Experimental Results

*Figure 1: Average runtime (ms) of various sorting algorithms on **randomly generated arrays** (size N from 2 to 2^14, y-axis in log scale). Each line represents one sorting algorithm.*

*Figure 2: Average runtime (ms) of various sorting algorithms on **already sorted arrays** (size N from 2 to 2^14, y-axis in log scale). Note the dramatic differences for some algorithms when the input is pre-sorted.*

Analysis and Conclusions

From the experimental results, several observations can be made about the sorting algorithms' performance:

- **Optimized QuickSort vs. Standard QuickSort:** The optimized QuickSort (with shuffling and insertion sort cutoff) outperforms the regular QuickSort for large N and for sorted inputs. In the random array chart, QuickSortOptimized runs faster than QuickSort for all N, and in the sorted array chart the standard QuickSort degrades significantly (due to its worst-case $O(n^2)$ on sorted data) whereas the optimized version maintains efficiency. This shows the value of optimizations like shuffling (to avoid worst-case pivots) and using insertion sort for small subarrays.

- **Effect of Input Order on Certain Algorithms:** Some algorithms benefit greatly from already sorted input. In particular, **BubbleSortOptimized** and **InsertionSort** run in almost linear time for sorted arrays (as seen in Figure 2, their lines are far lower for sorted input than for random input). The optimized bubble sort detects no swaps needed and terminates early, and insertion sort barely moves any elements when the array is already sorted. In contrast, **BubbleSort (naive)** and **SelectionSort** still perform $O(n^2)$ comparisons on sorted data, so their runtimes remain high in the sorted-case graph (nearly unchanged from the random case).

- **$O(n^2)$ vs. $O(n \log n)$ Scaling:** For large input sizes, the $O(n \log n)$ algorithms (MergeSort and QuickSort variants) massively outperform the $O(n^2)$ algorithms (Bubble, Selection, Insertion) on random data. This trend is clear in Figure 1, where the lines for MergeSort and QuickSort grow much more slowly as N increases, whereas the $O(n^2)$ algorithms shoot up steeply on the log scale. For example, at the largest N, the MergeSort and QuickSort algorithms complete in milliseconds, while BubbleSort takes orders of magnitude longer.

- **Recursive vs. Non-Recursive MergeSort:** The recursive and iterative (non-recursive) merge sort implementations have very similar performance, as expected. The non-recursive version avoids recursion overhead and can be slightly faster for very large arrays, but in our tests both versions yield comparable runtimes. This confirms that the primary factor in runtime is the $O(n \log n)$ complexity of

merging, rather than recursion overhead.

- **Linked List MergeSort:** Merge sort on linked lists also shows $O(n \log n)$ behavior. It is efficient for linked structures because it sorts in place without extra space and avoids poor random access performance of other algorithmsfile-h3tqhd1jaj4bb3a2urb4dffile-h3tqhd1jaj4bb3a2urb4df. In the random data chart, the linked list merge sort line is slightly above the array merge sort (due to pointer manipulation overhead), but it still grows similarly. This underlines that merge sort is a good choice for sorting linked lists, whereas algorithms like quicksort or insertion sort would be less ideal on linked data structuresfile-h3tqhd1jaj4bb3a2urb4df. The linked list merge sort outperforms the quadratic algorithms for moderate N and scales much better for large N.

Overall, the experiments highlight that **algorithm choice matters greatly depending on input size and data order**. Optimizations in QuickSort make it consistently one of the fastest methods for large arrays, whereas simple sorts like BubbleSort become impractical as N grows. For nearly sorted or small data sets, simpler algorithms (like insertion sort or optimized bubble sort) can perform adequately or even optimally. Having a mix of strategies (as in QuickSortOptimized) yields robust performance across different scenarios.