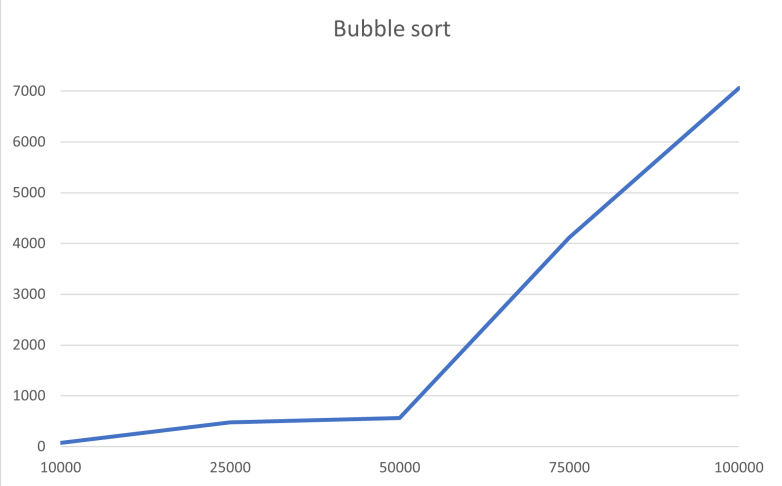
Natalie Belmonte
12-12-23
Data Structures and Algorithms: Sorting Algorithm Analysis

Throughout programming languages, there are many different algorithms developed to find the most efficient method of sorting a group of objects. They can be as straightforward as sorting ten integers using nested for loops, or as complex as sorting hundreds of thousands of objects within a table using a binary tree. Sorting algorithms vary in complexity, structure, and efficiency. Using a Java program that records the runtimes of various algorithms with certain amounts of integer inputs, seven of the most common sorting algorithms can be analyzed for runtime efficiency.

A. Bubble sort



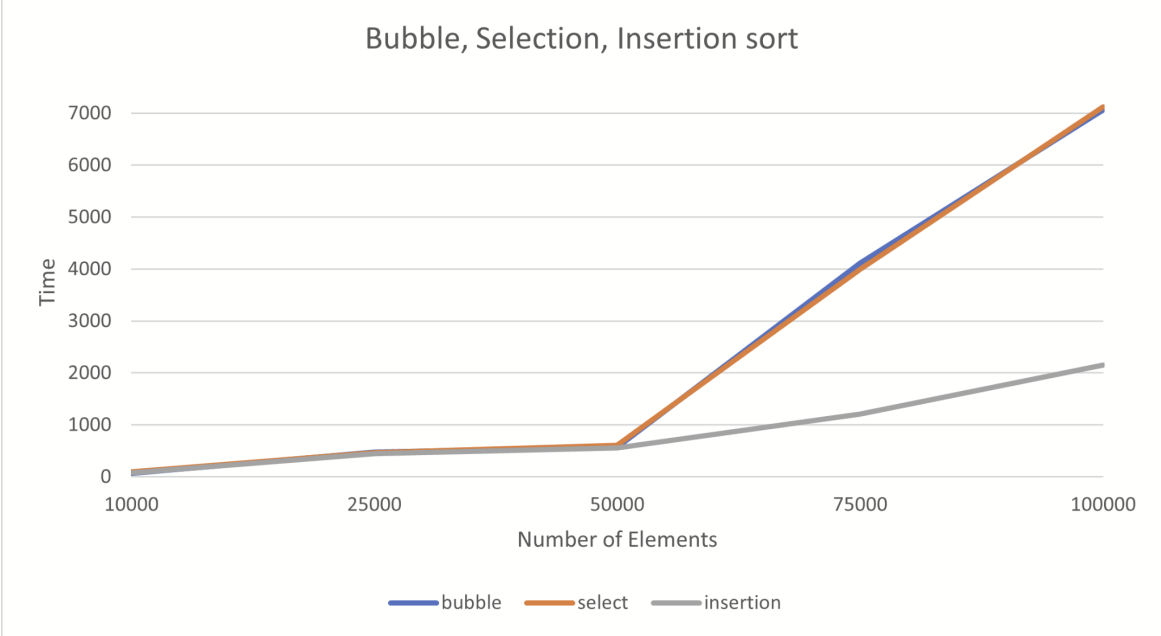
Bubble sort is widely regarded as one of the simplest and least time-efficient sorting algorithms. It consists of a nested for loop, which increases runtime exponentially as the field size increases. As shown in this graph, bubble sort maintains a relatively low runtime until 75,000 integers is reached, where the time then skyrockets. Because of this increasing runtime, using bubble sort for larger field sizes is inefficient at best and impossible at worst.

