Analysis of Various Sorting Algorithms

by Isaac Lee

Abstract: An analysis of the number of variable reads and writes that result from a vector going through various sorting algorithms. The vector consists of a subset of the full data set, which is a file that contains 16,599 video games ranked from the best sales worldwide to the worst. The vector is put through each sort in sizes from 100 to 1000, with the size increasing by 100 each iteration (i.e. 100, 200, 300, etc.). The sorting algorithms used here are: bubble sort, insertion sort, quick sort, heap sort, and a two-sort, which consists of sorting by the global sales of a subvector using merge sort, then by the japanese sales of that sorted vector using radix sort.

Data Set Attributes:

1. Rank - The rank of the video game based on sales number (Rank 1 ==> most sales)
2. Name - The name of the video game
3. Platform - The platform/console the video game was released on
4. Year - The year of the video game's release
5. Genre - The genre of the video game (Platformer, Simulation, Racing, Fighting, etc.)
6. Publisher - The publisher of the video game (Nintendo, Sony, Microsoft, etc.)
7. NA Sales - The total number of sales in North America
8. EU Sales - The total number of sales in Europe
9. JP Sales - The total number of sales in Japan
10. Other Sales - The total number of sales in all other regions
11. Global Sales - The total number of sales worldwide

Data Source:

This data was downloaded from https://www.kaggle.com/gregorut/videogamesales. I chose this data because I enjoy video games far more than I should, to the extent where I now run a gaming club and am considering a career in game development. This just seemed like it made sense to use.

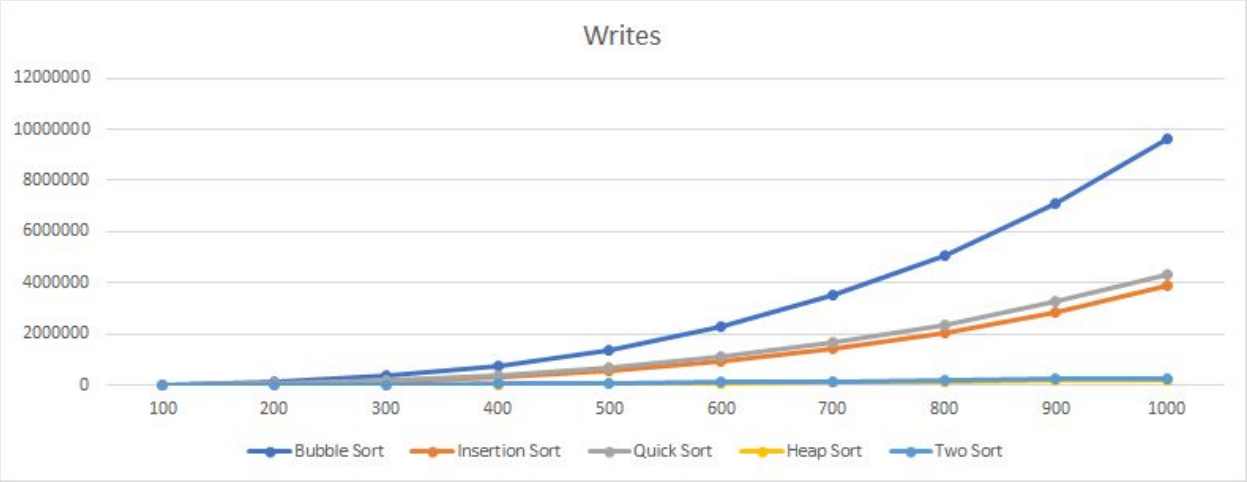
Entry Order by Default:

Entries are ordered from the video game with the highest global sales (Rank 1), to the video game with the lowest global sales (Rank 16599).

Data:

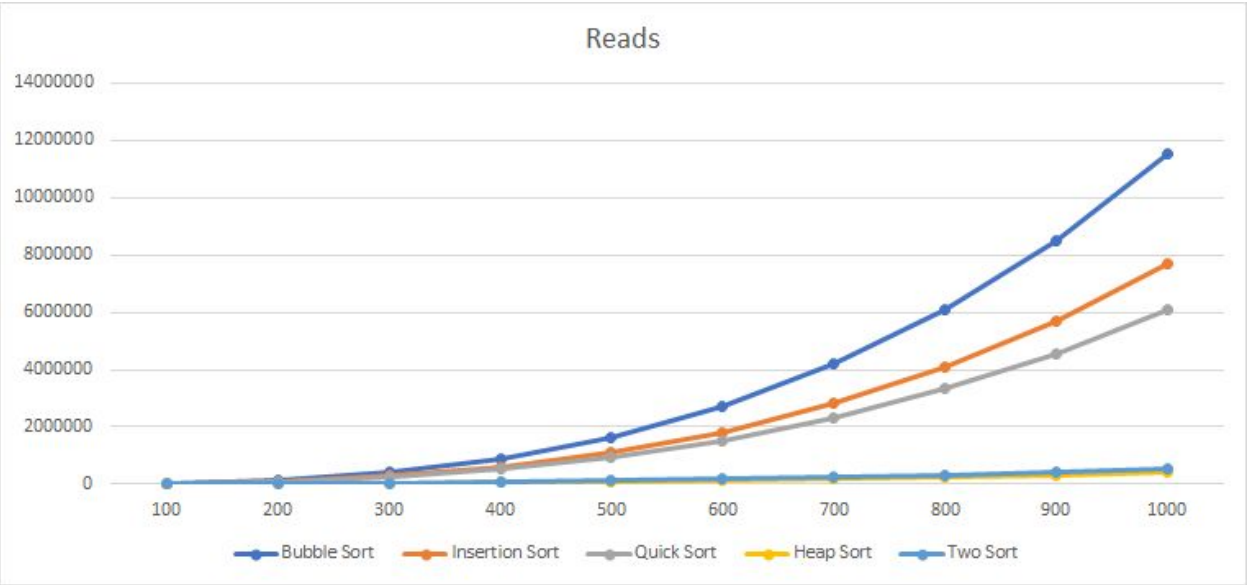
Number of Writes per Sort per 100 items:

	100	200	300	400	500	600	700	800	900	1000
Bubble Sort	24837	124445	348792	747761	1371229	2269040	3491192	5087444	7107652	9601597
Insertion Sort	10385	51120	142179	303488	554961	916498	1408099	2049594	2860917	3861970
Quick Sort	12805	62989	174139	368730	668215	1091668	1656242	2377340	3268216	4341409
Heap Sort	2771	9164	19531	34114	53010	76356	104333	136923	174325	216418
Two Sort	3657	11974	25311	43948	67885	97562	133039	174316	221393	274270



Number of Reads per Sort per 100 items:

	100	200	300	400	500	600	700	800	900	1000
Bubble Sort	29680	148958	417786	896012	1643468	2719950	4185456	6099656	8522382	11513366
Insertion Sort	20275	100750	281373	601996	1102447	1822526	2802233	4081228	5699379	7696490
Quick Sort	17866	88005	243440	515642	934644	1527158	2317203	3326348	4573151	6075169
Heap Sort	5066	16976	36494	64122	100035	144532	198024	260469	332264	413142
Two Sort	6942	22791	48259	84043	130333	187745	256614	337089	428965	532594



Reads and Writes per Sort per 100 items w/ total durations (copy/pasted output):

