Sorting Sorting Algorithms: Development Summary

Isabella Phung

January 2023

1 Introduction

This documents the specific issues and the rough process of putting together this library of sorting algorithms. It also analyzes the efficiency of these different sorting algorithms. The DESIGN.pdf document elaborates upon the design process of this library and contains pseudocode. The README.md file also contains information in regards to the structure and compilation of these files.

2 Notable Difficulties

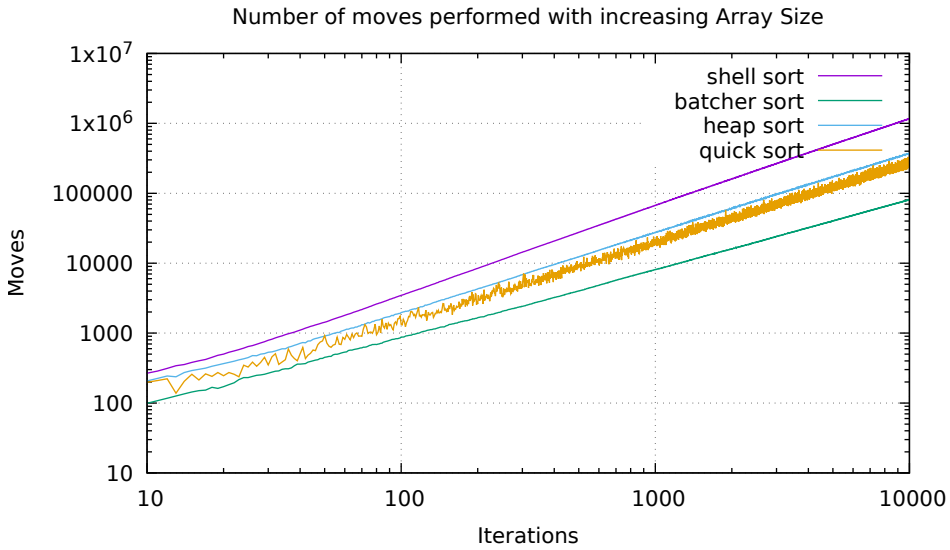
You'd think that putting together a library of algorithms with the python code already provided would be simple enough. It took exceptionally long trying to carefully comb through the code to determine what wasn't working. I took the python code provided in the spec and compiled it alongside my C code to check if my values were correct. There were two bugs that took the longest to unravel. One of which was simply because I wasn't paying proper attention to what belonged in what loop in the batcher.c file; the other was when I was missing a parenthesis when using the stats methods that wasn't performing the proper comparison in the heap.c file.

This project was incredibly intimidating at first. I have to admit, pointers frighten me. I've worked with pointers before and time and time again my code becomes incomprehensible. I think this was because I looked at the bv64.h file to understand what I needed to do for my set.c file. I initially overcomplicated things and thought I needed pointers every which way. But once I actually started working with it did I realize that it wasn't necessary and that the struct didn't need any pointers. I was also very confused about the difference between a typedef struct and a normal struct which required some research to understand.

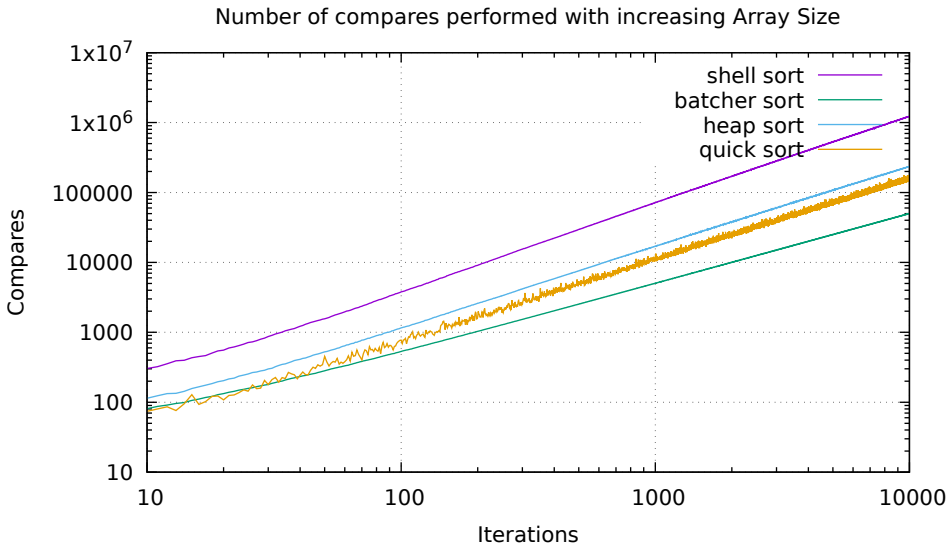
When I was trying to understand the python code for these sorting algorithms, it aggravated me to no end that most were written such that the first index would be 1 rather than 0. I did convert the quick.c file to start from 0, but after hours of trying to debug my heap.c file, it ultimately made more sense to just

stick with the 1 index rather than the 0 index.
Out of all of the algorithms, batcher sort probably was the most difficult to debug and understand. It took some time setting some of the variables to constants so that it wouldn't spit out a segmentation fault.

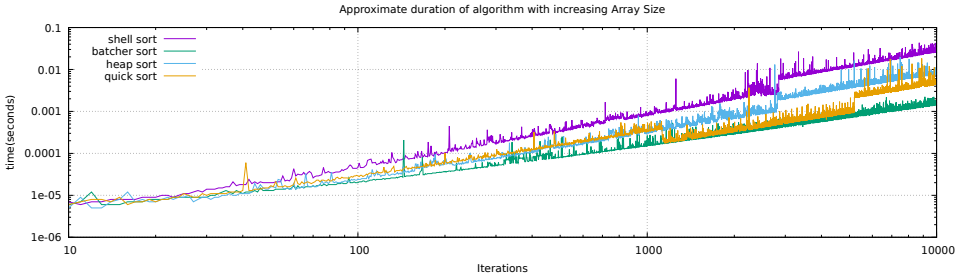
3 Graphs and Analysis



The above graph represents the total number of moves of shifts while using these sorting algorithms as the size of the array increases.



The above graph represents the total number of comparisons while using these sorting algorithms as the size of the array increases.



The above graph utilizes the time.h library in order to measure an approximate elapse time of the sorting algorithm. It uses the clock() function, which acquires the number of clocks from the start to the end of the sorting function, which then can be divided by the number of clocks per second to get a value in seconds.

4 Conclusion

Based off of the graphs, the most resource heavy algorithm seems to be shell sort, as it consistently has the highest number of moves, compares, and time elapsed in comparison to the other algorithms. Next would be heap, followed by quick, then batcher sort. In my initial prediction, which can be found in DESIGN.pdf, I hypothesized that batcher would be the slowest, followed by heap, shell, then quick sort. I thought that batcher sort might be slower as

it was known to be best utilized with multiple cores in parallel. However, it's performance is close to quick sort, and appears to be consistently faster and more efficient with larger arrays. The speed and efficiency of batcher sort is due to the fact it uses bit shifts, which are blindingly fast. Batcher also performs very few comparisons. Quick sort, although fast, has more erratic performance across all measurements. I thought heap sort would lag far behind quick sort, since it requires building the heap initially, it appears remarkably close to quick sort in resource usage and some moments in speed.

5 Resources

Structs vs. Typedef Structs Examples
Shell Sorting Diagrams
Heap Sort Wikipedia
Quick Sort Wikipedia
Batcher Sort
How to print stdints
How to print Pointers (for debugging)
Understanding Pointers
Understanding Time.h