Simulation of sorting techniques

Lingwei Meng, Yuxiao Guo

1. Description of each sorting algorithms

- Insertion sort

  Insertion sort is a simple and basic in-place sorting algorithm. It starts from the beginning of a dataset and loop over all positions to put every element in the right place. For each element, insertion sort compares its value with every element in the sorted part of the array and inserts that element at the right spot. The procedure will repeat till the end of the array. The time complexity for insertion sort is $O(n^2)$. This simply implemented algorithm is efficient for small dataset but not works very well when the data is getting larger in size.

- Selection sort

  Selection sort is another easily implemented sorting algorithm. Selection sort divides the input data into two sections: sorted part (left) and unsorted part (right). For each iteration, the algorithm goes through every element on the unsorted side and selects the smallest or largest item depending on sorting order. Then swap the element with the leftmost unsorted element. This procedure will be repeat will the end. Similar to insertion sort, this sorting algorithm could be used on small dataset and dataset with low sortedness (close to sorted) but not efficient enough for large dataset. Time complexity for selection sort is $O(n^2)$.

- Bubble sort

  Bubble sort starts from the beginning of the list to compare every adjacent pair of elements. The two elements will be swapped if they are not in right order. As consequence, every iteration will move one largest/smallest element to the end of unsorted array. The process will be repeated till the whole data is sorted. This algorithm has both worst-case and average time complexity as $O(n^2)$. This algorithm is mostly not being used in practice even though it is also easily to apply. Because similar sorting algorithm such as insertion sort has better average time complexity. One advantage of this algorithm is that it could identify sorted data and performs well on almost sorted data.

- Merge sort

  Merge sort is a comparison-based sorting algorithm that focuses on merging two sorted arrays into one. It uses divide-and-conquer to divide the array into n subarrays with only one element in each. Then repeatedly merge two subarrays into one sorted subarray till the entire array is in right order. Merge is one of the most widely used algorithm because of its high efficiency. The time complexity of merge sort is $O(n*\log(n))$. One thing about merge sort is, the most common implementation of merge sort doesn't sort in place. So, the memory size must be allocated to store the output.

- Quick sort

  Similar to merge sort, quick sort is also a divide-and-conquer sorting algorithm. The difference is quick sort algorithm sorts the array in-place, requiring small amount of memory to perform the sorting. Quick is a very efficient and widely used sorting

algorithm. The first step of quick sort is to pick an element(pivot) from the array. Then, put all bigger elements on its right side and all smaller elements on its left side, which is also known as partitioning. After this step, the pivot should be in its final position. Lastly, recursively repeat these steps to two subarrays of pivot. To sort an array with n elements, the average time complexity for quick sort is O(n*log(n)), but the worst-case time complexity is $O(n^2)$. The strategy of choosing pivot could largely affect its efficiency. Current method for choosing pivot is called median-of-three, which use the median of the first, middle and last element of the partition.

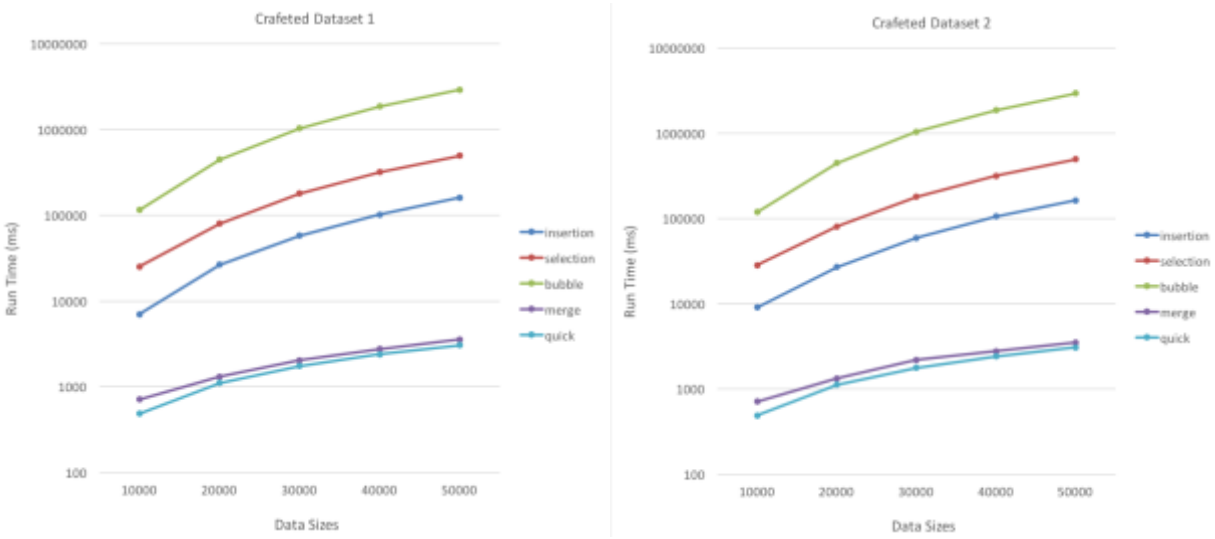
2. Description of data sets

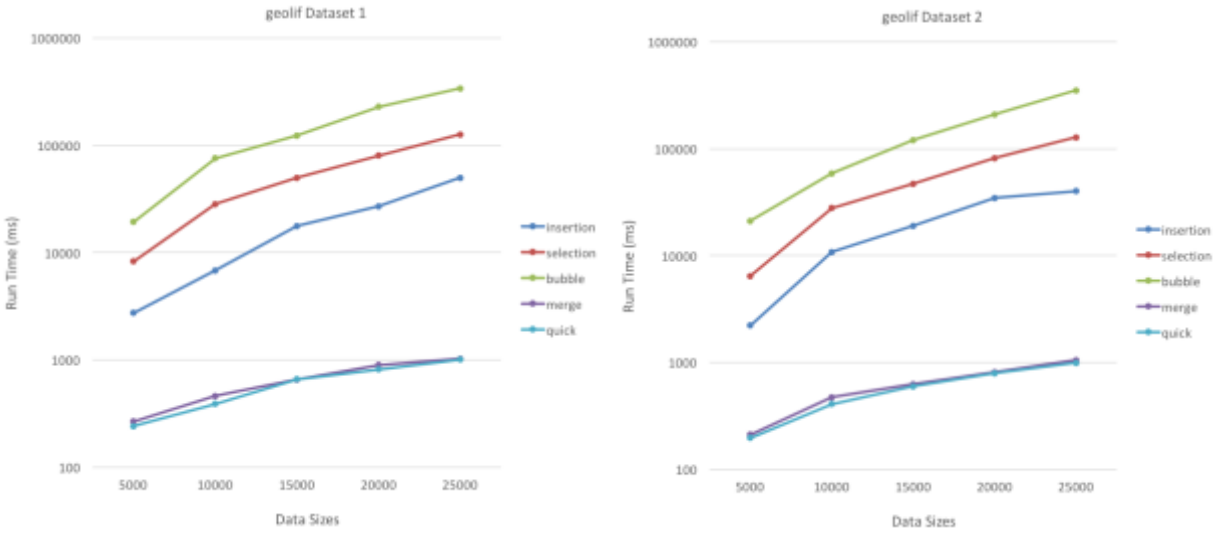
We generated two datasets with *uniform distribution*. For each dataset, we have five data sizes from 10,000 to 50,000.

We download Geolife GPS trajectories data and make processing of the row data. We extract two datasets with five data sizes from 5,000 to 25,000 for each dataset.

