Michael Thoreson
Adam Neeley
Richard Bennett
09/30/2024

Week 06 Lab - Fast Sort Comparison

When comparing the fast sort operations (Merge sort and quicksort), there was a considerable disparity in performance for each algorithm depending on the structure of the array it was used to sort. Once again we elected to use the total number of measured operations as a metric for performance. Because these algorithms are called recursively, we also measured recursive calls as an indicator of performance. Our group found the following values:

Random Values

	Comparisons	Moves	Recursive Calls	Total operations	Time (ms)
Mergesort	120,476	267,232	19,999	407,707	1435
Quicksort	**138,683**	**218,616**	**12,916**	**370,215**	**592**

Already Sorted Values

	Comparisons	Moves	Recursive Calls	Total operations	Time (ms)
Mergesort	69,008	267,232	19,999	356,239	1115
Quicksort	**124,107**	**188,409**	**12,286**	**324,802**	**218**

Nearly Sorted Values

	Comparisons	Moves	Recursive Calls	Total operations	Time (ms)
Mergesort	70,668	267,232	19,999	357,899	1023
Quicksort	**126,351**	**196,350**	**12,382**	**335,083**	**257**

Reversed Values

	Comparisons	Moves	Recursive Calls	Total operations	Time (ms)
Mergesort	**64608**	**267232**	**19999**	**351,839**	**992**
Quicksort	12,519,993	18,787,488	14,995	31,322,476	20700

Few Unique Values

	Comparisons	Moves	Recursive Calls	Total operations	Time (ms)
Mergesort	**94,985**	**267,232**	**19,999**	**382,216**	**1087**
Quicksort	1,363,009	154,656	29,410	1,547,075	943

Overall, do you think one of the sorts performs the best overall? If not one sort is the clear winner, then which sort would you least want to use overall?

In general, Quicksort seems to outperform Mergesort and slower sorting algorithms. In most cases Quicksort is probably the best choice.

Which sort worked best in each scenario? Which sort did the worst?

Scenario	**Best**	**Worst**
Random values	Quicksort	Mergesort
Already sorted	Quicksort	Mergesort
Mostly sorted	Quicksort	Mergesort
Reverse sorted	Mergesort	Quicksort (by far)
Many duplicates	Mergesort	Quicksort (by far)

For the random, nearly sorted, and already sorted arrays, we found that Quicksort had a marginal advantage over Merge sort in each test case. There was never more than a distance of about 50,000 operations, however. It is worth noting that Merge sort greatly outperformed quick sort for arrays of few unique values or values sorted in reverse order. Because of this, quick sort would likely be more performant in a large number of applications, but if performance is a concern, merge sort would be best suited for arrays with many duplicates.

Did the "efficient" O(n log n) recursive sorts always work better? When did they do better? When did they do worse (if that happened at all)?

No. Mergesort did better with reverse sorted values. Otherwise, Quicksort generally performed better. But when the recursive sorts are compared to the slower sorts, bubble sort is way more efficient comparison-wise when all of the values in the array are already sorted, meaning it is also the best performance-wise. All the non-recursive sorts do approximately 0 moves as well, which is better than the recursive ones. In a nearly sorted array, Selection Sort does the least amount of moves, and insertion sort does the least amount of comparisons. The general best one being insertion sort by about 10 fold over the recursive sorts. Even Bubble sort is still better than the recursive sorts, only doing about 127,000 total operations compared to the over 300,000 comparisons for both. However, when the total non-recursive operations are compared to the recursive sorts in the random, reversed and few unique categories, it is evident that the recursive sorts perform far better.

Try changing the number of items in the array – this can be done by changing the ARRAY_SIZE variable in main.cpp. (I recommend also keeping the MAX_VALUE to be 10x the ARRAY_SIZE).

