Fun with Sorting

I am coming into this study with only one goal: to learn more about sorting algorithms. I suppose it could be said that my hypothesis is simply that different sorting algorithms will perform differently. My only intention is to earnestly compare and contrast some of the more well-known sorting algorithms. The algorithms which I will be analyzing are Bubble Sort, Insertion Sort, Selection Sort, Shell Sort, Merge Sort, and Quick Sort.

Bubble Sort is commonly among the first sorting algorithms one learns. It is more commonly used as a learning tool than a practical sorting algorithm. It is called Bubble Sort because it works by carrying the most appropriate data as far as it can go in the list on each pass, much like a bubbles rising to the top of a bath. Bubble Sort is extremely simple, hence its status as a learning tool. However, Bubble Sort is notoriously slow both theoretically and practically. Bubble Sort’s theoretical performance is O(n^2) in the average and worst cases, and O(n) in the best case.

Insertion Sort is a sinking sort. It works in much the opposite way of bubble sort in that it sinks appropriate data towards the beginning of the list. Insertion Sort is also relatively easy to implement. Insertion Sort is theoretically equivalent to Bubble Sort, it too has theoretical performance of O(n^2) in the average and worst cases, and O(n) in the best case. However, practically Insertion Sort is much faster than Bubble Sort. Insertion Sort is therefore rather useful for sorting relatively small lists.

Selection Sort is another simple sorting algorithm. Selection Sort builds the sorted list from the beginning to the end like Insertion Sort (and unlike Bubble Sort), but Selection Sort works but only making one swap for each run through the list. Selection Sort is also easy to implement. Selection Sort is theoretically equivalent to Insertion Sort and Bubble Sort, with a theoretical performance of O(n^2) in the average and worst cases, and O(n) in the best case. Selection Sort tends to be quicker than Bubble Sort but slower than Insertion Sort in practice, so it is not commonly used in practical applications.

Shell Sort is slightly more complex than the previous sorting algorithms, at least implementation-wise. Shell sort works essentially by sorting a list very approximately at first, then less and less approximately each pass through of the list until it depreciates into Bubble Sort. Shell Sort's theoretical performance is heavily influenced by how it is implemented, but it tends to be faster than the previous four sorting algorithms and slower than Quick Sort and Merge Sort. Shell Sort also tends to be practically quicker than the previous four sorting algorithms, while remaining relatively simple to implement, so it does see some practical application.

Merge Sort is a divide and conquer sorting algorithm. Merge Sort works by repeatedly breaking down a list into sub-lists, then building them all back up into a sorted list. Because of this, Merge Sort tends to be very memory intensive, but very fast. Merge Sort’s theoretical performance is O(nlogn) in all cases. Practically, Merge Sort sees a lot of usage. Merge Sort tends to be faster than Quick Sort when sorting slow-to-access data, such as data on a hard drive or in a linked list, but slower than Quick Sort when it comes to data which is fast to access, such as an array.

Quick Sort is another divide and conquer sorting algorithm. Quick Sort works by selecting a point in the list, then swapping elements on either side of this point as long as they would be going to the side of the list which is appropriate to them relative to the selected point. Quick Sort’s theoretical performance is O(nlogn) in the average and best cases, and O(n^2) in the worst case, though this worst case is extremely rare. Practically, Quick Sort is very efficient. Quick Sort uses much less memory than Merge Sort and tends to be faster when it comes to fast-to-access memory.