The Power of Sorting

Mylo Lynch

January , 5, 2023

1 Intro

The power of sorting is unparalleled. Search algorithms on unsorted arrays are extremely slow compared to searching in sorted arrays. You could search through as many atoms as there are in the universe in less than a hundred iterations! In order to be even a decent programmer in the future, one must familiarize themselves with the theory, function, and implementation of multiple comparison-based sorting algorithms... or risk being fired for making faulty code that takes too long to run.

The sorting algorithms that will be examined are the shell, heap, quick, and batcher sort.

The classic shell sort algorithm was coded in c, using the pratt sequence to determine the gaps. Like an insertion sort but with varying gaps to save time.

The Heap sort creates a max binary tree. To sort, it moves the root (the max value) to the end of the array and fixes the binary tree according to the new root, until all roots are sorted.

Quick sort, recursively sets a partition and sorts through two halves of the array, comparing them to and moving them around the partition accordingly.

Batcher sort swaps a set amount of inputs across comparator networks. This makes it ideal for sorting that happens in hardware since it has finite parts.

2 Algorithm Results

Shell Sort, 100 elements, 3008 moves, 1558 compares

Batcher Sort, 100 elements, 990 moves, 1077 compares

Quick Sort, 100 elements, 1200 moves, 754 compares

Heap Sort, 100 elements, 3932 moves, 1266 compares

3 Algorithm Comparisons

Quick sort had the least compares and was the fastest. Meanwhile Batcher sort had the least amount of moves. This makes sense given what we learned about how Batcher sort is optimized for hardware (set amount of inputs). Heap sort used the most moves, which makes sense when you consider that you are building and fixing a heap many times, putting the root at the end each time until the heap is sorted. This sorting algorithm has to make moves both to create (and recreate) heaps, but to also create the sorted array. Shell sort is the simplest sort code-wise, but also not the most efficient as array size grew.

4 What Algorithm is Best?

That depends on what it is best at. Like previously mentioned batcher is great for hardware. Shell sort should be avoided for sorting arrays with very large arrays, as when it was tested with arrays much larger than the default size, it took whole seconds to sort through the array. Heap and Quick sort were the fastest with very large arrays, but it was ultimately the Quick sort which was the fastest, thanks to its “divide and conquer” approach to sorting.

5 Takeaways

In this assignment I learned how to properly and dynamically allocate memory for an array. I also learned the importance of keeping track of the sizes of everything and the numbers being passed as indexes in c code. C is ruthless and unforgiving, and a lot of problems came from properly calling pointers and the values of pointers within my code. Thankfully, I committed to git before destroying my code and was able to retrieve everything and properly use the swap function given to us as well as learned the importance of committing to git often. I apologize for the lack of graphs, I will be practicing this week to be able to make better graphs and be able to make them consistently.