B. Selection sort

Selection sort has a very similar runtime graph when compared to bubble sort. Though it, too, uses a nested for loop, it first selects the first value of the data field and sets it to the lowest value. If a lower value is found, it replaces the previous one in the field. This slight change decreases the overall runtime and curbs the extreme increase seen in bubble sort.

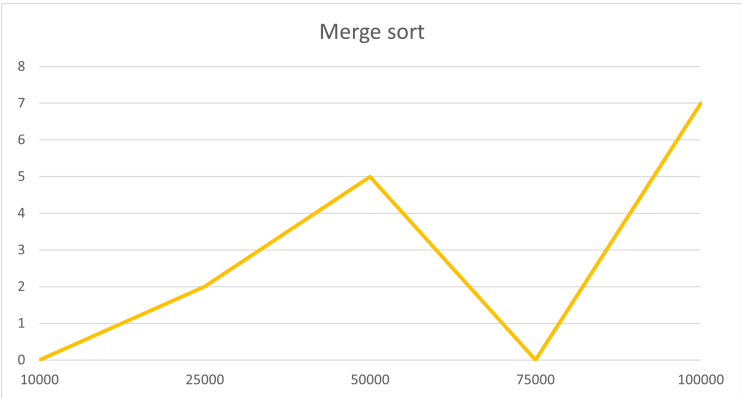
C. Insertion sort

Insertion has drastically lower runtimes than seen in the previous algorithms. While it utilizes for loops as well, it sorts as it traverses through the data field, which increases its efficiency. Though it is the most efficient of the three aforementioned algorithms, it still has a runtime spike as the field size increases.



At lower sizes, these three algorithms have near identical runtimes. While bubble and selection sorts have near identical efficiencies, insertion sort is comparatively much more efficient at higher sizes.

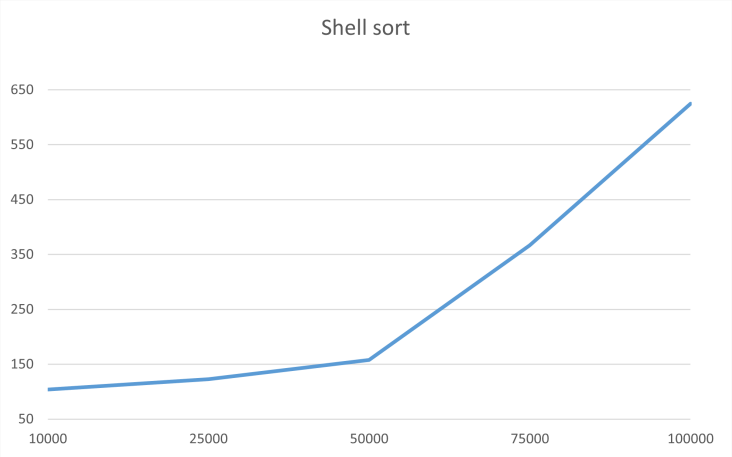
D. Merge sort



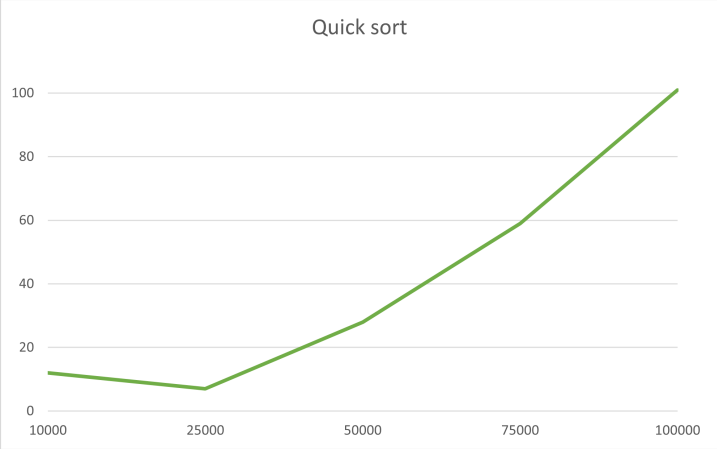
Merge sort is immediately differentiable from the previous three algorithms. By splitting the field size into parts, then putting those parts back together in the correct order, this sort is exceptionally more efficient than those that use for loops. At smaller field sizes, its runtime is negligible, which accounts for the variation shown in this graph.

