Michael Permyashkin

Empirical Study of Sorting Algorithms

April 29, 2020

An Empirical Study: Sorting Algorithms

The following empirical study compared runtime efficiency for 3 sorting algorithms: mergesort, quicksort and heapsort, each implemented in python. Each algorithm was tested on 270 files of randomly generated integers in 3 formats, unsorted, sorted, and reverse-sorted. Groups are labeled: small, medium and large where n was 10,000, 100,000, and 1,000,000 respectively. Each group of test data represents best, average and worst case and allows us to measure the runtime efficiency for each case respectively. The theoretical time complexity analysis for each algorithm is shown below.

Time Complexity	Best	Average	Worst
Mergesort	$\Omega(n \log n)$	$\Theta(n \log n)$	$O(n \log n)$
Quicksort	$\Omega(n \log n)$	$\Theta(n \log n)$	$O(n^2)$
Heapsort	$\Omega(n \log n)$	$\Theta(n \log n)$	$O(n \log n)$

Each time complexity above ignores constant factors and therefore we may expect a competitive analysis when each algorithm is run on the test data. However these constant factors do play a role in practice and can be observed when each implementation is compared side by side.

In the best case when the array was sorted, quicksort had the most efficient runtime while heapsort the least. All 3 algorithms have equivalent time complexities, however we see an evident discrepancy in constant factors which results in different runtime efficiencies. In the average case where the array is not sorted, quicksort beat both mergesort and heapsort. In both the best and worst cases, quicksorts' performance can be attributed to the reduced amount of logic that is executed recursively. Quicksorts' strategy of dividing the problem to its simplest pieces recursively, as opposed to mergsorts' more linear approach, and then constructing the solution from smaller solutions is an effective improvement to mergesort. In the worst case where the array is in reverse sorted order, mergesort ran more efficiently with the exception of medium sized inputs. It is worth noting that although quicksort has a worst case time complexity of $O(n^2)$, our median-of-three approach to picking a pivot value resulted in a much faster runtime. Quicksort's true worst case would have been a reverse sorted array where the pivot is always the leftmost or rightmost value. As such, a median-of-three implementation to quicksort showed competitive results against the other 2 sorting algorithms. Heapsort did not show it's superior performance in this empirical study. In implementing the algorithm, it is evident why this may be the case. Heapsort is guaranteed to swap every single element in the heap, while other sorting algorithms, such as quicksort avoid unnecessary swaps.

In conclusion of this study, the data supports the argument that quicksort, although a slower time complexity in the worst case of $O(n^2)$, with a more efficient pivot selection we can improve its performance to beat even merge sort which has a worst case of $O(n \log n)$.

Key		
Small:	10,000	
Medium:	100,000	Fastest
Large:	1,000,000	Slowest

Sorted: Runtime Summary					
Unsorted	Mergesort	Quicksort	Heapsort	Overall Avg.	Overall St. Dev.
Small	48.858881	27.28293737	78.20568085	51.4491664	25.56000068
Medium	478.7210385	276.3173421	833.1008434	529.3797414	281.8274205
Large	5720.05167	5204.430715	9215.810235	6713.430874	2182.405346
Overall Avg.	2082.543863	1836.010332	3375.705587		
Overall St. Dev.	3157.497847	2919.793904	5071.743626		

Unsorted: Runtime Summary					
Unsorted	Mergesort	Quicksort	Heapsort	Overall Avg.	Overall St. Dev.
Small	49.31169351	24.38139915	64.7876819	46.16025819	20.386652
Medium	580.4796537	313.8713678	820.7049529	571.6853248	253.5312129
Large	7296.410306	5996.961363	11343.45207	8212.274581	2788.430787
Overall Avg.	2642.067218	2111.738043	4076.314902		
Overall St. Dev.	4039.519407	3367.814023	6304.864369		

Reverse Sorted: Runtime Summary					
Unsorted	Mergesort	Quicksort	Heapsort	Overall Avg.	Overall St. Dev.
Small	40.97599188	99.95833238	63.19564184	68.04332203	29.78848981
Medium	470.1456149	382.9897881	787.356178	546.830527	212.8108929
Large	5548.029923	13892.99048	9691.507538	9710.842646	4172.513876
Overall Avg.	2019.717177	4791.979533	3514.019786		
Overall St. Dev.	3063.133978	7882.977031	5362.100178		