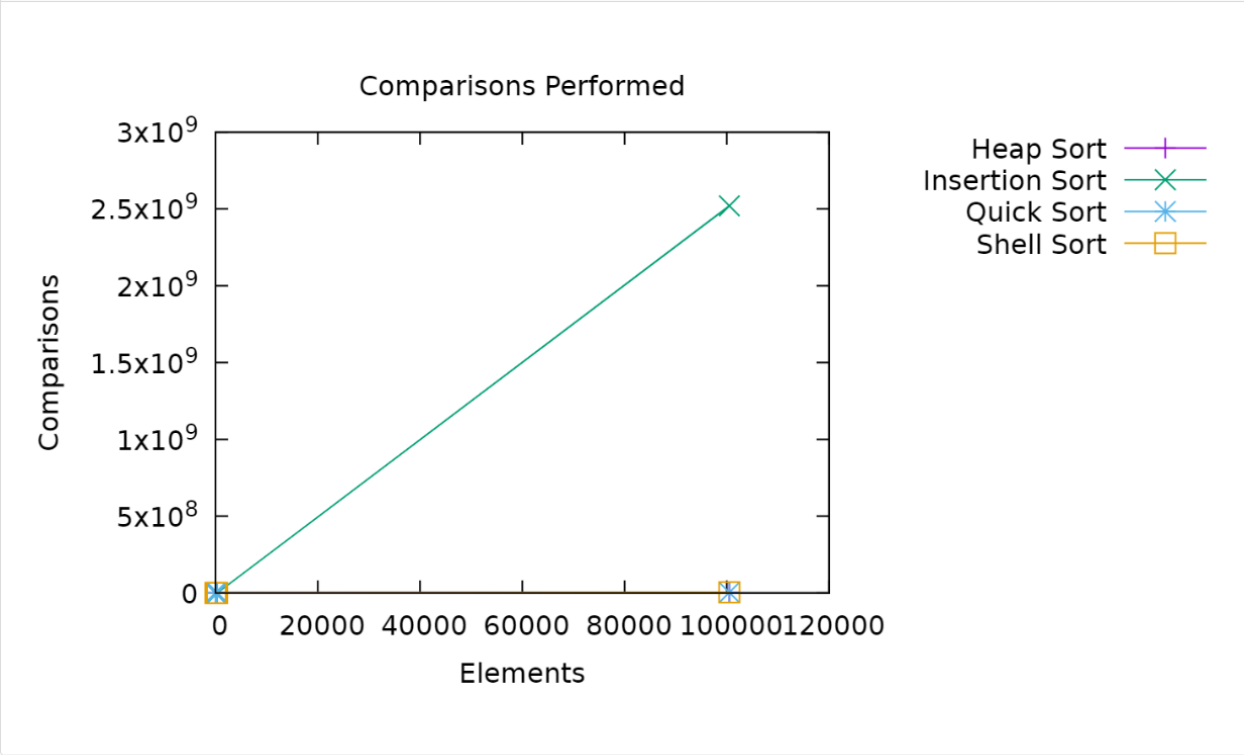
Alex Clanton

10/15/2021

Professor Long

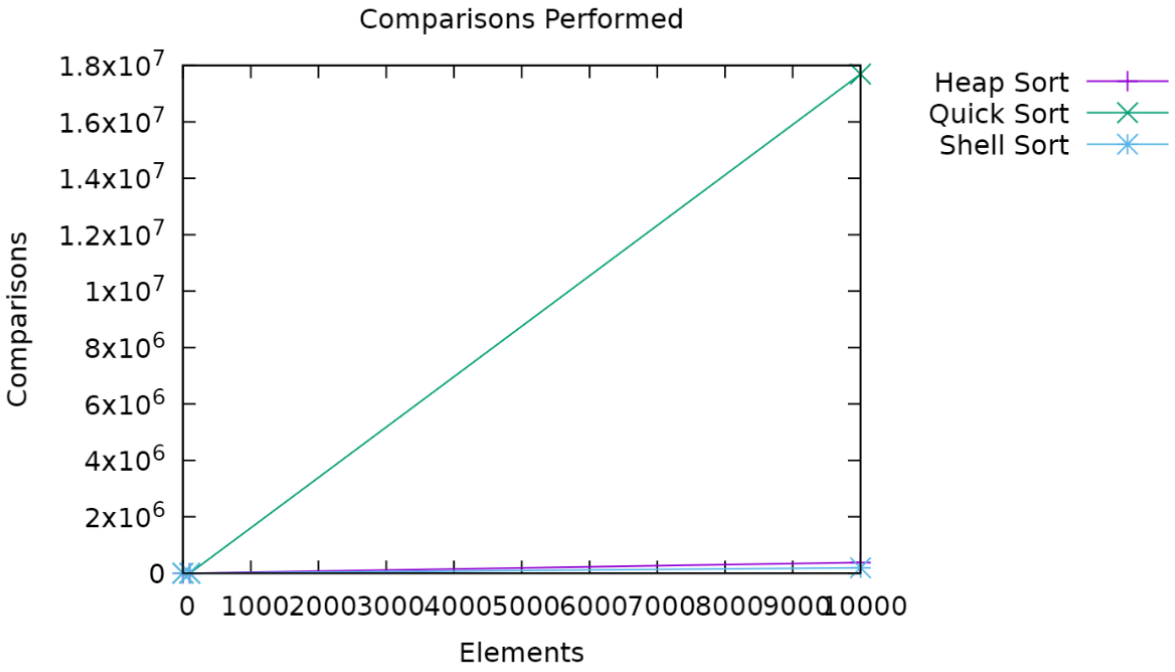
CSE 13S

The main takeaway from programming and graphing sorting efficiency is Insertion Sort is dangerous if you allow an unlimited or computer-mandated amount of memory to the user. At 100k elements, my computer took a while to process the near 2.5 billion moves.