E. Shell sort

Though not as efficient as merge, shell sort has similarly low run times. There is a spike in runtime as the field size increases, though it is not as sharp as it is for other algorithms. Shell sort follows a similar process to insertion sort in a more efficient way. By grouping the data into intervals, sorting each individually, and then sorting them together, shell sort decreases its runtime by preventing the entire data set needing to be shifted at one time.



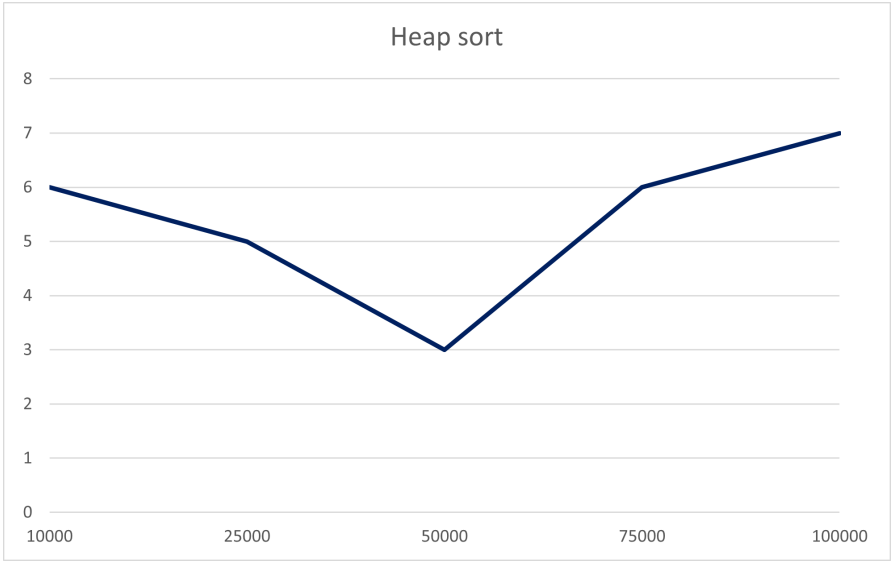
F. Quick sort

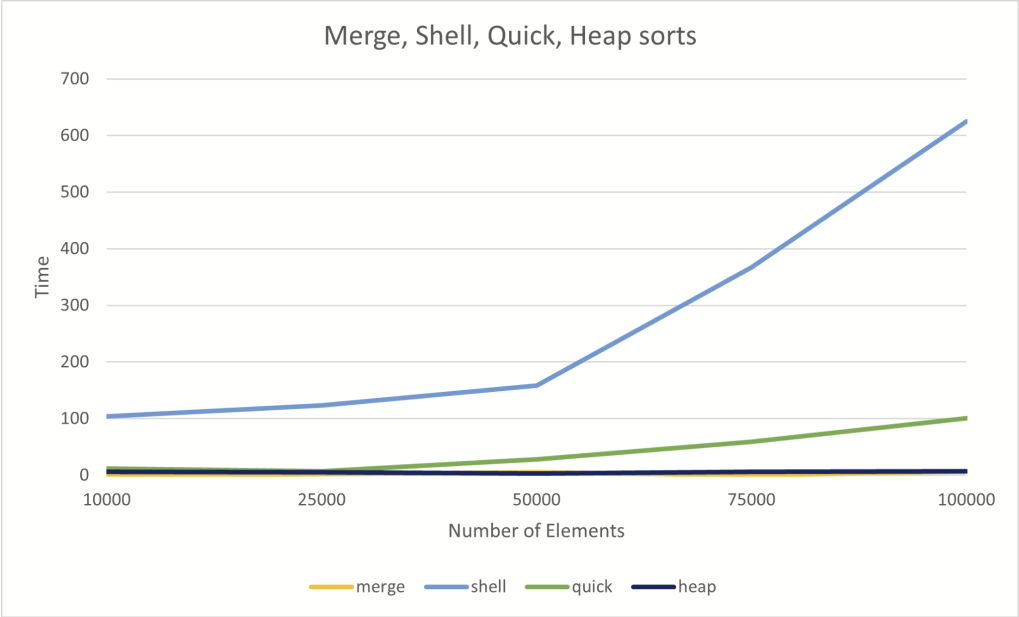


As can be assumed by its name, quick sort has low overall run times. It selects an element to use as a "pivot", splits the array by moving the pivot to its correct location, then selects a new pivot and continues. Similarly to merge sort, quick sort utilizes recursion to decrease both runtime and code complexity (in terms of number of lines).

