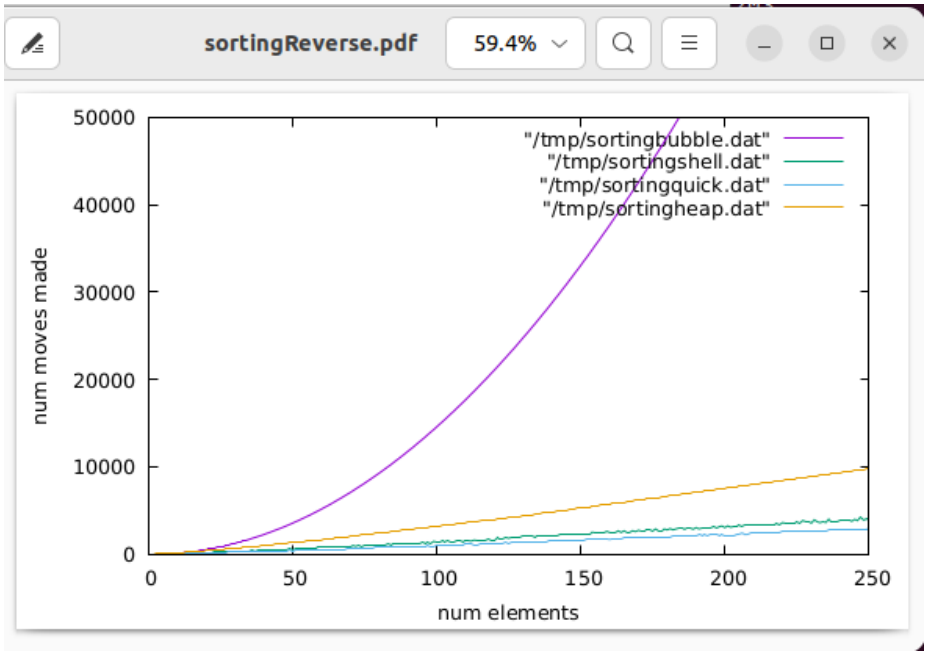
Writeup Assignment 4

1. What you learned from the different sorting algorithms. Under what conditions do sorts perform well? Under what conditions do sorts perform poorly? What conclusions can you make from your findings?
   The speed of different sorting algorithms heavily depends on the number of elements to be sorted, and, at times, how the array is sorted before undergoing sorting algorithms. For example, bubble sort is really fast when there are a few elements while quicksort can be slower, but every everytime you increase in elements, it becomes vice versa. Bubble has to read through every single element and do comparisons one by one and swaps if necessary and you have to do that over again until everything is in order. Quick is not iterative as bubble, since it instead uses a value to compare against and you repeat the process. However, quick can be very slow if the pivot is the smallest or largest element. This is why choosing approximating the median is better than fixing an element choice because there are possibilities where that element is the smallest or largest value. This is because it will not be able to properly split the array in half, especially when the array is already in sorted order. It becomes much harder to partition the array as you have to do it one by one instead of dividing the number of elements by 2. Heap sort and shell short are not the fastest, but their speeds are mainly between those of bubble and quick. They have rather consistent sorting speeds, as the number of elements increases. In conclusion, quick sort is the fastest sorting algorithm, even as the number of elements increases, but that also depends on how the pivot is picked. My findings below are supported by the following graph figure and analysis.

Vincent Siu
vsiu2
10/24/22
Asgn4.

2.Graphs explaining the performance of the sorts on a variety of inputs, such as arrays in reverse, and arrays with small/large number of elements.

Fig. 1 Sorting algorithms given an array in reversed order.



Purple = Graph for number of moves for Bubble Sort.
Green = Graph for number of moves for Shell Sort.
Blue = Graph for number of moves for Quick Sort.
Yellow = Graph for number of moves for Heap Sort.

In the figure above, when we are given an array that is sorted in reverse, bubble sort obviously takes the longest to sort. The more elements it's given, the time for it to sort seems to exponentially grow as seen in the curve. This is because it has to iteratively look through and swap through every element single to the order reversed. Quicksort is likely the quickest because, in opposition to swapping every single element, only half of the elements are swapped, given that the pivot should be close to the median due to the order of the array. The line for shell sort is not very smooth, which means that the moves made somewhat fluctuate, but stays in the general path where the number of moves increase with increasing elements. This is most likely due to the fact that there is not really a pattern in calculating the gap, but it’s just that it gets smaller.

Vincent Siu
vsiu2
10/24/22
Asgn4.

