Specifications of the environment

CPU model	Intel(R) Core(TM) i7-10510
OS version	Windows 10 Home
RAM size	GB 8.00
Cache size	256 kb
Java version	17
Laptop brand	Lenovo yoga

Random Array, Numiter=100, Threshold = 10

Algorithm > Size	Quick Sort Class (milliseconds)	Quick Sort Recitation (milliseconds)	Radix Sort b=2^10 (milliseconds)	Merge Sort Recursive (milliseconds)	Merge Sort Iterative (milliseconds)	Java Sort (milliseconds)
10,000 Avg/deviation	1.31/1.35	1.39/1.44	0.74/0.58	1.54/1.87	1.87/2.67	0.63/0.35
50,000 Avg/deviation	6.34/0.19	7.01/0.29	2.65/0.14	6.69/0.35	6.66/0.29	2.72/1.35
100,000 Avg/deviation	15.39/0.14	15.5/0.58	6.39/0.004	14.66/0.31	14.89/0.48	6.05/0.16
500,000 Avg/deviation	139.95/25.59	140.45/27.59	40.15/18.1	226.57/53.8	284.39/128.08	36.77/10.66
1000000 Avg/deviation	456.06/182.8	462.76/177.71	97.99/40.99	634.7/197.7	638.93/291	97.41/41.9

Sorted increased Array, Numiter=100, Threshold = 10

Algorithm > Size	Quick Sort Class (milliseconds)	Quick Sort Recitation (milliseconds)	Radix Sort b=2^10 (milliseconds)	Merge Sort Recursive (milliseconds)	Merge Sort Iterative (milliseconds)	Java Sort (milliseconds)
10,000 Avg/deviation	34.05/35.35	131.12/130.35	0.4/0.51	2.13/2.6	2.02/2.48	0.4/0.52
50,000 Avg/deviation	Error	Error	1.69/0.7	6.19/1.64	6.48/1.79	1.57/0.62
100,000 Avg/deviation	Error	Error	2.98/0.69	11.34/1.27	11.51/1.45	2.92/0.59
500,000 Avg/deviation	Error	Error	14.24/5.04	111.61/14.25	113.58/14.6	13.74/4.13
1000000 Avg/deviation	Error	Error	34.84/18.25	248.94/34.67	258.4/43.2	32.47/12.97

Sorted decreased Array, Numiter=100, Threshold = 10

Algorithm > Size	Quick Sort Class (milliseconds)	Quick Sort Recitation (milliseconds)	Radix Sort b=2^10 (milliseconds)	Merge Sort Recursive (milliseconds)	Merge Sort Iterative (milliseconds)	Java Sort (milliseconds)
10,000 Avg/deviation	37.45/35.35	133.01/124.5	0.33/0.53	1.87/2.08	2.18/2.76	0.48/0.51
50,000 Avg/deviation	Error	Error	1.6/0.6	6.25/1.66	6.36/1.18	1.55/0.53
100,000 Avg/deviation	Error	Error	3.14/0.62	11.87/1.53	12.25/1.99	3.05/0.76
500,000 Avg/deviation	Error	Error	13.61/2.73	115.9/15.3	116.06/12.13	14.47/5.166
1000000 Avg/deviation	Error	Error	30.16/9.55	246.03/27.76	253.92/18.08	30.89/8.93

Radix sort results, Numiter=100, Threshold = 10

Range > Base	2	2^5	2^10	2^15	2^20	2^25	2^30
[0,2^10]	69.4/7.25	13.87/1.35	7.04/0.85	7.02/0.57	7.72/0.75	35.68/8.60	Space error
[0,2^20]	133.03/10.25	27.56/2.55	14.34/1.52	14.31/1.21	8.59/1.16	34.38/6.03	Space error
[0,2^30]	197.27/13.97	39.57/3.1	20.15/1.46	13.75/0.98	20.49/3.71	127.29/1.12	Space error

Explanation and analyzes:

Random Array:

• Quick sort: Both quicksort Sort Class and Quick Sort Recitation implementations perform similarly, but the class one looks better.

• Radix Sort: consistently demonstrates competitive performance, showcasing relatively lower execution times compared to the Merge Sort and Quick Sort implementations across various input sizes.

• Merge Sort: Both merge sort implementations have similar performance, which is consistently good across different array sizes, the recursive version looks better.

• Java Sort: exhibits superior performance with the lowest execution times among the sorting algorithms across different input sizes.

Sorted Increased Array:

• Quick Sort: Both implementations of the Quick Sort algorithm encounter stack overflow errors when attempting to sort an array that is already sorted in increasing order. This issue likely arises due to an uneven distribution of elements during the partition process, resulting in unbalanced recursive calls and excessive usage of the call stack.

• Merge Sort: Both versions perform well and have stable performance regardless of the array being sorted in increasing order.

• Radix Sort, Java Sort: These algorithms perform well and have stable performance regardless of the array being sorted in increasing order. These algorithms perform much quicker results of sorting.

Decreased Array:

• Quick Sort: Like the sorted increased array, both implementations of the Quick Sort algorithm encounter stack overflow errors when attempting to sort an array that is already sorted in increasing order.

• Merge Sort: Like the sorted increased array, both versions perform well and have stable performance regardless of the array being sorted in increasing order.

• Radix Sort, Java Sort: Like the sorted increased array These algorithms perform well and have stable performance regardless of the array being sorted in increasing order. These algorithms perform much quicker results of sorting.

Radix sort results:

The provided results show the performance of Radix Sort with different base values and array ranges.

The algorithms encounter "Space error" for the input sizes of 2^30, 2^20, and 2^10, indicating that they exceed the available memory or encounter issues related to space allocation and management.

From the provided table, it is evident that across different types of range arrays, the algorithms' performance time follows a consistent order from lowest to highest: 2, 2^25, 2^5, 2^20, 2^10, 2^15.

Summary:

The observed sort algorithm in the tables is in line with the theoretical analysis to some extent.

Quicksort has an average time complexity of O(nlogn), but it can deteriorate to O(n^2) in certain situations, like when sorting already sorted or reverse-sorted arrays.

Merge sort algorithm exhibits more stable and consistent performance, with better guarantees for worst-case time complexity These implementations perform well for random and sorted arrays (increasing or decreasing) with average time complexity of O(nlogn).

Radix sort demonstrates exceptional efficiency when sorting integers, especially when both the range and base of the numbers are known. It boasts an average time complexity of O(n), which is evident from the results of the performed tests. The outcomes clearly indicate that radix sort outperforms other sorting algorithms by a significant margin, further highlighting its superiority in terms of performance and effectiveness.

The Java sort algorithm consistently achieves the best results among the tested sorting algorithms, demonstrating superior performance and effectiveness.