36.4 Summary

Radix Sort vs. Comparison Sorts. In lecture, we used the number of characters examined as a cost model to compare radix sort and comparison sort. For MSD Radix Sort, the worst case is that each character is examined once for NM characters examined. For merge sort, MNlogN is the worst case characters examined. Thus, we can see that merge sort is slower by a factor of logN if character comparisons are an appropriate cost model. Using an empirical analysis, however, we saw that this does not hold true because of lots of background reasons such as the cache, optimized methods, extra copy operations, and overall because our cost model does not account for everything happening.

Just-In-Time Compiler. The “interpreter” studies your code as it runs so that when a sequence of code is run many times, it studies and re-implements based on what it learns while running to optimize it. For example, if a LinkedList is created many times in a loop and left unused, it eventually learns to stop creating the LinkedLists since they are never used. With the Just-In-Time compiler disabled, merge sort, from the previous section, is indeed slower than MSD Radix Sort.

Radix Sorting Integers. When radix sorting integers, we no longer have a charAt method. There are lots of alternative options are stilizing mods and division to write your own getDigit() method or to make each Integer into a String. However, we don’t actually have to stick to base 10 and can instead treat the numbers as base 16, 256, or even base 65536 numbers. Thus, we can reduce the number of digits, which can reduces the runtime since runtime for radix sort depends on alphabet size.

Previous
36.3 Radix Sorting Integers

Next
36.5 Exercises

Last updated 1 year ago