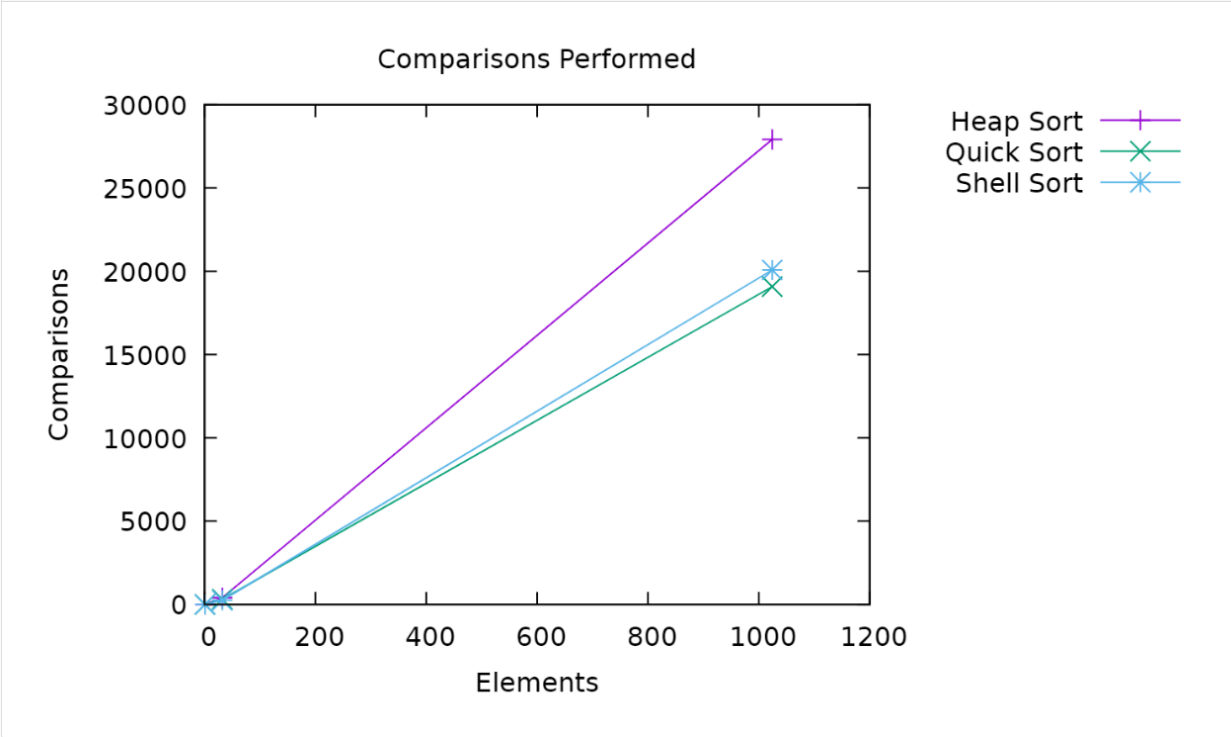
Insertion sort is good for ‘small’ arrays that are at least 100 or less, preferably something like 0-50 sized. Due to its small C, or constant value allows its efficiency on low elements. But its O(N^2) means you are playing with fire with any decent-sized array. At this scale, everything looks good compared to insert, with quick being the fastest.