ARRAY_SIZE = 100

```
        *** Running sorts on RANDOM values ***
        MERGESORT SORT REPORT: 551 comparisons, 1344 moves, 199 recursive calls
        QUICKSORT SORT REPORT: 635 comparisons, 1014 moves, 133 recursive calls
total:  merge - 2094, quick - 1782

        *** Running sorts on ALREADY SORTED values ***
        MERGESORT SORT REPORT: 356 comparisons, 1344 moves, 199 recursive calls
        QUICKSORT SORT REPORT: 561 comparisons, 873 moves, 109 recursive calls
Total: merge - 1908, quick - 1543

        *** Running sorts on MOSTLY SORTED values ***
        MERGESORT SORT REPORT: 369 comparisons, 1344 moves, 199 recursive calls
        QUICKSORT SORT REPORT: 576 comparisons, 879 moves, 118 recursive calls
Total: merge - 1912, quick - 1573

        *** Running sorts on REVERSE SORTED values ***
        MERGESORT SORT REPORT: 316 comparisons, 1344 moves, 199 recursive calls
        QUICKSORT SORT REPORT: 1443 comparisons, 2238 moves, 145 recursive calls
Total: merge - 1859, quick - 3826

        *** Running sorts on MANY DUPLICATES values ***
        MERGESORT SORT REPORT: 356 comparisons, 1344 moves, 199 recursive calls
        QUICKSORT SORT REPORT: 5243 comparisons, 588 moves, 295 recursive calls
Total: merge - 1899, quick - 6126
```

ARRAY_SIZE = 100000

```
        *** Running sorts on RANDOM values ***
        MERGESORT SORT REPORT: 1536663 comparisons, 3337856 moves, 199999 recursive calls
        QUICKSORT SORT REPORT: 1779289 comparisons, 2806821 moves, 129232 recursive calls
Total: merge - 5074518, quick - 4715342

        *** Running sorts on ALREADY SORTED values ***
        MERGESORT SORT REPORT: 853904 comparisons, 3337856 moves, 199999 recursive calls
        QUICKSORT SORT REPORT: 1541267 comparisons, 2351874 moves, 103393 recursive calls
Total: merge - 4391759, quick - 3996534
```

```
        *** Running sorts on MOSTLY SORTED values ***
        MERGESORT SORT REPORT: 869827 comparisons, 3337856 moves, 199999 recursive calls
        QUICKSORT SORT REPORT: 1582605 comparisons, 2458065 moves, 115300 recursive calls
Total: merge - 4407682, quick - 3906534

        *** Running sorts on REVERSE SORTED values ***
        MERGESORT SORT REPORT: 815024 comparisons, 3337856 moves, 199999 recursive calls
        QUICKSORT SORT REPORT: 1250199993 comparisons, 1875374988 moves, 149995 recursive calls
Total: merge - 4352879, quick - 2000544976

        *** Running sorts on MANY DUPLICATES values ***
        MERGESORT SORT REPORT: 1284799 comparisons, 3337856 moves, 199999 recursive calls
        QUICKSORT SORT REPORT: 7653326 comparisons, 2291700 moves, 294019 recursive calls
Total: merge - 4822654, quick - 11339125
```

How does making the array bigger/smaller affect the relative results of the sorts? How do they measure up with 10 items in the array? 100? 1000? 20,000? (Warning: setting ARRAY_SIZE to values larger than 20,000 means you can go get a cup of tea or maybe even lunch. And the folks who run CS50 might not like it if we hog all the CPU time!) Do the number of comps and/or moves in Merge sort and Quicksort seem to roughly correspond to an O(n log n) sort? Do the O(n log n) sorts get noticeably better relative to the O(n^2) sorts as the size of the array increases?

Both quick and merge sorts compare very similarly at large and small arrays, and the number of operations that each one requires scales almost proportionally to the size of the dataset. This is a strong contrast to the vastly diminishing returns that some of the slow sort algorithms showed as their arrays got larger and larger. It is also interesting that the difference in performance between the two fast sorts is so consistent across array sizes, whereas different slow algorithms outperformed others depending on the size of the array.

Feel free to add your observations about these sorting algorithm comparisons.

Using the total operations performed as a metric, it would seem that Merge sort is the safest choice, because despite the frequent performance benefit provided by quick sort, it is significantly less costly when processing reversed lists or lists with many duplicates. However, measuring performance in this way is somewhat inaccurate. However, when we measured the time that each algorithm took, quick sort had a much more considerable time advantage compared to merge sort. It even outperformed merge sort on some reverse sorted arrays (provided the arrays were under a certain size). Because of this, there is reason to believe that quicksort is the best choice out of the two.