**Radix Sort**

Sorting is a relatively simple task for humans to accomplish. We learn from a young age how to take a group of numbers and sort them in numerical order. However, if given the task to sort hundreds of thousands of numbers it is best to let a computer do the sorting instead. In my spreadsheet application, numbers that are put into cells can be sorted by row or column, and it is our responsibility to do it in the fastest and most efficient way possible. In the past we have used multiple algorithms to accomplish this, but recently we have adopted radix sort as the most favorable approach to sorting data.

First of all, radix sort is a special kind of algorithm which does not compare numbers when sorting. This type of sorting allows us to be faster than every sort algorithm which has to compare two numbers to get them in the correct order. Given the correct conditions, we can sort numbers faster than ever before. Previously, we have used insertion sort which has problems with large data sets. The more numbers getting sorted, the longer it takes to use insertion sort. Conversely, radix sort will sort large amounts of numbers much quicker. Even though there are other algorithms which sort large data sets faster than insertion sort – such as merge sort – it is still bound by the need to compare numbers which will ultimately throttle the speed.

In addition, radix sort is a stable sort. Stable sorts keep the numbers in relative order to each other while sorting, which gives multiple advantages. For example, if you have two numbers with the exact same value attached to a name in your spreadsheet, the names will maintain in the same order as before they are sorted instead of getting mixed up in the process. Unstable sorts such as quick sort cannot guarantee that those two items will be in the original order. Stability allows for data with multiple keys (such as a name and date) to be sorted properly.

**Justification**

Since radix sort is a non-comparison sorting algorithm, it can theoretically reach a time complexity faster than the fastest comparison sort. Common fast comparison sorting algorithms such as merge sort, quick sort, and heap sort all have Θ(nlog(n)) average time complexity. In brief, merge sort is Θ(nlog(n)) because the entire array needs to be iterated through – which takes *n* time – and half of the array is sorted every pass through, so it is iterated log(n) times. Conversely, LSD radix sort has Θ(*wn*) complexity for *n* number of keys with average key width *w*. This is because instead of comparing two numbers and moving them to the correct position in two steps, you can read the least significant digit and move the number to the correct position in one step.

In the cases where w is less than log(n), radix sort can achieve a practical efficiency advantage over merge sort. Given the conditions that the data consists of integers of a constant width, radix sort will achieve a linear runtime because *w* will be fixed. Depending on the data type you can also convert them to integers to be usable with radix sort, such as how all Extended ASCII characters are 8-bit numbers. The possibility of falling below Θ(nlog(n)) presents a strong argument towards radix sort because the highest performance comparison sorts cannot fall below that mark. The difference is obvious for slower comparison sorts such as insertion sort. Insertion sort will iterate through the array one item at a time, moving the item to its correct position with another round of iteration; resulting in an O(n2) runtime. With a runtime of O(n2), it works the best with small arrays. Larger data sets cause insertion sort to take exponentially slower time compared to radix sort which would require the number of keys and the length of the key to be equal to reach O(n2) time.

Stability in a sorting algorithm can depend either on the implementation or the nature of the sort. LSD radix sort is stable by nature. There are multiple different implementations of LSD radix sort – such as using queues – but they all function similarly: iterate through the array, scan the least significant digit of every number, and move them to their appropriate place in another array. Since the array is being read left to right, the moving portion will move the leftmost of two identical keys first, effectively keeping their relative positions constant. This also explains why radix sort is non-comparative. As mentioned before, digits can be read in order to move the number to the correct position in one step. Assume we have a list to be sorted: 34, 651, 78, 54 ,652, 13, 92. The first pass will read the underlined digit from left to right, move the numbers, and create the new list: 92, 13, 34, 54, 651, 652, 78. The second pass will read the underlined digit again, and create the sorted list: 13, 34, 54, 651, 652, 78, 92. Both the original and sorted list have duplicate 651 come before 652, proving that radix sort is stable.

On the other hand, a sorting algorithm such as quick sort is unstable because during the sorting process, it swaps elements that are not adjacent. Meaning that in sets with repeating numbers, the outermost elements will be swapped before innermost elements (with respect to the pivot value). Then, when the innermost elements are eventually swapped, they will be swapped out of order. Merge sort is different because it can be stable depending on the implementation while quick sort will always be unstable. As mentioned before, stability is important for the proper sorting of data sets with more than one key.