However a word of caution in making everything sort a quick sort. With a small chance, Quick Sort can just be as dangerous. An array that is sorted and then flipped, and presented to Quick Sort is just as bad as insertion and is in fact O(N^2).

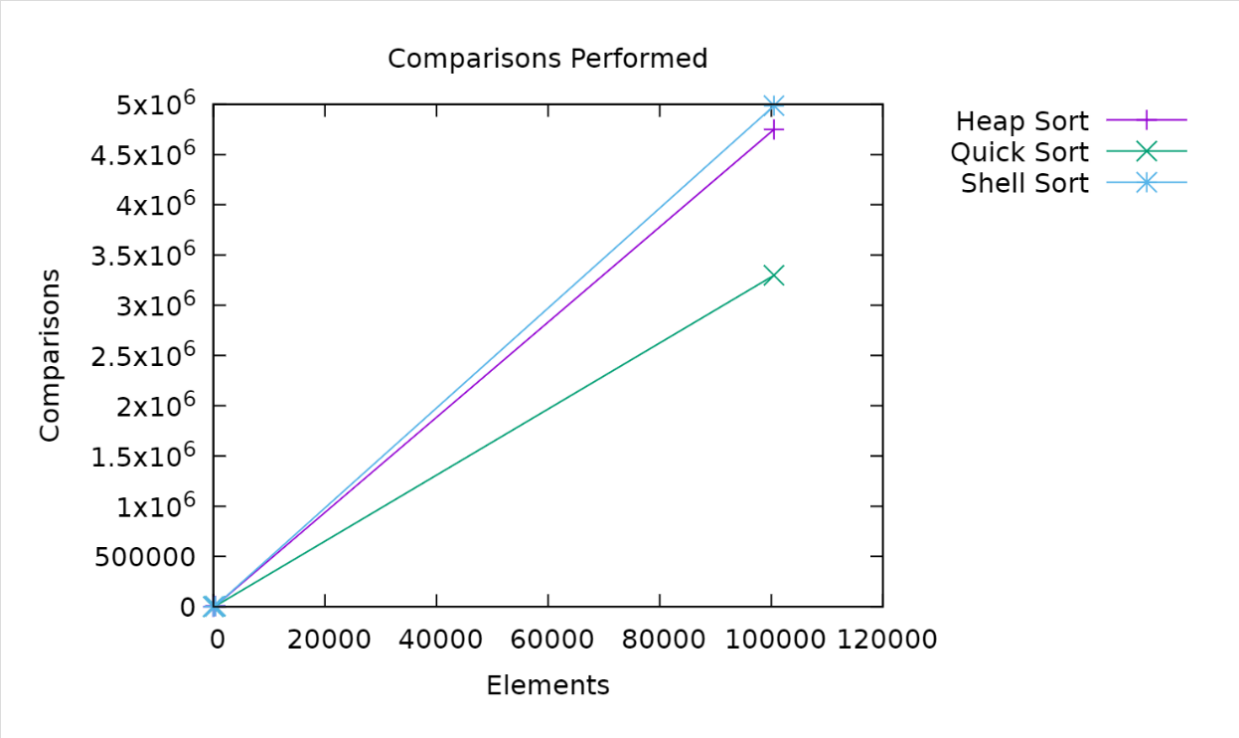


This 10,000 element array is sorted then flipped and can achieve the worst case for Quicksort O(N^2) so Quick Sort is good, but knowing to weed out any potential worst-case scenarios is important for running the algorithm. Running a large dataset and getting hung up on N^2 could be disastrous.

