Matt Ritchie

CS360

3-9-18

Empirical Sorting Comparison

In the beginning, when we investigated Insertion Sort, I found the runtimes did not vary significantly between allowing elements 1->32768 and elements 1->1024 across all n’s used. However, in the cn^2 line we added to find the best fit asymptotic curve, with my computer’s results I found c to be 0.75 for the element possibilities of 1->1024, while the 1->32768 range produced a c of 3. (The data is shown in fig. 1, and the graphs are fig. 2 and fig. 3, labeled with their respective 1-> # ranges.)

Merge Sort’s runtime was exactly the same for both ranges of elements allowed, so both of their C values calculated to 5.5 in the asymptotic equation cn\*log(n) (fig. 4). Heapsort also uses cn\*log(n) and both large and small integer ranges take the exact same runtime. However, Heapsort produces a c of 0.5 (fig. 6). The third sort that uses the asymptotic equation cn\*log(n) is Quicksort, but it varies somewhat in the runtimes between the 1->32768 range and elements 1->1024 range. The 1->32768 range of elements actually took slightly over half the time of the 1->1024 range, and produced a c value of 3.8 instead of the 1024 range’s c of 6 (fig. 5). Interestingly, with the longer runtime, the 1->1024 range of Quicksort (fig. 7) curves upward steeper than the larger range of possible integers (fig. 8).

Even though 3 sorting algorithms follow the same O(nlogn) behavior, they are not the same in actual run time. The empirical run times reflect the actual procedure used. For instance, with the n’s used over all the algorithm implementations, Heapsort goes from a runtime of a little over 100 commands to roughly 525,000, while Quicksort goes from just under 200 to over 7,000,000. Just for comparison, Merge sort ran from about 400 to 5.5 million. Regardless, the runtimes for all the n’s for each sort trace out roughly the same logarithmic curve shape, which Big O notation ignores the multipliers and other constants (the c’s and Counting’s k).

Counting sort takes longer with the larger range (fig. 9), naturally, but it produces a nearly perfect linear result for the increasing n’s. The asymptotic equation given was cn+k, a linear equation. That predicted linear behavior fits the empirical results to a T. Both 1-> # ranges had a slope (c) of 5, although the 1->32768 range (fig. 9) needed a k of 130,000 but the 1->1024 range (fig. 10) only needed a k of 5,000.

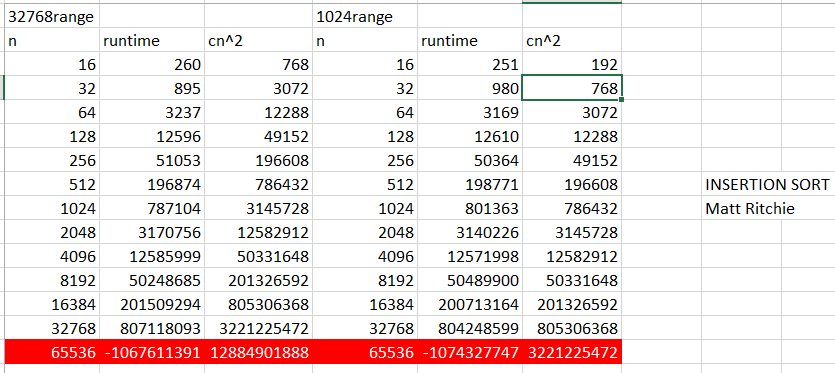
For small n’s, say under 1000, Heapsort would work best. Even though Counting sort is the best for around 1000 elements, at very small n’s like 16, Counting sort starts at a time of four thousand instead of all the other sorts’ couple hundred (figs. 5, 1, 4, 11). If your data set is large, say over 10,000 elements, Counting Sort is by far the best, topping off at something like 10% of the runtime of the other sorts, nothing else is even close.

Summary of Tested Algorithms:

|  |  |  |
| --- | --- | --- |
| Sorting Method | Asymptotic Behavior | Empirical Asymptotic Formula (1024/32786) |
| Insertion | O(n2) | 0.75n2 / 3n2 |
| Merge | O(nlogn) | 5.5nLog(n) / 5.5nLog(n) |
| Heap | O(nlogn) | 0.5nLog(n) / 0.5nLog(n) |
| Quick | O(nlogn) | 6nLog(n) / 3.8nLog(n) |
| Counting | O(n) | 5n + 5,000 / 5n + 130,000 |

Appendix:

Fig. 1:



(red highlighting denotes data that was excluded from the graphs, probably caused by an overflow error)

Fig. 2:

(Adding a legend is easy, but Excel won’t let you edit the actual text of the entries, this applies to all the other graphs here too.)

Fig. 3:

Fig. 4:

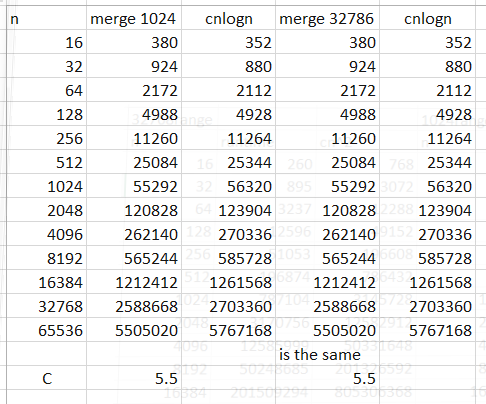


Fig. 5:

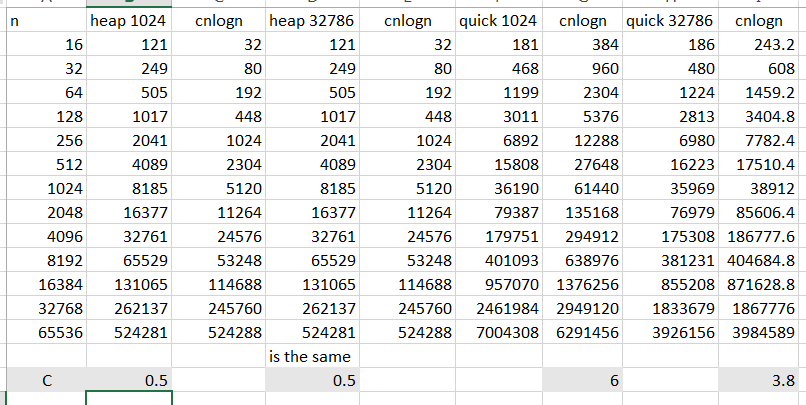


Fig. 6:

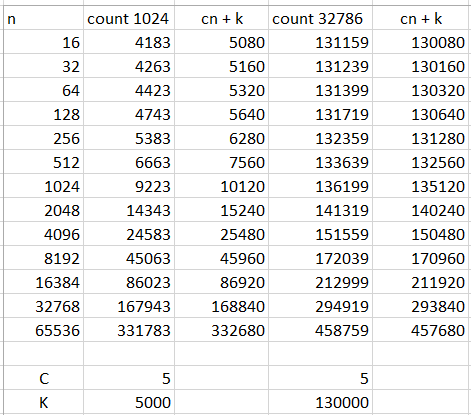
Fig. 7:

Fig. 8:

Fig. 9:

Fig. 10:

Fig. 11:



Other relevant graph(s) not directly referenced: