Performance Comparison of Selection Sort, Quick Sort, and Merge Sort

This report compares the performance of three sorting algorithms by measuring the number of comparisons and swaps made when sorting the same dataset of 26,397 strings.

Part 1: Performance on Unsorted Data (very_long.txt)

Algorithm	Comps	Swaps
Selection	348,387,606	26,397
Quick	3,506,661	3,368,641
Merge	37,917	0

Part 2: Performance on Sorted Data

Algorithm	Comps	Swaps
Selection	348,387,606 (same)	26,397
Quick	4,558,407	4,501,752
Merge	21,920	0

Analysis and Discussion

Observations:

1. **Selection Sort**:

- O(n^2)
- Minimal number of swaps (exactly n swaps in worst case)
- Consistently the worst performer in terms of comparisons -but easy to implement-

2. **Quick Sort**:

- On unsorted data: 3,506,661 comparisons (best performance)
- On sorted data: 4,558,407 comparisons (worse performance)
- High number of swaps in both cases

3. **Merge Sort**:

- Most efficient in terms of comparisons
- Performs better on sorted data (21,920 vs 37,917 comparisons)
- No swaps needed as it uses merging instead of in-place swapping
- Consistent O(n log n) performance regardless of input order

Why do algorithms behave differently on sorted vs unsorted data?

- **Selection Sort**: Unaffected by input order because it always searches for the minimum element in the remaining unsorted portion, requiring the same number of comparisons regardless of initial order. but this makes it worse in all the cases
- **Quick Sort**: Performance depends heavily on pivot selection and inputs.
- **Merge Sort**: Shoes better performance when the data is sorted because it does less merge but it's n log n in any case

Conclusion:

Merge sort is the fastest, becuase it uses merging. where as quick sort and selection sort perform very poorly.