LSD Radix Sort: The Best Approach

In building a spreadsheet application, sorting data is a fundamental feature. It helps us to arrange data in a way that's intelligible. Yet, how does computer data sort itself?

Alas, this is the purpose of computer algorithms. An algorithm can be defined as a set of clear sequential directions as a resolution to a given problem. Two of the most prevalent/fastest sorting algorithms are merge sort and quicksort.

However, the best way to go about sorting a set of data is to use another algorithm known as Least Significant Digit (LSD) Radix sort. This is the sorting algorithm that my spreadsheet application will implement. Here's why:

Typical spreadsheet data will involve sequences, letters or ID numbers which are repetitive and have similar lengths, most of the time. It will be faster than both quick sort and merge sort, on average, no matter how well the other sorting methods do it.

Concurrently, let's assume you have time-sensitive data; data that needs to be handled soon or records entered into your spreadsheet from oldest to most recent (top to bottom). No one would want to lose the time-sensitive properties when sorting. LSD Radix sort guarantees during sorting that data will be preserved in their original relative order unlike quicksort.

Of importance, what if you are low on computer space and need to sort? LSD Radix sort will use less space always than merge sort and can do so as well in comparison to quicksort. The memory differences may not be staggering but if you're very low on computer space then every "bit" counts!

Technical Explanations

On a more technical note, LSD radix sort is a non-comparison-based sorting algorithm which runs in $O(n*w)$ time where n is the number of elements in the given data set and w is the number of characters that comprise a given data point. Customary implementations of this algorithm enforce that all data points in a set have an equivalent number of characters. This may not be necessary if the given data type can be modifiable so that all entries have an equivalent character length (such as integers). This $O(n*w)$ can outperform any comparison-based sorting algorithm when $w < \log(n)$. We assume that in the average-case $w < \log(n)$ and so this even outperforms quicksort and merge sort's $O(n*\log(n))$ best case.

Although merge sort has an average, worst, and best case of $O(n*\log(n))$, quicksort has a worst case $O(n^2)$ assuming that the pivot-selection step chooses either the largest or smallest number in a data set to be the pivot. In this case, LSD radix sort drastically outperforms quicksort.

Essentially, LSD radix sort's best case are data points with low numbers of characters and ideally of the same length. Spreadsheet data could include social security numbers, work ID numbers, etc. which are cases matching the previous description.

Of utmost importance is that LSD radix sort maintains the relative order of equal values from a data set. This property is known as being stable. Stability is an important concept in sorting algorithms which can be illustrated by the following example. Let's say that you have a spreadsheet of data where no sorting has been enforced yet but inherently the data is entered and maintained from earliest record to most recent record, since the spreadsheet is appended in a top-down fashion.

Furthermore, you would like to sort the data based on other parameters whilst still maintaining the time-entered order the original data set has. This would be to see between equal keys which data was entered before vs. after. Other sorting algorithms such as quick sort would not guarantee this to be the case. Due to the need to sort each character or digit one at a time and still give a correct sorted output, LSD radix sort maintains the relative order of the original data set so as not to mess up later iterations of the algorithm.

Finally, LSD radix sort can use less memory than both merge sort and quick sort in certain cases. Unlike quicksort and merge sort, LSD radix sort does not use recursion. The overheads of recursion are a downside to the previous algorithms. They require pushing, popping, and resizing of the stack which may take up much more memory as opposed to the auxiliary arrays that radix sort uses.

In the case of merge sort, this algorithm uses an auxiliary array as well as multiple recursive calls causing lots of memory overhead being used just for the sake of sorting. It may guarantee stability but why go for an algorithm with such memory requirements when we have radix sort?

Similarly, quicksort may not use O(n) memory for an auxiliary array, but there may be issues with the partitioning step. If this step is done incorrectly and the worst case is entered, the algorithm will make O(n) recursive calls to the stack. With LSD radix sort, we can still maintain stability (unlike quicksort) and use at most O(n) memory that does not need to be resized.

To conclude, for the sake of the aforementioned reasons, LSD radix sort is the best sorting algorithm and the one I will implement in my spreadsheet application. It is faster, uses less memory, and maintains stability.