Fig 2. Sorting algorithms for number of elements from 0-10,000

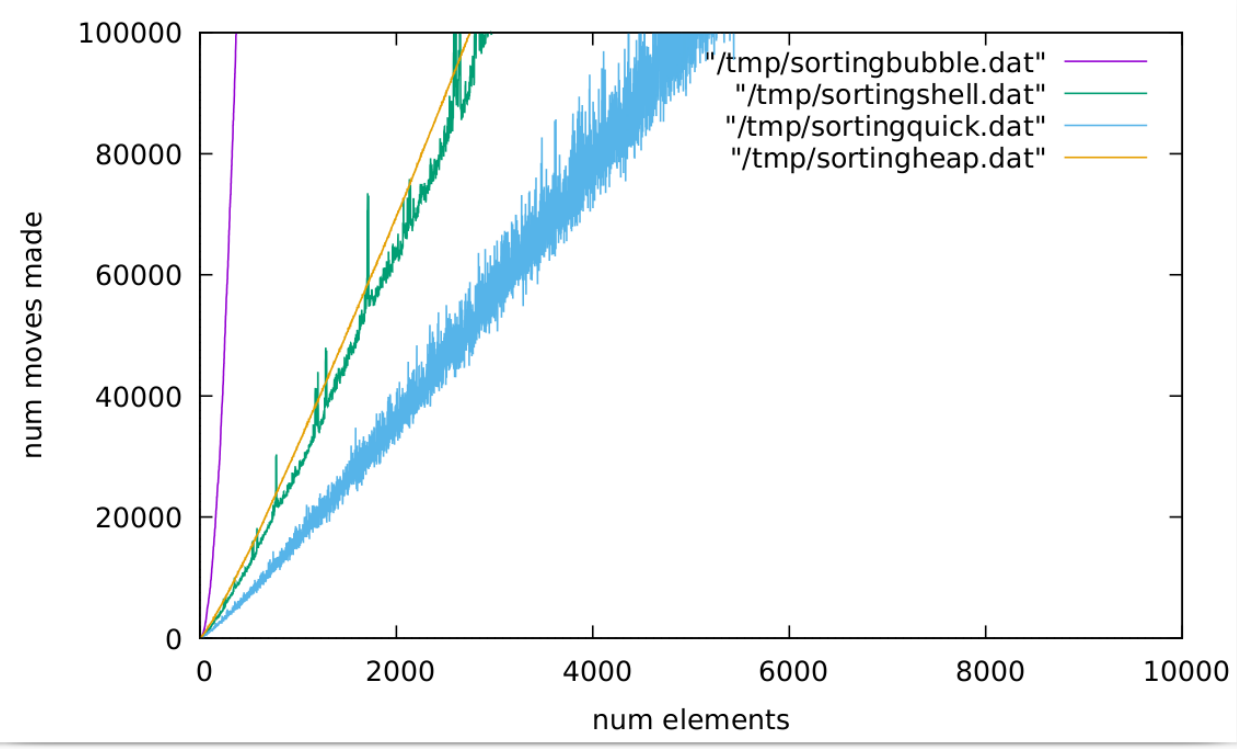
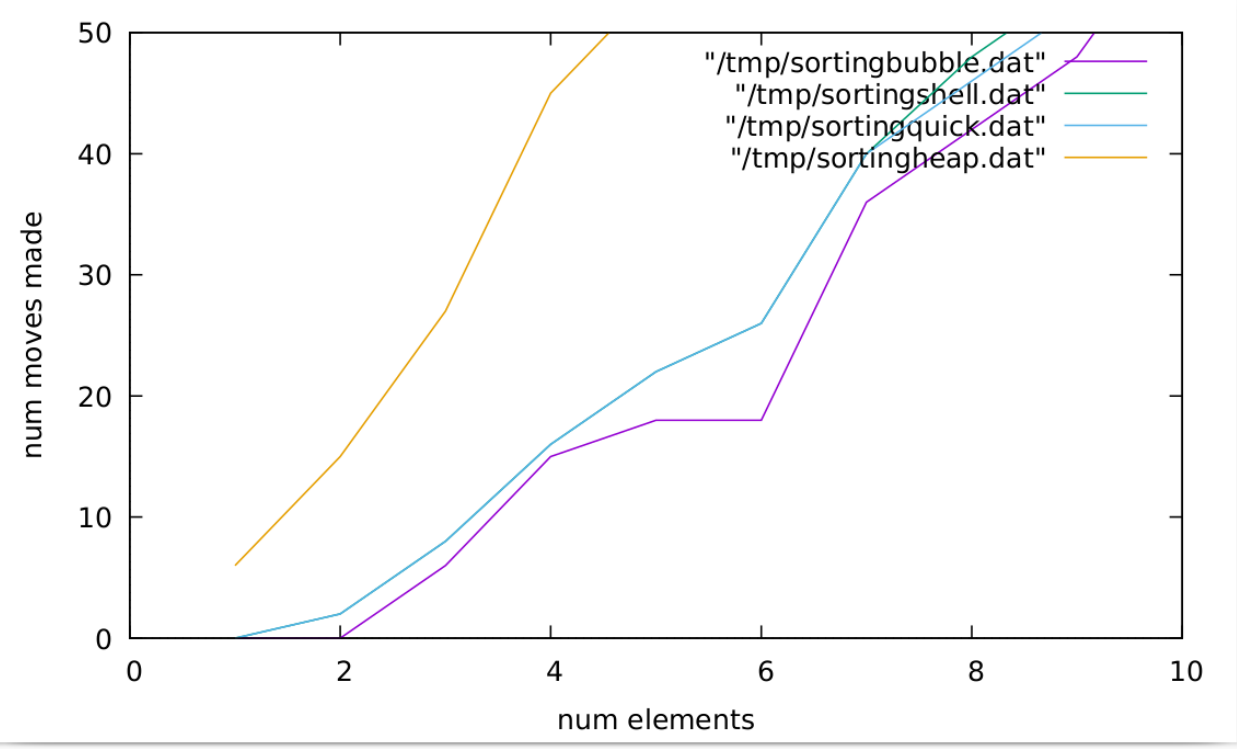


Fig 2a. Sorting algorithms for the number of elements from 0-10,000 looking closer at the zero axis.(paste figure here)



Vincent Siu
vsiu2
10/24/22
Asgn4.

To prove how I came to these conclusions for number 1, I ran the sorting algorithms starting from 0 elements to 10,000 elements under the default seed. In figure 2a, I zoomed in the graph of each of the sorting algorithms closer to the zero axis. Bubble sort seems to be the fastest at this point. However, if we zoom out to the full scope of the graph, the number of moves per each element increases substantially than all the other sorts. Quick, again, seems to be the fastest algorithm with the fewest number of moves per each increase in elements.