Asgn 3 Write-Up

Brandon Chuang

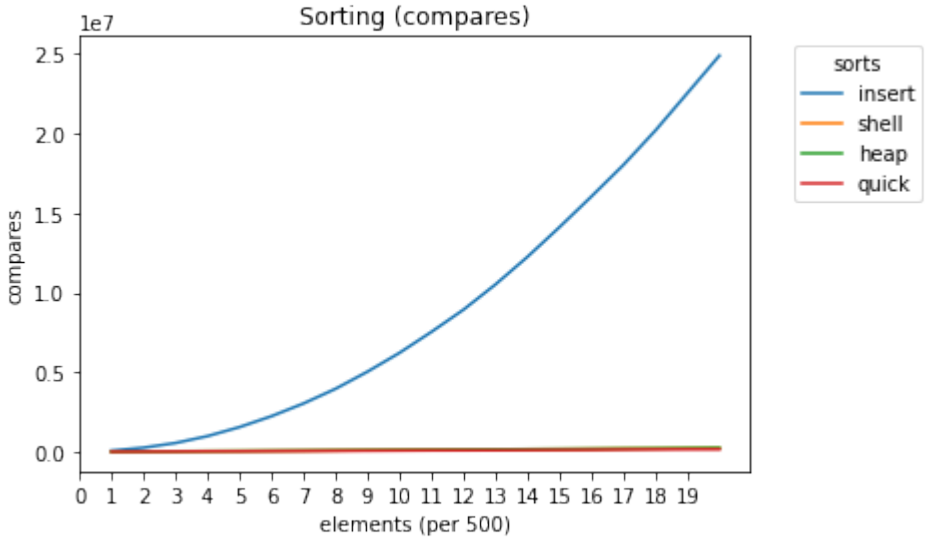
10/18/2021

An analysis of the efficiency of various sorting algorithms.

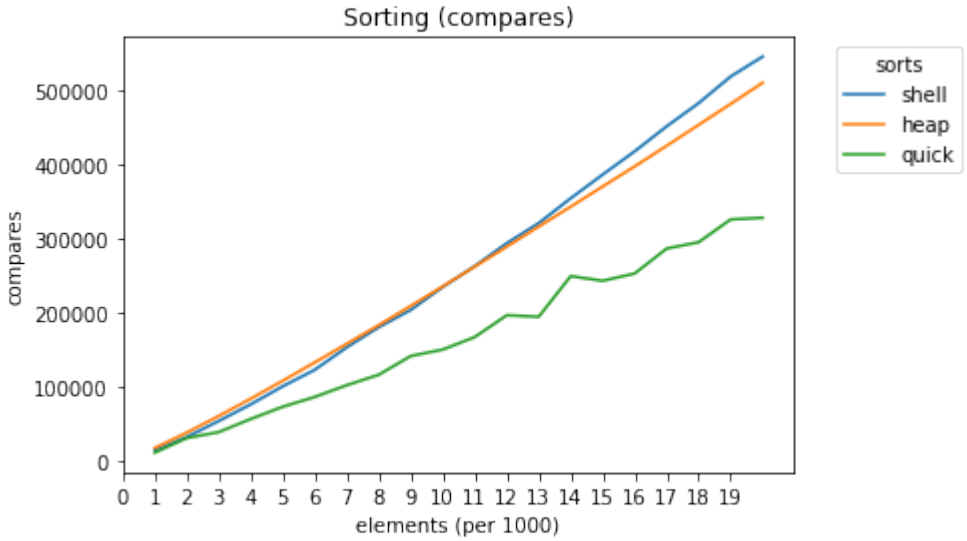
1 Introduction

This is an analysis for the differences in compares/moves between heap, insert, quick, and shell sort. Insight as to why the results are the way they are are provided.

