If given a task to sort a million fixed-length 5-digit integers used to represent some sort of non-unique identifier, there are many sort options to use to make this process seamless and efficient. Two of those common sorting algorithms used today are called Quicksort and Mergesort. They streamline this process of sorting large arrays of numbers, where in a business setting, could save a significant amount of time and ultimately money as well. As a matter of fact, due to its generalized purpose and compatibility within almost any context, these algorithms are preferred due to its runtime efficiency. However, I am here to tell you that within the context of how our data is being managed and represented currently, there is even a faster algorithm that can help sort the data which can help save you even more time and more money. This algorithm which is called Radix Sort can sort the data within the spreadsheet application 2-3x faster (Figure 1 Proof 1) than Quicksort and Mergesort. Both QuickSort and Mergesort deal with runtime complexities logarithmic to the number of elements, with Quicksort having the worst worst-case runtime scenario. The Radix Sort complexity deals with the number of elements and the size of the data representation (word size). Since our spreadsheet only deals with fixed length numbers, the benefits of Radix Sort really come into play. On another note, a benefit of Mergesort is the idea of data stability and maintaining original relative order in the case of duplicates. Radix Sort also is a stable sort and offers that advantage over the Quicksort which is not stable. Finally, an important restriction we must consider is the hardware capabilities and limitations. Radix Sort requires more memory than Quicksort for example, as it requires extra memory to deal with the actual sorting. However, our current hardware will not be limited by this and for the sake of argument can be ignored. That being said, Radix Sort offers the advantages found in both of the commonly used comparison based sorting algorithms while also guaranteeing a faster sort time.

There are two major factors in determining the speed of an algorithm: the implementation and the benchmark. The Big-O notation of Quicksort, Mergesort, and Radix Sort are O(n2), O(n log(n)), and O(nk) (where k is the word size) respectively. However, Quicksort generally tends to run at the same time complexity of Mergesort but the worst-case scenario must be noted as Big-O notation is an asymptotic representation of its runtime. Furthermore, since we are dealing with fixed length integers, the complexity of Radix Sort really is an O(n) since the multiplicative constant k can be ignored. But again Big-O notation is an asymptotic representation as k and n head toward infinity. Therefore, in order to find under what circumstance which algorithm is asymptotically preferred, we can set Equation 1:

This shows that asymptotically, n must be sufficiently large enough where the log(n) equals to the word size to at least match the complexity of O(nlog(n)). As n increases, Radix sort becomes asymptotically more efficient. In the case of our spreadsheet application as long as the number of elements is sufficient enough and the integer is of fixed length, the Radix Sort will be faster than either the Mergesort or the Quicksort. However, runtime complexity is not the only factor in efficiency. The implementation of the algorithm and how it tailors to the hardware, also known as its benchmark, plays a large role in determining whether Radix Sort is actually faster than Quicksort or Mergesort. For example, Quicksort makes the assumption that each comparison is O(1) which is not the case on actual hardware when taking things like cache misses and variable sized elements into consideration. At least for Radix Sort, while it is proven to be asymptotically faster at a sufficient n, the actual impact is how costly the overhead is for the required auxiliary memory. Erik Gorset1 had recorded benchmark results using a Radix Sort on k = 4 for 32-bit integers relative to other sorting algorithms using a recursive approach and an insertion sort to reduce some of the overhead when handling the low remaining elements. While benchmarks vary greatly from hardware to hardware, what can be extrapolated at least from the asymptotic proof and an actual implementation of Radix Sort is that in the case of our spreadsheet application that uses a fixed length of non-unique 5-digit integers and sufficiently large enough dataset to satisfy Equation 1, the Radix Sort will be the better sorting approach than either Quicksort or Mergesort.

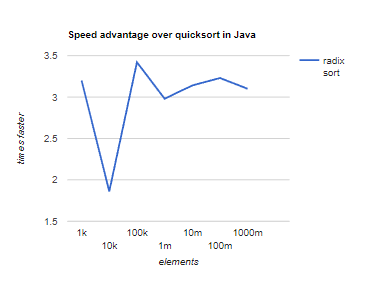


Figure 1 Source: <https://erik.gorset.no/2011/04/radix-sort-is-faster-than-quicksort.html>

Proof 1:

1. Let n = 10000 elements, k = 5
2. O(nk) = 50000, O(nlog(n)) ≈ 133000
3. O(nk) : O(nlog(n)) = 2.66 : 1
4. ⸫ 2.66 faster

1 https://erik.gorset.no/2011/04/radix-sort-is-faster-than-quicksort.html