G. Heap sort

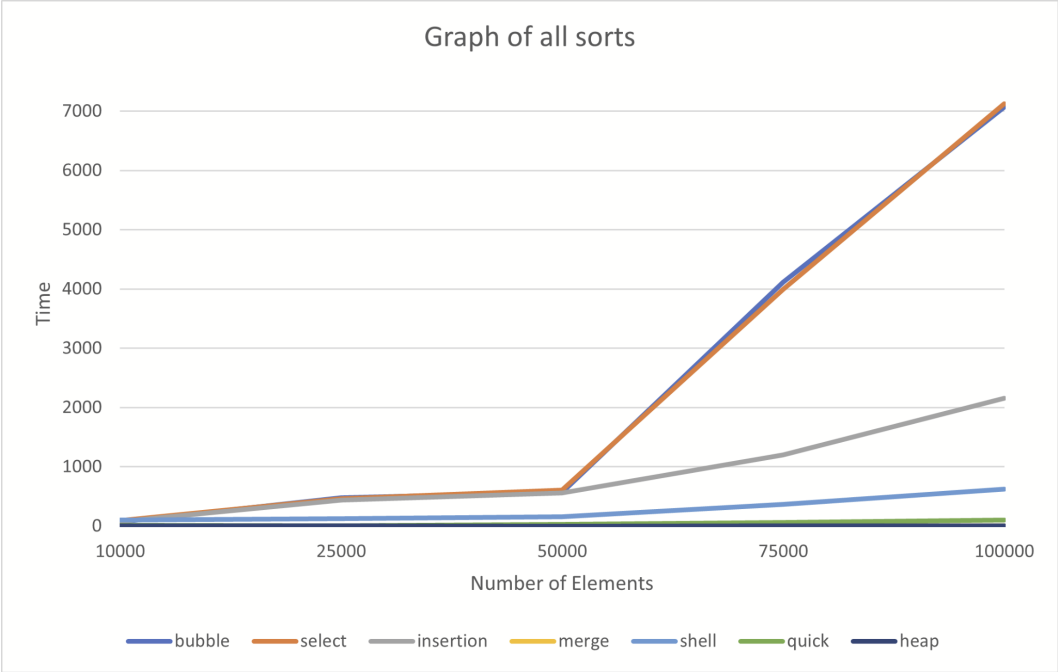
Similarly to merge sort, heap sort's low runtime makes it difficult to accurately measure runtime at lower field sizes. This results in interesting graph behavior, such as the dip shown at 50,000. Heap sort uses the structure of a binary tree to split, traverse, and sort the data set.





As shown above, merge, quick, and heapsort are comparable in runtime at lower field sizes. Shell sort has a slightly higher overall runtime.

H. Conclusion



It is difficult to observe the behavior of these seven algorithms in detail when they are graphed on the same axes, because of their varying runtimes. However, this visualization is still useful for observing which algorithms are generally most and least efficient. When selecting a sorting algorithm, runtime is one of the most essential factors. With larger data fields, an efficient sorting algorithm can exponentially decrease processing time and prevent stack overflow errors. Visualizations such as these graphs are a simple way to determine runtime efficiency with the naked eye.

Works Cited

A. Programming references

"Data Structure and Algorithms - Shell Sort." *Online Tutorials, Courses, and eBooks Library*, www.tutorialspoint.com/data_structures_algorithms/shell_sort_algorithm.htm. Accessed 12 Dec. 2023.

Koffman, Elliot B., and Wolfgang Paul A T. *Data Structures: Abstration and Design Using Java*. John Wiley, 2016.

B. Research references

"Data Structure and Algorithms - Shell Sort." *Online Tutorials, Courses, and eBooks Library*, www.tutorialspoint.com/data_structures_algorithms/shell_sort_algorithm.htm. Accessed 12 Dec. 2023.

"Quicksort - Data Structure and Algorithm Tutorials." *GeeksforGeeks*, GeeksforGeeks, 16 Oct. 2023, www.geeksforgeeks.org/quick-sort/.