For a subvector of size 100 the # of reads w/ Bubble Sort is: 29680 and the # of writes is: 24837 DURATION: 1ms
For a subvector of size 200 the # of reads w/ Bubble Sort is: 148958 and the # of writes is: 124445 DURATION: 3.6ms
For a subvector of size 300 the # of reads w/ Bubble Sort is: 417786 and the # of writes is: 348792 DURATION: 8.6ms
For a subvector of size 400 the # of reads w/ Bubble Sort is: 896012 and the # of writes is: 747761 DURATION: 16.3ms
For a subvector of size 500 the # of reads w/ Bubble Sort is: 1643468 and the # of writes is: 1371229 DURATION: 24.7ms
For a subvector of size 600 the # of reads w/ Bubble Sort is: 2719950 and the # of writes is: 2269040 DURATION: 35.5ms
For a subvector of size 700 the # of reads w/ Bubble Sort is: 4185456 and the # of writes is: 3491192 DURATION: 47.8ms
For a subvector of size 800 the # of reads w/ Bubble Sort is: 6099656 and the # of writes is: 5087444 DURATION: 58.1ms
For a subvector of size 900 the # of reads w/ Bubble Sort is: 8522382 and the # of writes is: 7107652 DURATION: 70.4ms
For a subvector of size 1000 the # of reads w/ Bubble Sort is: 11513366 and the # of writes is: 9601597 DURATION: 86.1ms
For a subvector of size 100 the # of reads w/ Insertion Sort is: 20275 and the # of writes is: 10385 DURATION: 0.3ms
For a subvector of size 200 the # of reads w/ Insertion Sort is: 100750 and the # of writes is: 51120 DURATION: 1.3ms
For a subvector of size 300 the # of reads w/ Insertion Sort is: 281373 and the # of writes is: 142179 DURATION: 2.9ms
For a subvector of size 400 the # of reads w/ Insertion Sort is: 601996 and the # of writes is: 303488 DURATION: 5.2ms
For a subvector of size 500 the # of reads w/ Insertion Sort is: 1102447 and the # of writes is: 554961 DURATION: 7.2ms
For a subvector of size 600 the # of reads w/ Insertion Sort is: 1822526 and the # of writes is: 916498 DURATION: 10.7ms
For a subvector of size 700 the # of reads w/ Insertion Sort is: 2802233 and the # of writes is: 1408099 DURATION: 13.7ms
For a subvector of size 800 the # of reads w/ Insertion Sort is: 4081228 and the # of writes is: 2049594 DURATION: 19.3ms
For a subvector of size 900 the # of reads w/ Insertion Sort is: 5699379 and the # of writes is: 2860917 DURATION: 23.8ms
For a subvector of size 1000 the # of reads w/ Insertion Sort is: 7696490 and the # of writes is: 3861970 DURATION: 30ms
For a subvector of size 100 the # of reads w/ Quick Sort is: 17866 and the # of writes is: 12805 DURATION: 3.5ms
For a subvector of size 200 the # of reads w/ Quick Sort is: 88005 and the # of writes is: 62989 DURATION: 12.5ms
For a subvector of size 300 the # of reads w/ Quick Sort is: 243440 and the # of writes is: 174139 DURATION: 28ms
For a subvector of size 400 the # of reads w/ Quick Sort is: 515642 and the # of writes is: 368730 DURATION: 50ms
For a subvector of size 500 the # of reads w/ Quick Sort is: 934644 and the # of writes is: 668215 DURATION: 76.3ms
For a subvector of size 600 the # of reads w/ Quick Sort is: 1527158 and the # of writes is: 1091668 DURATION: 106.9ms
For a subvector of size 700 the # of reads w/ Quick Sort is: 2317203 and the # of writes is: 1656242 DURATION: 139.9ms
For a subvector of size 800 the # of reads w/ Quick Sort is: 3326348 and the # of writes is: 2377340 DURATION: 181.1ms
For a subvector of size 900 the # of reads w/ Quick Sort is: 4573151 and the # of writes is: 3268216 DURATION: 223.1ms
For a subvector of size 1000 the # of reads w/ Quick Sort is: 6075169 and the # of writes is: 4341409 DURATION: 270.4ms
For a subvector of size 100 the # of reads w/ Heap Sort is: 5066 and the # of writes is: 2771 DURATION: 0.1ms
For a subvector of size 200 the # of reads w/ Heap Sort is: 16976 and the # of writes is: 9164 DURATION: 0.4ms
For a subvector of size 300 the # of reads w/ Heap Sort is: 36494 and the # of writes is: 19531 DURATION: 0.6ms
For a subvector of size 400 the # of reads w/ Heap Sort is: 64122 and the # of writes is: 34114 DURATION: 0.8ms
For a subvector of size 500 the # of reads w/ Heap Sort is: 100035 and the # of writes is: 53010 DURATION: 1ms
For a subvector of size 600 the # of reads w/ Heap Sort is: 144532 and the # of writes is: 76356 DURATION: 1.2ms
For a subvector of size 700 the # of reads w/ Heap Sort is: 198024 and the # of writes is: 104333 DURATION: 1.4ms
For a subvector of size 800 the # of reads w/ Heap Sort is: 260469 and the # of writes is: 136923 DURATION: 1.6ms
For a subvector of size 900 the # of reads w/ Heap Sort is: 332264 and the # of writes is: 174325 DURATION: 1.8ms
For a subvector of size 1000 the # of reads w/ Heap Sort is: 413142 and the # of writes is: 216418 DURATION: 2.1ms
For a subvector of size 100 the # of reads w/ Two Sort is: 6942 and the # of writes is: 3657 DURATION: 0.9ms
For a subvector of size 200 the # of reads w/ Two Sort is: 22791 and the # of writes is: 11974 DURATION: 1.9ms
For a subvector of size 300 the # of reads w/ Two Sort is: 48259 and the # of writes is: 25311 DURATION: 3.5ms
For a subvector of size 400 the # of reads w/ Two Sort is: 84043 and the # of writes is: 43948 DURATION: 5.4ms
For a subvector of size 500 the # of reads w/ Two Sort is: 130333 and the # of writes is: 67885 DURATION: 5.7ms
For a subvector of size 600 the # of reads w/ Two Sort is: 187745 and the # of writes is: 97562 DURATION: 6.7ms
For a subvector of size 700 the # of reads w/ Two Sort is: 256614 and the # of writes is: 133039 DURATION: 8.4ms
For a subvector of size 800 the # of reads w/ Two Sort is: 337089 and the # of writes is: 174316 DURATION: 9.3ms
For a subvector of size 900 the # of reads w/ Two Sort is: 428965 and the # of writes is: 221393 DURATION: 11ms
For a subvector of size 1000 the # of reads w/ Two Sort is: 532594 and the # of writes is: 274270 DURATION: 12ms

