Experimental Design:
I measured how the execution time of each program varied with input size and initial sorting. I used the following input types: random, sorted, and reversed.
I chose these test cases because analyzing the performance of each sorting algorithm on these three input types helps identify the algorithm in question.
Due to timing mechanisms on Linux, the same test needed to be repeated multiple times. For each algorithm and input type, I ran five tests, with input sizes of 10,000, 20,000, 40,000, 80,000, and 160,000 elements to ensure robustness.
I also tested the stability of the algorithms using datasets containing repeated numbers and different strings.

Experimental Results:
For sortA, I observed the following:
Based on stability test's outcome, I observed sortA is unstable. Based on sortA's performance on inputs ranging from 10,000 to 160,000 elements, sortA performs poorly on all three types, with a time complexity of O(n^2). Based on time complexity, bubble sort, selection sort, and insertion sort can all have a time complexity of up to O(n^2) in some cases. However, the time complexity of bubble sort and insertion sort for sorted data should be O(n). Therefore, only selection sort meets these time complexity characteristics. Combined with sortA's instability, bubble sort and insertion sort are ruled out. Therefore, my guess for sortA is selection sort.

For sortB, I observe the following:
Based on stability test's outcome, I had not observed sortB is unstable so I guess sortB is stable. sortB performs very well on all three inputs, with a time complexity of O(nlogn). Sorting algorithms that meet this time complexity characteristic include Merge Sort and Heap Sort. Heap Sort is unstable and does not meet sortB's stability characteristics, so I speculate that sortB uses Merge Sort.

Appendix contains experimental data.