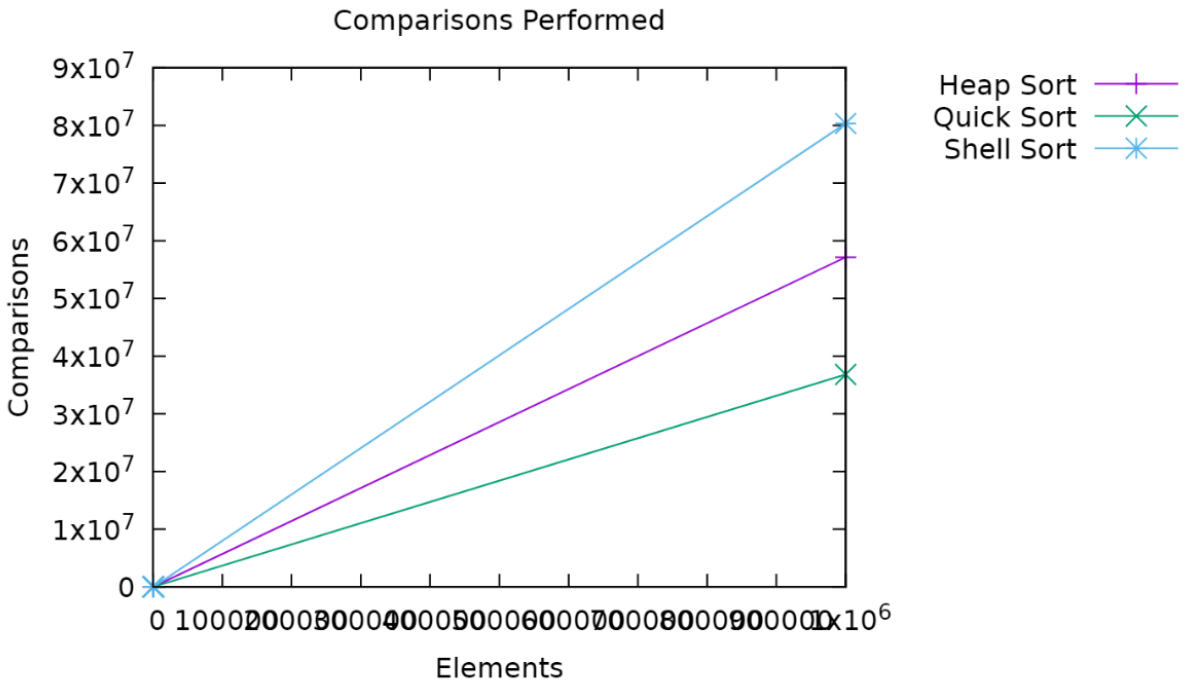
For a relatively ‘medium’ sized array of 1000 elements, Quick Sort is the fastest as long as it’s randomized. However Shell sort is not that far behind, and without the potential worst case of Quick Sort, with Knuth gap of n^3/2 and a small constant. Shell is very effective for a medium-sized array with potential worst cases that could plague Quick Sort.



On very large arrays 10k to 20k and beyond Shell sort loses out with its small C value to Heap Sort which has a large C but a smaller O(nlog(n)). This means Heap is a good choice for very large arrays, 100k+ and will be consistently fast. In comparison, it is quite a bit slower than Quick. But if there's an outside chance you know potential input is a worst-case, Heap is an ok choice.



In this case, a 1 million or more randomly sized array, Shell continues to get worse, Heap improves, and Quick remains the fastest with the chance of being very very bad.



Overall programmers can conclude that different sorts are better for types of expected input, and different risk levels must be accounted for if anticipating weird or non-natural data. Quicksort is a good example where depending on randomized data it can vary from different sized elements positions. Give Quick Sort a 200 element array that leans toward its worst case, it can shoot up in difficulty, but a 200 randomly sized array is very very fast. This does lead to some choppiness overall versus other sorts, but it remains the fastest overall at O(n log n), the same speed as heap but a smaller C value.