Colby Wirth
Professor Behrooz Mansouri
10 December 2024

Assignment 9 Results from MySorts Implementation

Analysis 1

This analysis compares the performance of two consecutive quick-sort method calls with a comparator based on each Tweet's LocalDateTime field. I measured the performances of these two method calls across five iterations. On average, the first method call took 67 milliseconds to execute, and the second method call took 28 milliseconds to execute. The second method performed 46% faster on average than the first method call. Therefore the quick-sort algorithm will perform faster if the Array has previously been sorted.

Analysis 1 - QuickSort with LocalDateTime

Iteration	Quicksort 1 (ms)	Quicksort 2 (ms)
1	69	46
2	70	46
3	63	30
4	63	32
5	70	28
Averages	67	36.4
Difference		**0.46**

Table 1

Analysis 2

This analysis focuses on if quick-sort is a stable algorithm. An algorithm is stable if it maintains relative order of an ordered collection based upon a previous sorting algorithm after the new algorithm is called. It is not possible to determine if the quick-sort algorithm is stable without prior knowledge of the indices of the top tweets in Table 2.

Analysis 2: Top 10 Results

ID	Date:
2000000	2009-06-16T05:24:09
2000001	2009-06-15T20:14:49
2000002	2009-06-24T22:49:40
2000003	2009-06-23T14:56:27
2000004	2009-06-23T10:18:16
2000005	2009-06-25T04:47:18
2000006	2009-06-23T16:48:36
2000007	2009-06-16T05:14:02
2000008	2009-06-16T03:11:21
2000009	2009-06-15T22:27:19

Table 2

Analysis 3

This analysis compares the performance of a quick-sort and radix-sort methods on the an array of custom Tweet objects. The field used for sorting is Tweet ID, a 32 bit integer. I measured the performances of these two methods across five iterations. On average, the quick sort method took 14.8 milliseconds to execute, and the radix-sort took 27.0 milliseconds to execute. The quick-sort algorithm performed 45% faster than the radix sort on average. When measuring run-times, the quick-sort is the superior algorithm.

Analysis 1 - QuickSort and RadixSort with ID

Iteration	Quicksort (ms)	RadixSort (ms)
1	16	33
2	13	22
3	15	32
4	15	23
5	15	25
Averages	14.8	27.0
Difference	**0.45**	

Table 3