2 Compares



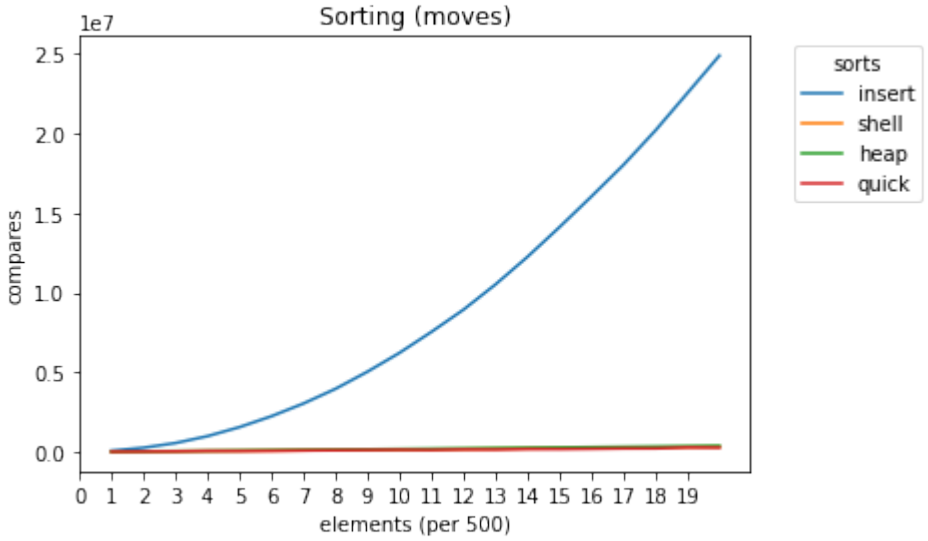
Since Insert Sort is not very efficient, it ruins the grpah for the rest of the sorts. This is because it's too simple and straighforward to take any shortcuts or do anything that cuts down on runtime: all it does is search the remaining unsorted array for the lowest number every time. Fittingly, this sort has the least efficient big O notation of $O(n^2)$.



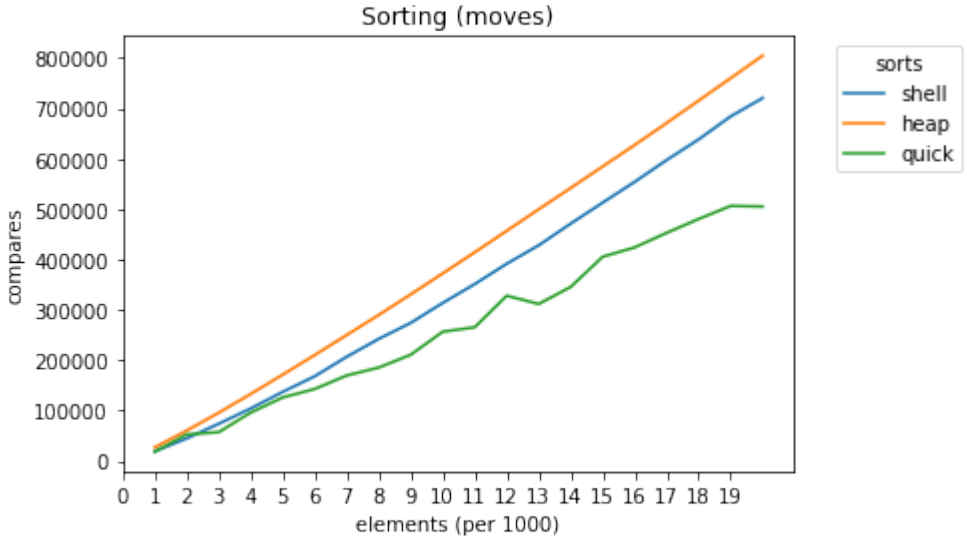
there's some interesting things to note about the rest of the sorts is that their amount of compares seem to be on the same trajectory. All of their big O notations include a log(n), so it would make sense that they have similar trajectories. A noticeable trait of this graph is that it shows shell is faster than heap up to a certain point, and then the two switch, meaning one is faster for smaller arrays and one is faster for bigger arrays.

Quick sort also has an interesting trend, since its line is not as straight and stable as the other three sorts. Because of the pivot chosen being a completely random number, although it is, on average, faster, its worst case scenario is obscenely bad. If the pivot is either one of the biggest or smallest element, it creates more work for itself by comparing and moving too many smaller values to its right, only to move them back to the left again when less extreme values are chosen. This leads to seemingly random outliars when the worst case scenario happens to get chosen early on, when the array is still completely unsorted.

3 Moves



As with compares, insert has the same exact problem where it is so inefficient compared to the rest of the sorting methods that it makes the other lines incomprehensible, because of its big O notation.



All the sorts follow mostly the same trajectories as the compares, but it is interesting to note that shell and heap sort no longer intersect. Heap stays less efficient over the entire time.

4 Conclusion

Aside from Insertion Sort, all of the sorting methods follow the same trajectory and therefore share the same elements in their Big O notations (log(n)). The actual difference in runtime between shell and heap sort cannot be deduced from this data (since we do not know if compares or moves are faster execution-wise). Even though quicksort may be the fastest sorting method tested here, it is somewhat held back by its notoriously slow worst case, because of its unique method involving a pivot.