3. Performance curves

- Time-datasize





Standard Deviation

Crafted data1

size	insertion	selection	bubble	merge	quick
10000	687.3873435	2741.291783	3192.585347	4.445222154	2.8
20000	762.5056065	12680.40194	36123.92491	7.6	2.683281573
30000	1313.572076	27780.69977	87623.73555	6.794115101	6.368673331
40000	2918.362239	49166.26247	162664.0086	8.133879763	2.638181192
50000	5294.522959	78063.11093	274863.9337	30.56795708	6.4

Crafted data2

size	insertion	selection	bubble	merge	quick
10000	4307.676655	5077.965514	4187.360534	5.571355311	6.280127387
20000	2113.596404	12580.44059	32926.48125	7.364781056	125.2239594
30000	894.5335321	28049.97295	80967.58059	6.85857128	3.16227766
40000	2157.992771	50191.62226	167192.9309	7.626270386	1.624807681
50000	2409.808905	78119.73242	280318.4014	13.37161172	3.826225294

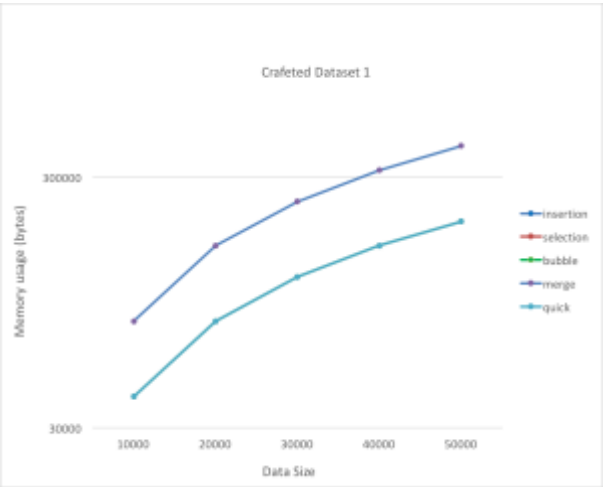
Geolif data1

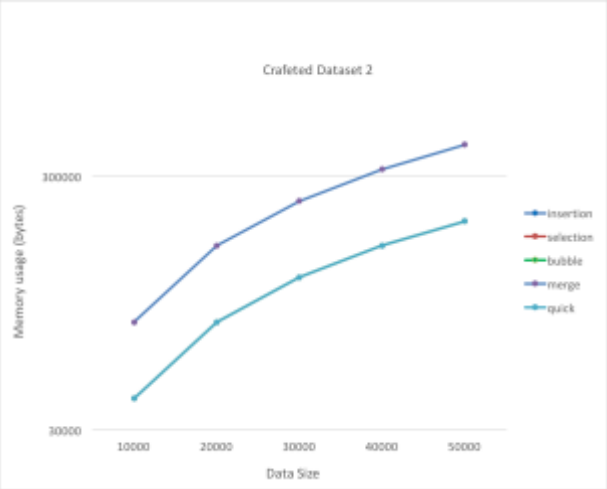
size	insertion	selection	bubble	merge	quick
5000	1904.994131	1648.45302	3235.814853	53.05091894	34.08577416
10000	1812.153194	1700.255698	7691.995179	11.14629983	4.915282291
15000	2097.128475	16180.52925	30018.44386	4.92341345	74.62439279
20000	6258.704614	12685.67181	45761.52176	5.966573556	3.709447398
25000	1291.304612	20600.41102	80804.85917	4.955804677	6.76461381

