**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(n2). 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(n2).

* 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(n2). 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(n2). 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.

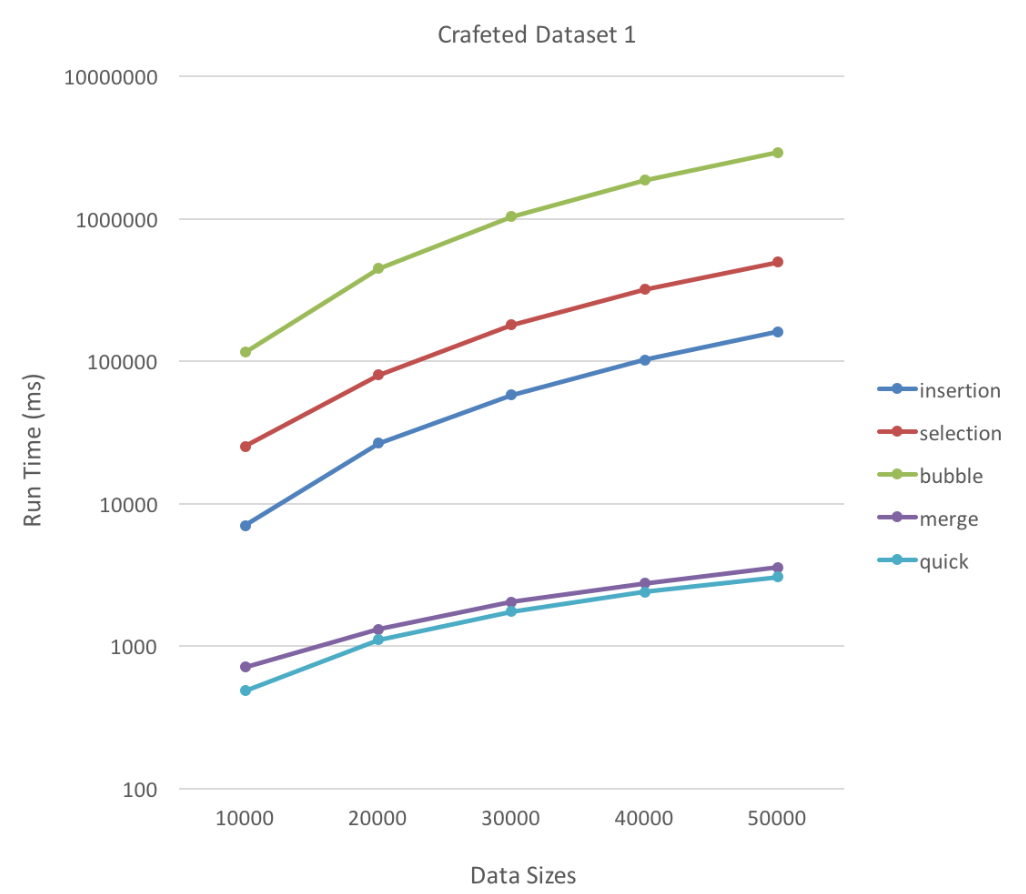
1. Description of data sets

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

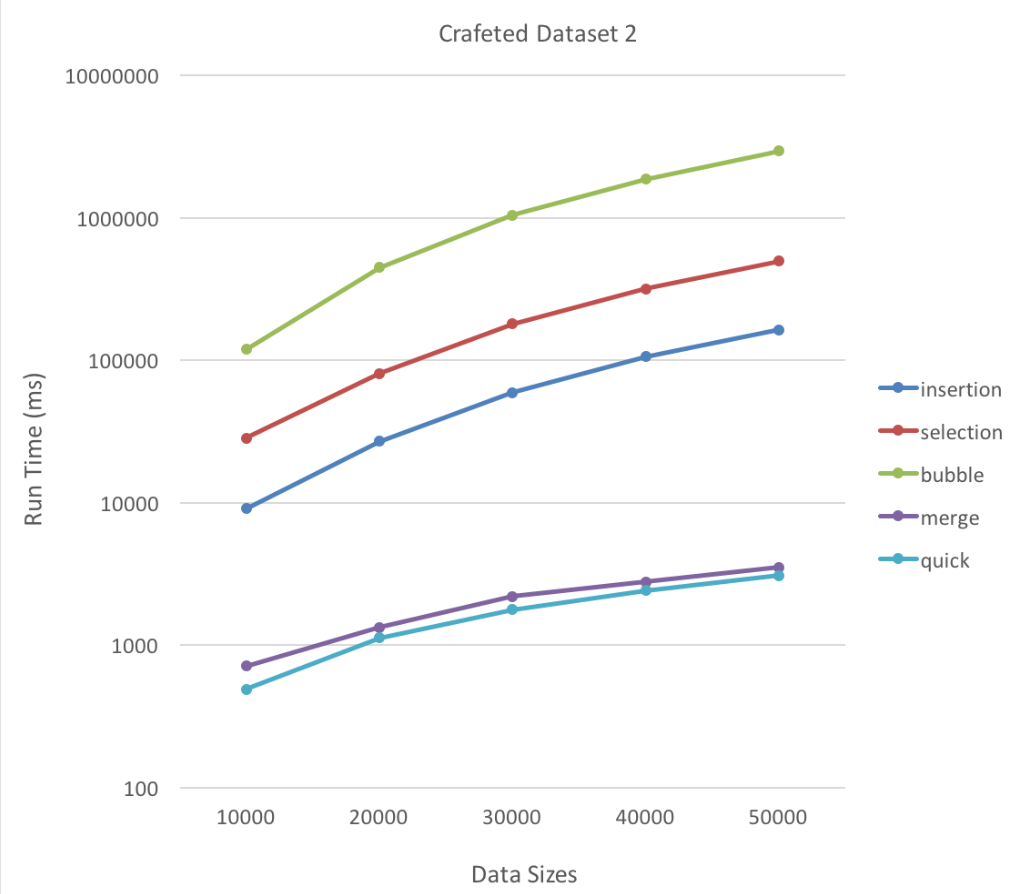
We download Geolife GPS trajectories data and make processing of the row data. We extract two datasets with five data sizes from 5,000 to 25,000 for each dataset.

