Jadon Stronge

Problem and Strategy

Not knowing the order of five different sorting algorithms (capable of sorting arrays of integers in ascending order) that exist in a statically linked library, I implemented code that tests a variety of arrays different in both size and element order. The program passes my arrays along with the size of each array to the library and records the time it takes for each sorting function to complete. Knowing the five possible options of sorts in the library, the objective is to deduce which sort is which by comparing the times of each sort against the arrays with an order of elements that would produce their “best” and “worst” O(n) (mentioned in the ‘How I Figured It Out’ Section).

What I Figured Out

Mystery sort one continuously produces some of the fastest times despite the size or order of elements. The option that has the fastest Big O despite element order is Merge Sort. I believe mystery sort one is merge sort.

Inversely, mystery sort five produces one of the longest times among the five sorts during most cases. However, it’s time is relatively consistent despite the order of the array and this is seen amongst all array sizes. I believe this is a selection sort.

Mystery two and three are optimized bubble sort and insertion sort respectively. Reason being, their best case Big O (mentioned later) is O(n), which is faster than any other sorting case of the other algorithms. That is consistent with results of the times when elements of the array are in ascending order. One hurdle to overcome was differentiating between optimized bubble sort and insertion sort when they have the same Big O best and worst case following the same criteria. However, from what I’ve been able to research, “bubble sort is [always] slower than insertion sort” (click links\*). From what I understand of the functions, this is due to bubble sort creating temporary values in order to swap neighboring elements - increasing its time complexity in comparison to insertion sort. This is seen when elements are descending or shuffled.

Mystery four is quicksort for the following reason: Firstly, process of elimination. Secondly, ideally, we would see a consistent, relatively ‘large’ time from quicksort at times of elements ordered in ascending and descending order (worst case) and a dramatic drop in time when elements are shuffled (best case). That is seen of mystery sort four across all array sizes.

How I Figured It Out

What I know about the five possible options:

**Optimized bubble sort**

Best Case - O(n) when the elements are already sorted

Worst Case - O(n^2) when the elements are in descending order

**Insertion Sort**

Best Case - O(n) when the elements are already sorted

Worst Case - O(n^2) when the elements are in descending order

**Merge Sort**

Best Case - O(nlgn)

Worst Case - O(nlgn)

**Quicksort (with the last element as pivot)**

Best Case - O(nlgn) when the pivot divides the list into two equal halves

Worst Case - O(n^2) when elements are in ascending or descending order

**Selection Sort**

Best Case - O(n^2)

Worst Case - O(n^2)

Graphs Mapping Growth Rate

(Check excel sheet in root folder\*\*)

Sources

\*

* <https://stackoverflow.com/questions/17270628/insertion-sort-vs-bubble-sort-algorithms>
* <https://pediaa.com/what-is-the-difference-between-bubble-sort-and-insertion-sort/>