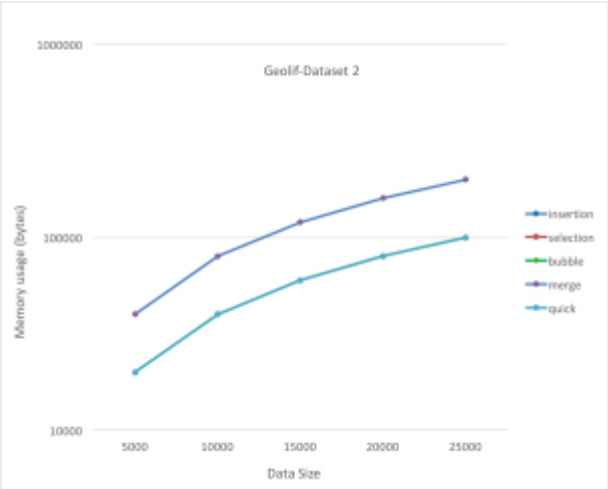
Geolif data2

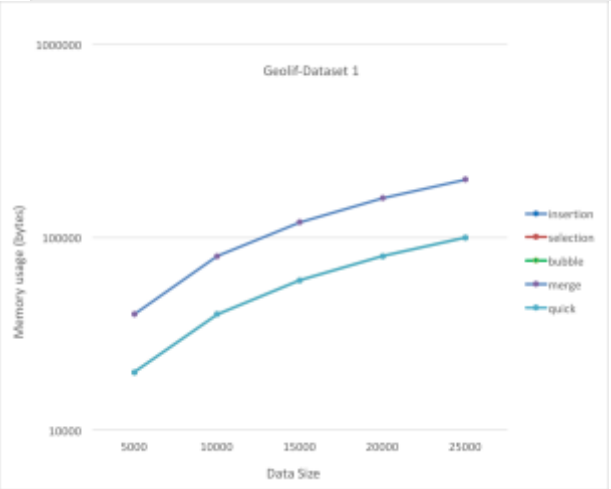
size	insertion	selection	bubble	merge	quick
5000	13.65869686	749.6358049	1254.895693	4.176122604	1.854723699
10000	2589.533348	4821.63072	15156.05953	21.37849387	17.32512626
15000	2683.29256	10094.78121	29532.51126	3.826225294	9.350935782
20000	1935.602583	12628.82798	52950.31013	4.758150901	6.144916598
25000	2024.816298	19759.96686	71050.44183	6.431174076	6.462197769

- Memory-datasize *(the curves of all four sort algorithms besides merge sort are all similar(the lower curve))*

