Phoenix Hillard

For this project, seven different sorting algorithms were programmed and tested for the amount of time each algorithm took when sorting an array with a variable number of elements. The number of elements started at 10,000 and incremented by 10,000 up to 200,000 elements. The algorithms tested were bubble sort, selection sort, insertion sort, merge sort, shell sort, quick sort, and heap sort. All algorithms were validated to work through the same tester that the data collection used. For comparison purposes, I would like to divide the algorithms into two categories: the first category being bubble sort, insertion sort, and shell sort; the second category being selection sort, merge sort, quick sort, and heap sort. The first of the two categories I've put together because all of them compare the whole array to itself, while the second category divides the array in some way. The sorting algorithms in the second category are also significantly faster.

The slowest out of all the algorithms by a wide margin was bubble sort, taking a little longer than 68 seconds to sort 200,000 elements. The way my implementation of bubble sort worked, it had only ever compared two values that were directly next to each other in the array, which meant that the loop used to verify whether or not the array was sorted had to run through many times. The next slowest was insertion sort, which compared values directly next to each other, and then sorted everything before values that were switched to place everything in order. This algorithm took slightly longer than 18 seconds to sort 200,000 elements. Next was shell sort, which worked very similarly to insertion sort, except had a gap that was used. The gaps would get smaller until the algorithm compared elements that were next to each other, which should have been mostly sorted at that point. If two numbers were switched, the program would check the next gap prior to the swap to order things correctly. This algorithm took a little under 16 seconds to sort 200,000 elements. In the case of all of these algorithms, the amount of the array being compared is never reduced.

In the second category, the slowest is selection sort, which works by looking for the smallest value in the array, then swapping it with the beginning of the array. The index from which the array begins to search for the next smallest element is then incremented, which also means that the amount of the array being looked through decreases with each iteration. This algorithm took 4.6 seconds to sort 200,000 elements. The remaining three algorithms all timed similarly, each taking somewhere close to 0.03 seconds to sort 200,000 elements. Heap sort was the most predictable and most efficient of the sorting algorithms overall, sorting through 200,000 elements the fastest at 0.029 seconds. This algorithm worked by sorting the array in a heap, which meant that its ordering process did not appear to be in order while it was sorting. After being sorted into a heap, the first element, which was the largest value, was swapped with the end of the array. The array would then need to be reorganized into a heap while excluding the largest element, which was just added to the end. The end that would be considered would be

decremented and the process would continue until the whole array was in order. This was also the only algorithm that fit a linear trendline when graphed. Both merge sort and quick sort did not perform in a predictable way, both only really fit a moving average trendline. Merge sort specifically had an unexpected sharp increase in the amount of time it took to sort at 60,000 to 80,000 elements, all of which it sorted slower than 200,000 elements. After 80,000 elements it mostly evened out into a predictable pattern. Merge sort worked by splitting the array in half repeatedly, then sorting once the array was very small, then comparing the pre-sorted smaller arrays into a larger and larger array until all elements were sorted. This was done through recursion where there would be a left and right half which would split into their own left and right halves repeatedly. This also meant that the number of elements being compared at once was very small. Quick sort was more unpredictable than merge sort, having a spike at 40,000 elements, then another spike at 90,000 to 120,000 elements. The algorithm finished sorting 200,000 elements in 0.034 seconds, which is close to the time it took to sort 40,000 elements at 0.032 seconds. Quick sort worked by having a 'pivot' point to begin comparing elements to, in this case the last element was always used. The array would be sorted relative to the value of the pivot, after which the pivot would be moved to where it belonged relative to everything else. The array would then be split into two smaller arrays, one side would be everything smaller than the value of the pivot, the other being everything larger. This algorithm was recursive and would continue the process through these sub-arrays, then both arrays would be combined again. Notably all of the fastest algorithms, merge sort, quick sort, and heap sort, utilized recursion in some way.

Both merge sort and quick sort have higher space complexity compared to all the other algorithms which is part of the sacrifice for the faster sort times. Additionally, in the worst case, quick sort can have a time complexity of $O(N^2)$, putting it right next to bubble, selection, and insertion sort, while both merge sort and heap sort have a worst case time complexity of O(N log N) (GeeksforGeeks, 2023). From what I can tell, heap sort may be the only algorithm that has no immediately apparent downside.

Citations

GeeksforGeeks. (2023, August 7). *Analysis of different sorting techniques*. https://www.geeksforgeeks.org/analysis-of-different-sorting-techniques/#

References

GeeksforGeeks. (2023a, July 25). *Heap sort - data structures and algorithms tutorials*. https://www.geeksforgeeks.org/heap-sort/

GeeksforGeeks. (2023c, October 16). *Quicksort - data structure and algorithm tutorials*. https://www.geeksforgeeks.org/quick-sort/

Programiz. (n.d.). *Heap Data Structure*. https://www.programiz.com/dsa/heap-data-structure

Wikimedia Foundation. (2023, December 8). *Shellsort*. Wikipedia. https://en.wikipedia.org/wiki/Shellsort