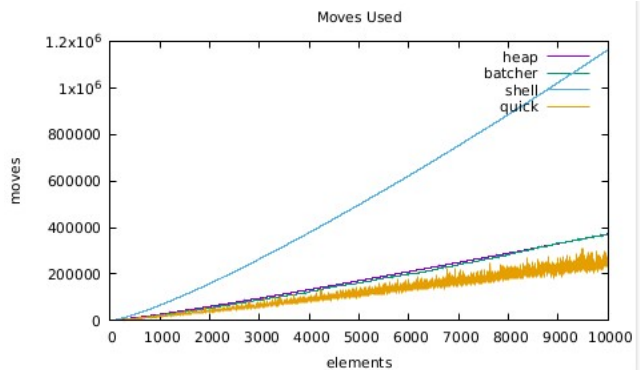
Assignment 2 Writeup

Caitlin Smith

February 6, 2023

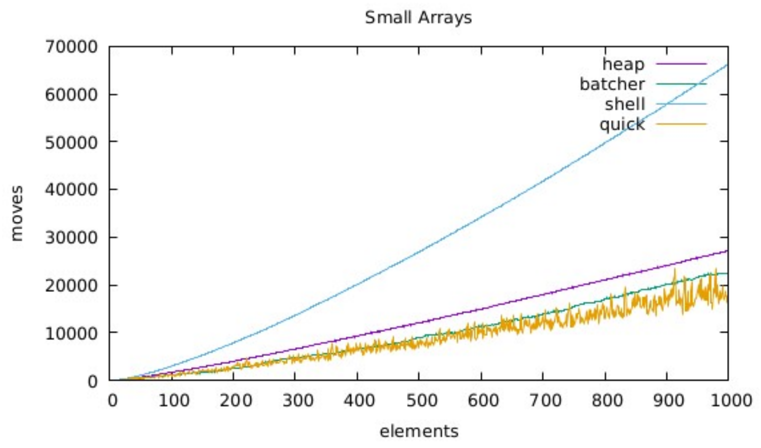
For all of the graphs below, I was able to slightly alter print statements in my functions to collect the correct data. I created a bash script that ran ./sorting with command-line options, putting data into .dat files, and plotted the graphs using gnuplot.

1 Comparing Moves



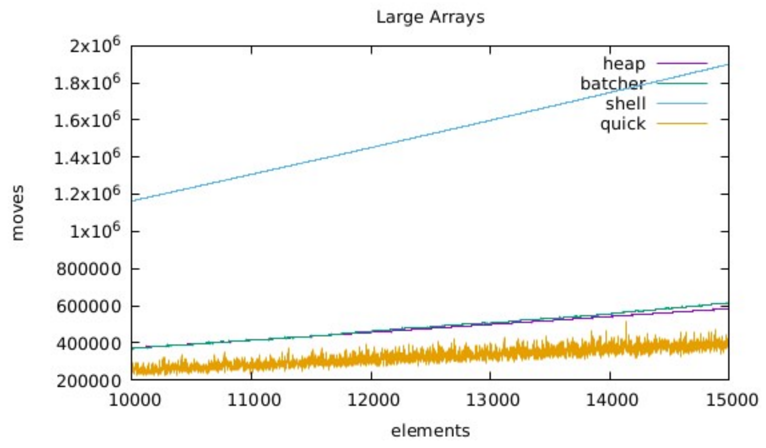
The graph above displays the number of elements passed when calling sorting and the number of moves it took to sort. There are lines for each of the sorting methods. I ran sorting with arrays up to the length of 10,000. The four different lines correspond to the different sorting methods. Shell takes a significantly larger number of moves to complete the sort, but the line is smooth. Batcher and Heap overlap for almost all data points, with some weaving. Quick is the fastest, but there are many jumps in the data, almost like zigzagging.

2 Small Arrays



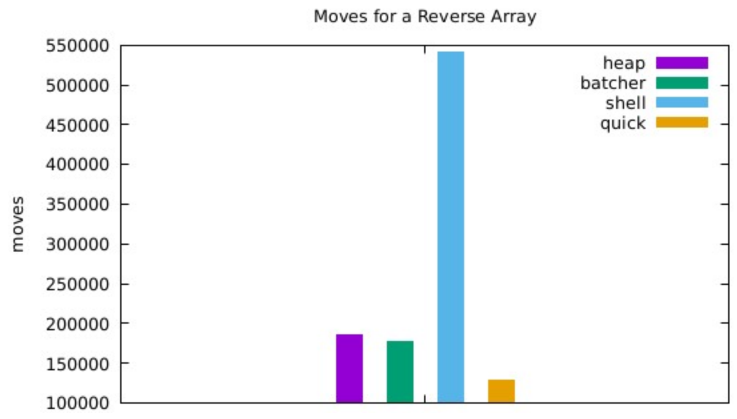
The graph above displays the number of elements passed when calling sorting and the number of moves it took to sort. I ran sorting with a small number of elements for the arrays, up to 1,000. This graph looks very similar to the previous one. Shell sort takes many more moves in order to sort completely. The others are much more efficient. Heap is visibly more efficient than Batcher this time, and Quick, again, is the best. The jumps in the Quick data is much more visible in this graph.

3 Large Arrays



The graph above shows the performance for arrays with large numbers of elements. In this case, I ran the sorts with arrays the size of 10,000 to 15,000. In this graph, Heap and Batcher are virtually overlapping again. Quick is very close to the bottom, jumping back and forth like the other graphs. Shell remains significantly the least efficient.

4 Array in Reverse Order



The graph above is a histogram that shows the number of moves it takes in order for the various sorting methods to sort an array that is in order from maximum to minimum (reverse order). The sorts were each called for an array going from 100 to 1. Shell Sort takes a significantly larger amount of moves to sort. It is incredibly inefficient for a reverse array. Batcher and Heap Sorts take about the same amount of moves. Quick was by far the most efficient.