1. Performance curves

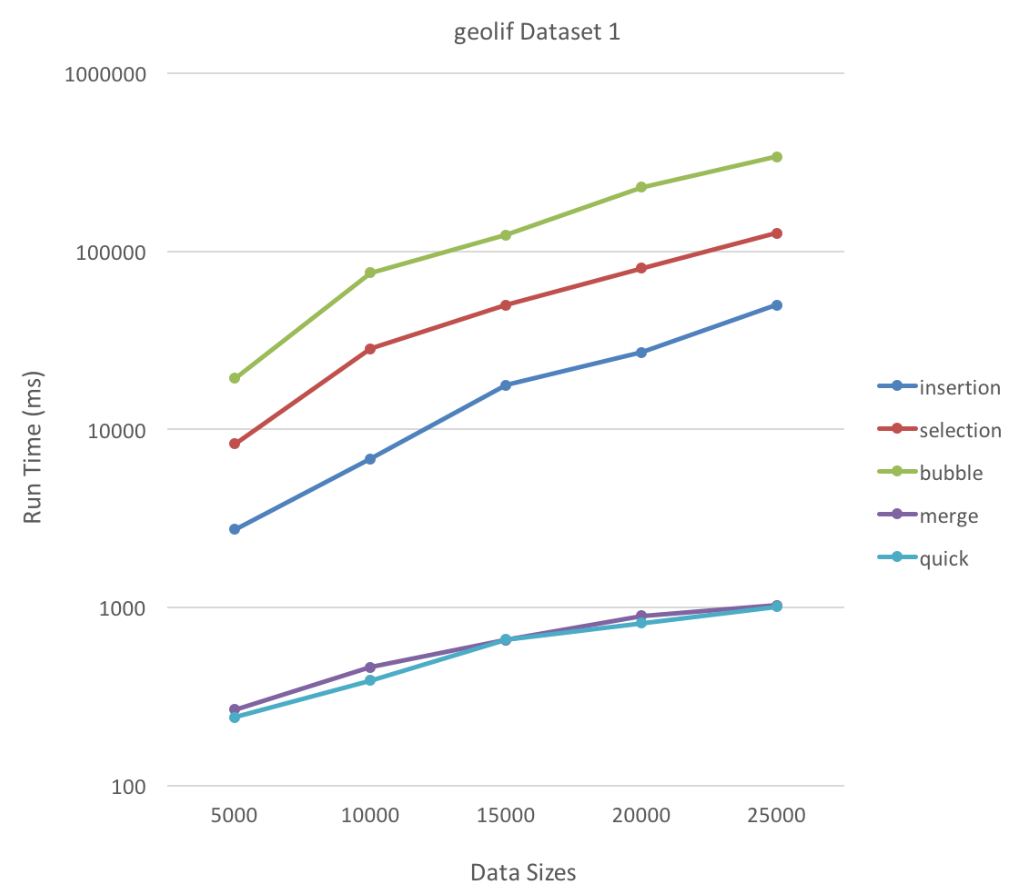
Time-datasize



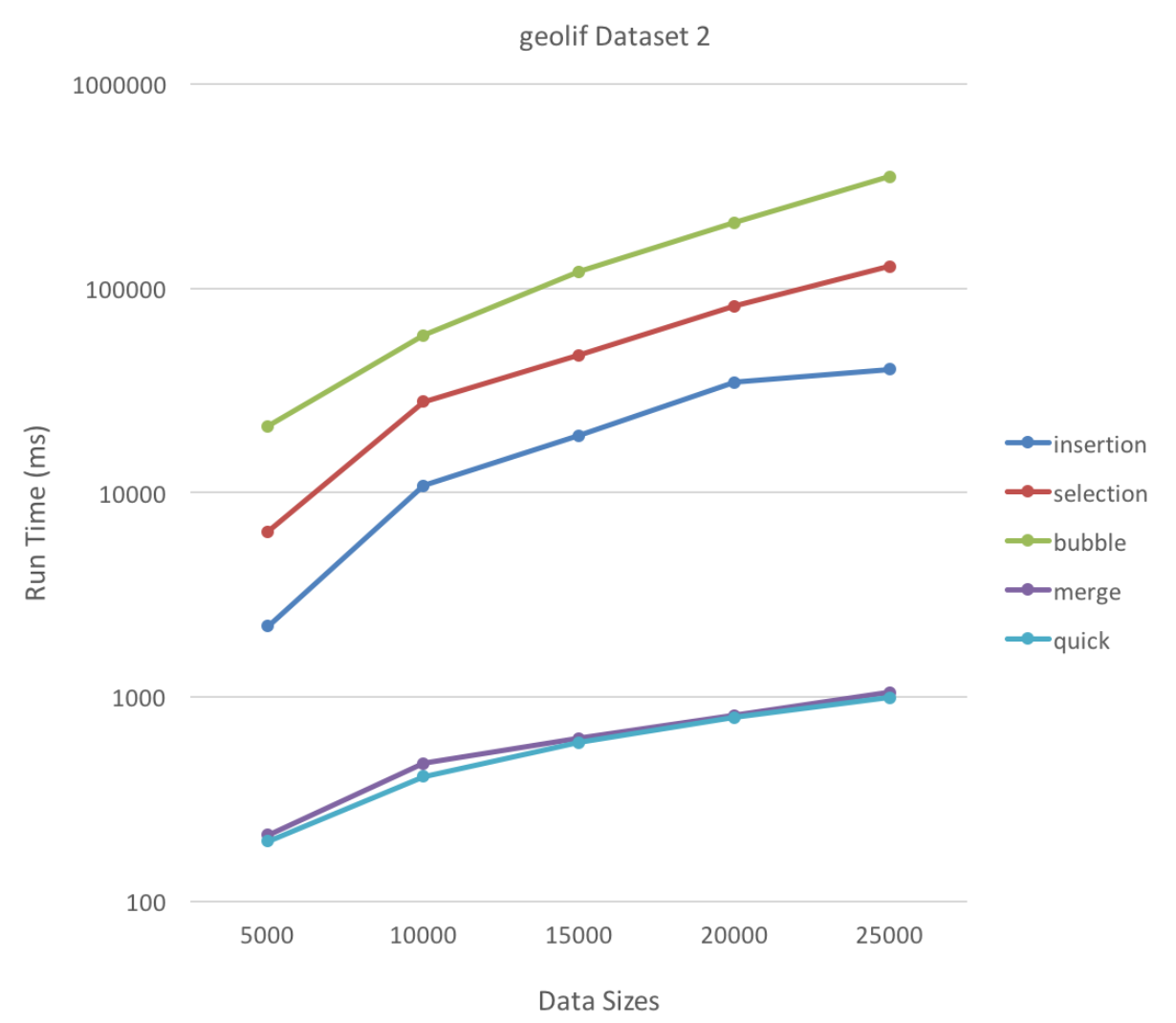
Curve1



Curve2



Curve3



Curve4

Memory-datasize

Time-sortedness

Memory-sortedness

1. Discussions

From curve1-4, we can see that the time increases with size increasing. Merge and quick sort are close and they two fastest sort algorithms at size from 5,000 to 50,000. The other three sort algorithms are slower and their rank is insertion, selection, bubble.

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. But merge sort is a out-place sort, and it requires a lot of additional memory to operate the sorting. So the memory usage of merge sort is distinctly higher than other sort algorithms.

Time-Sortedness

Memory-Sortedness

1. Conclusions

Quick sort has the fastest speed to sort data. When tasks require a limited time to be finished, I prefer quick sort.

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