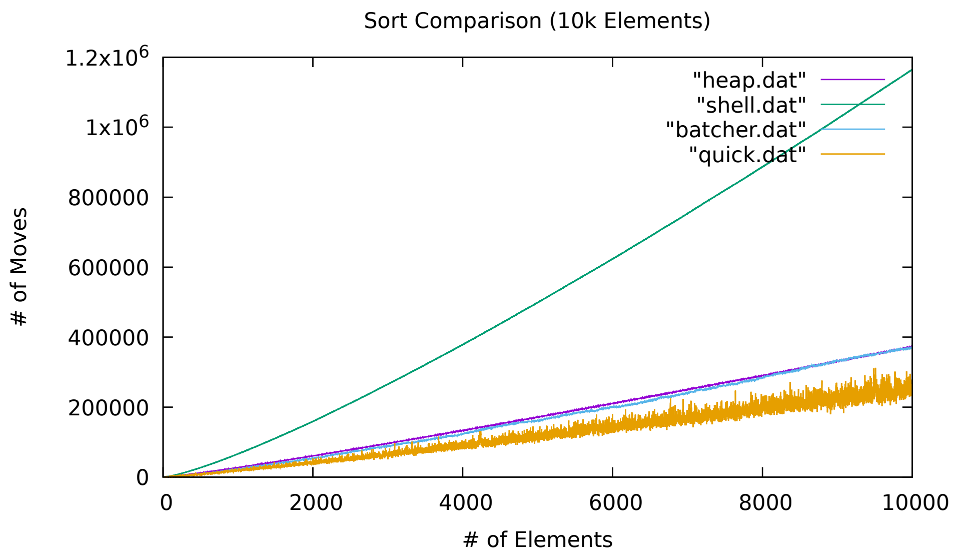
asgn3 writeup

Ryan Hui

rhui1

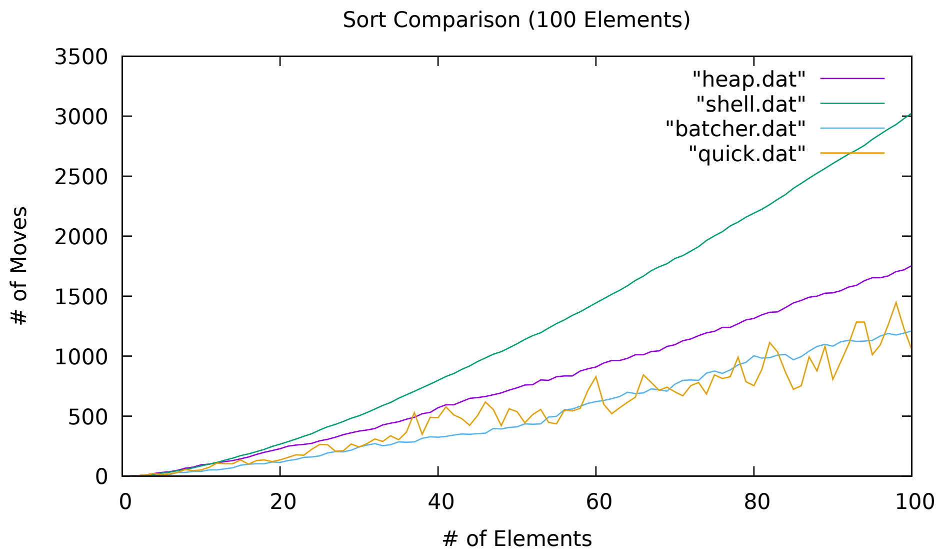
Sort Plotting

After doing asgn3, I can say that I have learned a lot about the different types of sorting algorithms. Each sort has their own pros and cons since they are all created differently. To examine how sorts perform given an array of elements, I have plotted all 4 sorts with a large number of elements of 10,000 and a smaller set of elements of 100.



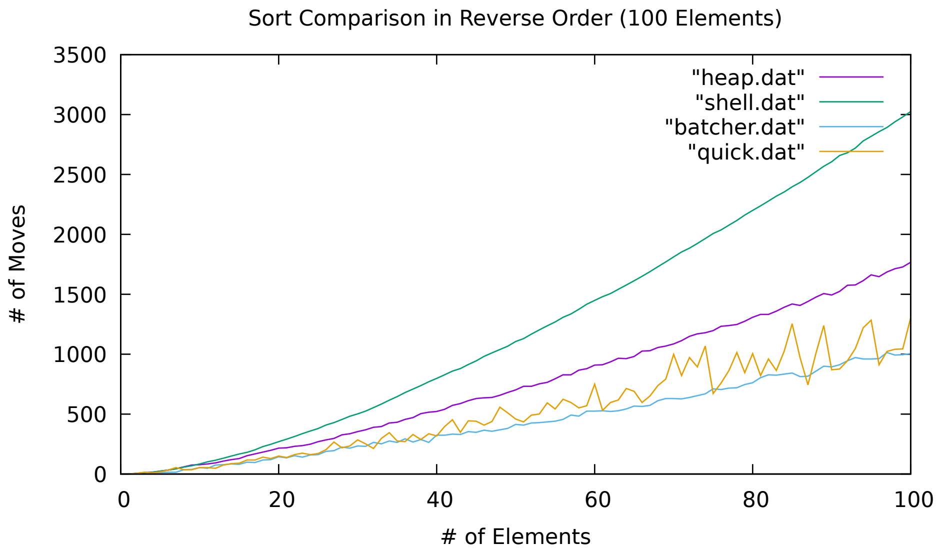
After looking at the plot for 10,000 elements, I can see that quick sort uses the least amount of moves, then it looks like batcher and heap are basically tied at moves with batcher being slightly under in moves up until 8000 elements. The sort with the most amount of moves is shell with more than twice the amount of moves of the other sorts.

It seems under the condition of a large array, quick is the most efficient sort. Heap and batcher are close behind in efficiency while shell is a lot worse then every sort. I notice that most of the lines except quick are relatively straight with shell being the straightest. Quick has a sporadic line which can mean that it's sorting moves has variation.



For the plot for 100 elements, it shows a different sorting ranking. It is shown that batcher is barely below quick in moves with quick having a similar move count but constantly dipping and and rising around batcher. This matches the sporadic pattern shown in the 10k element graph. Heap is a 3rd in move counts with a difference greater than the one shown in the 10k element graph. Finally, shell has the most moves again but the magnitude of difference to the other sorts is smaller than the difference shown in the 10k graph.

Under a small array condition, quick is still close to being the fastest sort but I would say batcher starts off having less moves and is more consistent in moves. Batcher improves when there are less elements. Heap is shown to decrease in efficiency with 100 elements while shell increases in efficiency given a small number of elements.



When I plotted the elements using sorts in reverse order (greatest to smallest), I got a similar graph to 100 elements in normal order. I think this is because most of these sort's efficiency don't really change while comparing elements in a different way. I did notice that quick sort's moves increased and is clearly less efficient than batcher now since it increased in moves and is above batcher's line. One reason could be that quick sort's move variation caused it to change because of the reverse order. Overall, the other sorts didn't change with reverse order.

To close, after analyzing both large amount of elements and small elements plots, I would quick would be considered the fastest sorts for both conditions. Sorts like batcher and heap are similar in efficiency while shell is clearly the slowest. I also notice that quick's move variation starts to increase as elements increase since the line's variance increases as well. One explanation for why quick, heap, and batcher are all better than shell could be that shell is the only n 3/2 sort while the rest of o log n sorts (batcher being o log 2n). Because of this, I learned that the average time complexity of sorts will affect is efficiency.