Project Sorting Algorithms Report

Aaron Subichev

Kamran Khan

September 29, 2022

The following sorts performed best by order of original list based on comparisons:

* In Order - Insertion/Selection
* Reverse Order - Selection
* Almost Order - Selection
* Random Order - Selection

The following sorts performed best by order of original list based on movements:

* In Order - Insertion/Selection/Quick
* Reverse Order - Quick Sort
* Almost Order - Quick Sort
* Random Order - Quick Sort

The following sorts performed best by order of original list based on time:

* In Order - Insertion Sort
* Reverse Order - Quick Sort
* Almost Order - Quick Sort
* Random Order - Radix Sort

| Across ALL sorts | Best Time (ms) | Worst Time (ms) | Avg. Time (ms) |
| --- | --- | --- | --- |
| Insertion Sort | 2.17 | 1959.78 | 653.57 |
| Selection Sort | 2.04 | 1183.53 | 289.34 |
| Quick Sort | 2.48 | 7.94 | 4.60 |
| Merge Sort | 3.68 | 11.28 | 6.99 |
| Heap Sort | 10.05 | 73.13 | 31.9 |
| Radix Sort | 2.65 | 10.44 | 6.00 |

Interestingly enough, Selection Sort always contained N comparisons. This is likely because I counted the comparison of an element to the already sorted list as one comparison when in reality, it would probably be N comparisons again, if not log(N) + 2, only if you can traverse backwards. If my counts were more accurate, I’m sure radix sort would beat Selection in terms of comparisons because I believe Radix is, in fact, N comparisons.

Across the board, it is clear that Quick Sort wins. The only situation in which quick sort is genuinely, significantly worse, is a situation in which the given list is already sorted. While this is a possibility, performing 10x the amount of needed comparisons sounds terrible, but with a list of size 50,000, Quick Sort only loses to Insertion Sort by a millisecond. In fact, it actually performs twice as fast as Selection Sort despite the 10x comparisons.

This is clear evidence that there is more to a sorting algorithm besides comparisons and movements. Just to name a few factors that come to mind, a sorting algorithm also might have to manage memory allocation and increase code complexity to deal with special cases. The thing about optimization is that written code becomes less readable, but for that sacrifice, you gain increased performance.

Something else worth pointing out is Radix Sort’s performance across the board. In every situation, Radix Sort, in theory, should have performed the same amount of operations. After all, the idea behind Radix Sort is that it compares all digits of all 50,000 numbers. No matter if they’re in order, out of order, or whatever, it should always average around the same time. However, as per my extensive testing, we can tell that it performs significantly worse for a list in order than for any other. It’s interesting to think about since I didn’t even touch Radix Sort. To be clear, I didn’t change implementation in any of the sorts except for quick sort since it was giving me trouble, but besides that, I will trust that all implementations handed to me were correct and worked well. At the end of the day, I was able to collect results just fine and analyze them here.

One factor that is completely unpredictable and makes this project somewhat moot in my opinion is that the machines students perform the tests on will vary greatly. Some have gaming rigs while others light notebooks. Of course, the simple fact that no two students will have the same results is alright by itself, but when you consider the possibility of background processes running on certain computers, it makes it difficult to effectively predict the performance of some of these sorts. I’m not just referring to a constant load on the CPU in the background too, since certain processes run when they see fit and some are more taxing than others... there’s really a lot to consider. During my tests, I had a couple of weird results. Namely, Radix Sort performing poorly on In Order lists, and Heap Sort getting slower and clocking some abnormally high times on Almost Order. It’s just not feasible for me to shut everything down on my computer and run the tests in a sterile environment. Even at that point, there’s still room for improvement. Ideally, I shouldn’t go for a supercomputer since that would be *too* fast, but I need to get a new computer built to ensure it’s not full of malware or background processes, and that way I would be able to get the cleanest results possible.