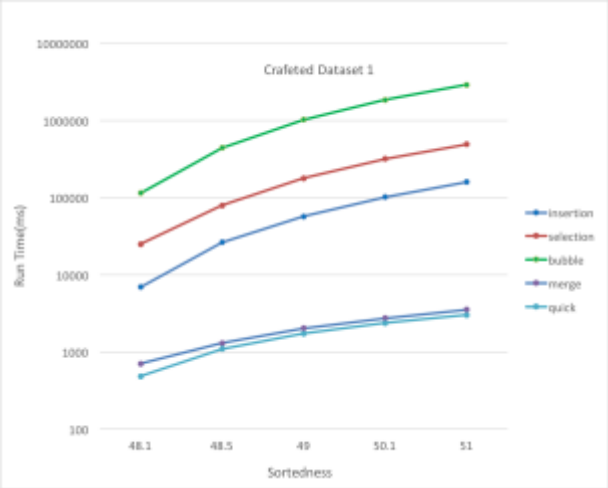


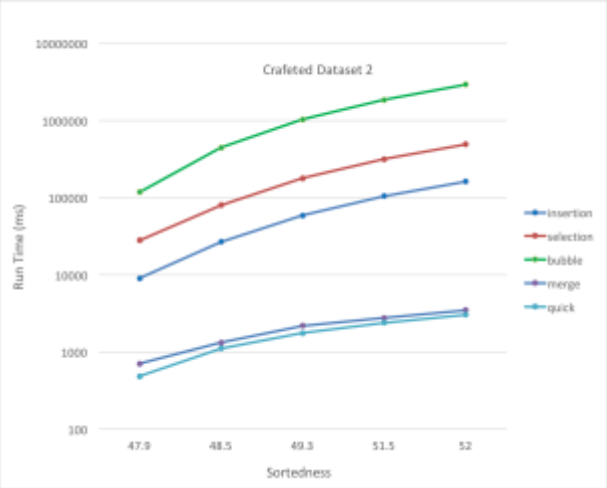


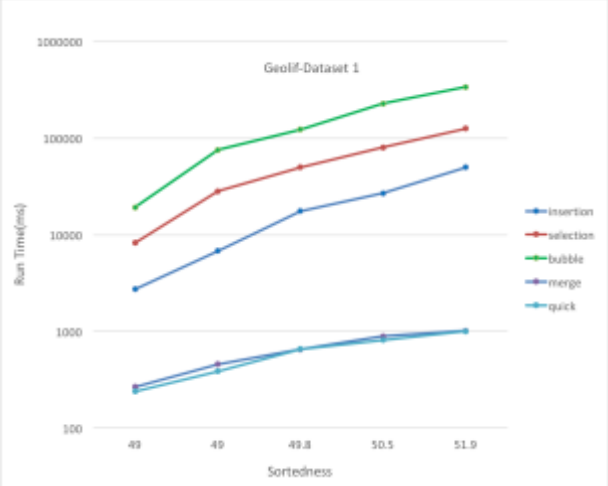


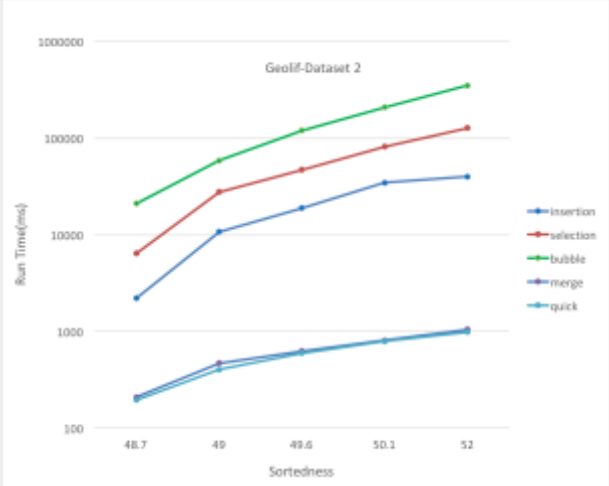


- Time-sortedness

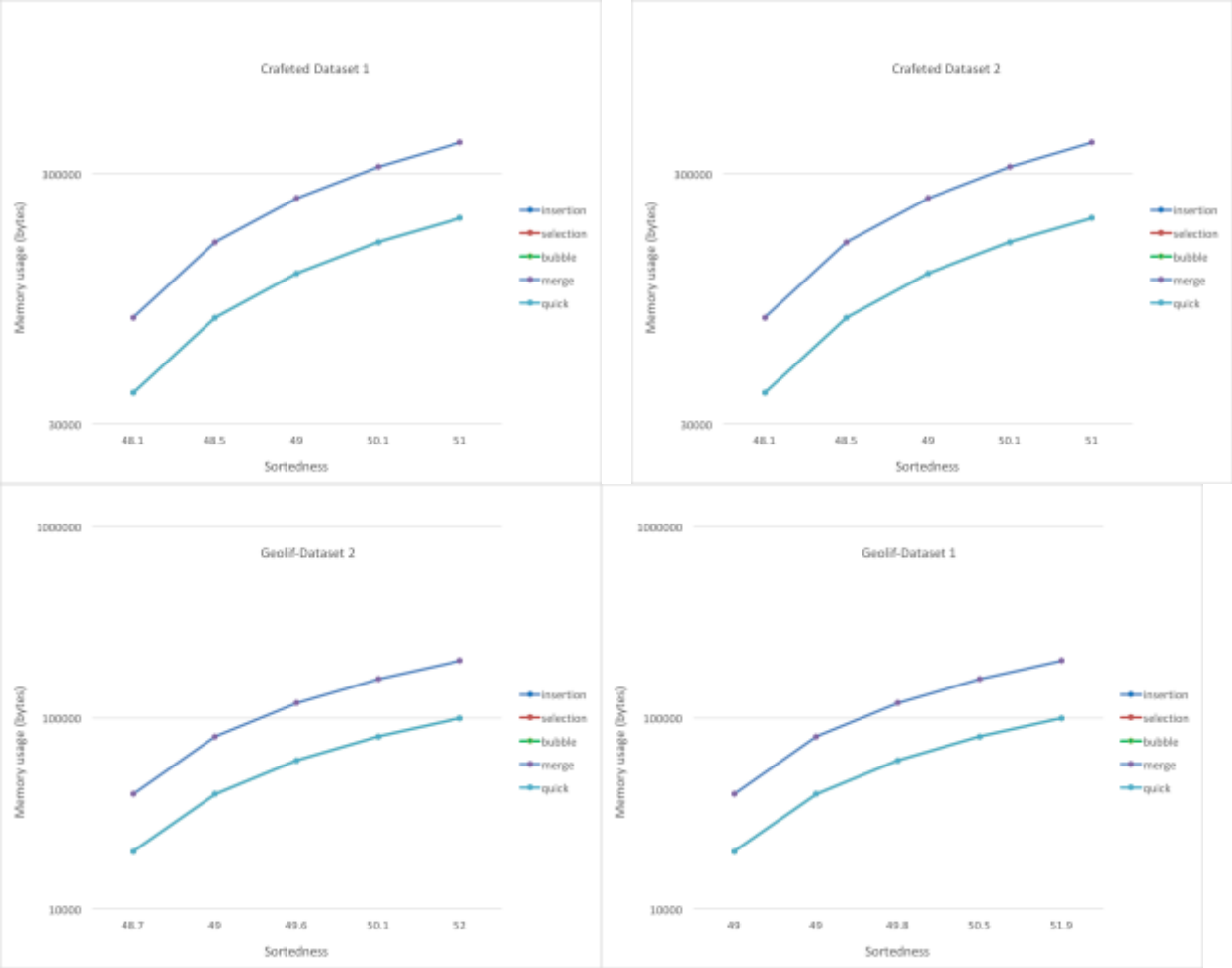








- Memory-sortedness *(the curves of all four sort algorithms besides merge sort are all similar(the lower curve))*



4. Discussions

From curve1-4, we can see that the running time increases with size increasing. Merge and quick sort have similar performance. They are the two fastest sort algorithms for data sizes from 5,000 to 50,000. The other three sort algorithms are not so good at sorting large set of data. They are significantly slower than merge and quick sort. Among them, the ranking is insertion, selection, bubble. This is consistent with our hypothesis. One thing that we might want to do is to try sets of smaller data (for example, from 5 to 100) on them. Because as the literature mentioned, for real-life applications, some advance sorting algorithm (like quick sort) will be transferred to simpler sorting algorithms (like insertion sort). Hence, we think the lower ranked three algorithms might perform better and similar to the two advanced algorithms if we use very small input to test.

The more data to be sorted, the more memory storage is required. The most of memory are used to store the array that need to be sorted. For the in-place sorts, including insertion, selection, bubble and quick sort, they do not need much more additional memory space. Hence, their space complexity is O(n + 1) or O(n + log(n)). But merge sort is an out-place sort, requiring a lot of additional memory to temporally store the result and to operate the sorting process. So, the memory usage of merge sort is distinctly higher than other sort algorithms (O(2n + log(n)).