Analysis:

As expected, Bubble Sort and Insertion Sort, which have a quadratic O(N^2) time complexity, have the longest durations. These two also have the highest increase in space complexity. This is justifiable given that these two sorting algorithms require more variable access and storage than the others.

Quick sort is an anomaly here. One would expect that a sort with the word "quick" in the name and that has an O(N logN) time complexity would be faster than bubble and insertion sort, however that is not the case here. After trying many different methods to fix this, including using the code from the other class section, changing CLion RAM settings/capacity, and double checking all of the code, I am still unsure as to what may have caused this issue.

One possibility that was suggested was that the recursive nature of quick sort may be causing this, however, I ended up having no problems with merge sort later on in the program so I'm positive that this is not the cause of the issue at hand. This also may somehow be a result of using the stable version of quick sort, though I suspect this is not the case. Nonetheless, quick sort, at the very least, ends up requiring fewer reads and writes than both bubble and insertion sort, so I'm sure that the end result of the sort is as intended.

Heap sort, which also has an O(N logN) time complexity, both takes a significantly smaller amount of reads and writes, needing only around 400,000 compared to insertion sort's 7 million. Heap sort also takes significantly less time to complete, taking only about 2ms compared to the multiple whole seconds that it takes to sort by the previously used algorithms. This is expected of heap sort, as it is a highly efficient sorting algorithm when compared to bubble and insertion sort.

Now for our two-sort. Similar to heap sort, it takes a comparatively smaller amount of reads and writes, and also takes less time to complete. The two-sort does require a slightly larger number of reads and writes than heap sort, though this is likely due to the fact there are two sorts in play here.

Ultimately, all of the sorting algorithms follow a relatively linear space/auxiliary complexity, with the number of variable accesses and overwrites increasing at a rate generally proportional to the number of items to be sorted in a vector. The algorithms perform as expected respective to their time complexities (save for Quick Sort).

If you need to sort a contacts list on a mobile app, which sorting algorithm(s) would you use and why?

To sort a contacts list, I would use bubble sort as it is a stable algorithm and also very slow. This way, I could force the user to pay for better performance and have things sorted by merge sort (which is also stable) for $0.99.

What about if you need to sort a database of 20 million client files that are stored in a datacenter in the cloud?

I would choose radix or sort, as it is one of the most efficient forms of sorting and it is also stable. This way, I can quickly sort 20 million files without having to pass through it a million times to find a certain item, and instead just sort them by digit as I am passing through the entire database.

References

Elliott. (1963, April 1). How do I pass multiple ints into a vector at once? Retrieved from https://stackoverflow.com/questions/14561941/how-do-i-pass-multiple-ints-into-a-vector-at-once

Greg Rogers. (1959, March 1). Best way to extract a subvector from a vector? Retrieved from https://stackoverflow.com/questions/421573/best-way-to-extract-a-subvector-from-a-vector.

Martin G. (1960, September 1). How to use clock() in C . Retrieved from https://stackoverflow.com/questions/3220477/how-to-use-clock-in-c.

Semyon Burov. (1966, January 1). Format double value in c . Retrieved from https://stackoverflow.com/questions/33125779/format-double-value-in-c.