The complexity of input data (degree of sortedness) could also affect the running time of a sorting algorithm. However, the data size is still the key factor that determine algorithm's performance and how long it would take for the array to be sorted. We figured that it would be very hard to analyze the data if we have two variables in one set of data. To be specific, for one set of data, if both length and sortedness are different crossing each group, we couldn't tell whether it is the length or the sortedness of that group that lead to better or worse running time. So, we decided to use data size as our main variable to perform this experiment.

But to satisfy the goal of this part, we slightly changed the sortedness for each group using shuffle function. As shown above, all algorithms need longer running time and to sort high complexed input. However, it wouldn't affect the memory usage very much since it mostly determined by the data size and algorithm type.

5. Conclusions

It is really hard to pick the best sort algorithm because each of them have the beauty of their own. Like it is really simple to implement the insertion and selection sort algorithm. Merge sort and quick sort are both so efficient to use of big data sets.

Based on our experiment and knowledge, quick sort has best performance on sorting data. When tasks require a limited time to be finished, we prefer quick sort.

Merge sort can finish tasks very quickly but requires a high memory usage cost. If user do not need to worry about memory storage, merge sort is also a good choice.

For very small dataset and simple experiment, we would recommend simple sorting techniques like insertion sort and selection sort. This is because they are both simple to code and have good performance.