THEORETICAL COMPLEXITY ANALYSIS

SORTING ALGORITHMS – COMPARATIVE STUDY

Design and Analysis of Algorithms (DAA) Project

---

1. Time Complexity Analysis

Algorithm	Best Case	Average Case	Worst Case	Stable?
Bubble Sort	$O(n)$	$O(n^2)$	$O(n^2)$	Yes
Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$	No
Insertion Sort	$O(n)$	$O(n^2)$	$O(n^2)$	Yes
Merge Sort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	Yes
Quick Sort (Det)	$O(n \log n)$	$O(n \log n)$	$O(n^2)$	No
Quick Sort (Rand)	$O(n \log n)$	$O(n \log n)$	$O(n^2)^*$	No
Heap Sort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	No
Counting Sort	$O(n + k)$	$O(n + k)$	$O(n + k)$	Yes
Radix Sort	$O(d(n + k))$	$O(d(n + k))$	$O(d(n + k))$	Yes
Bucket Sort	$O(n + k)$	$O(n + k)$	$O(n^2)$	Yes

** Expected case; worst case $O(n^2)$ is rare with randomization*
where: n = number of elements, k = range of input, d = number of digits

2. Space Complexity Analysis

Algorithm	Best Case	Average Case	Worst Case	In-Place?
Bubble Sort	$O(1)$	$O(1)$	$O(1)$	Yes
Selection Sort	$O(1)$	$O(1)$	$O(1)$	Yes
Insertion Sort	$O(1)$	$O(1)$	$O(1)$	Yes
Merge Sort	$O(n)$	$O(n)$	$O(n)$	No
Quick Sort (Det)	$O(\log n)$	$O(\log n)$	$O(n)$	Yes*
Quick Sort (Rand)	$O(\log n)$	$O(\log n)$	$O(n)$	Yes*
Heap Sort	$O(1)$	$O(1)$	$O(1)$	Yes
Counting Sort	$O(k)$	$O(k)$	$O(k)$	No
Radix Sort	$O(n + k)$	$O(n + k)$	$O(n + k)$	No
Bucket Sort	$O(n + k)$	$O(n + k)$	$O(n + k)$	No

** Requires $O(\log n)$ to $O(n)$ stack space for recursion.*

3. Key Observations

3.1. Comparison-Based Sorts (Lower bound: $\Omega(n \log n)$)

- Bubble, Selection, Insertion Sort: $O(n^2)$ — inefficient for large datasets.
- Merge, Heap Sort: $O(n \log n)$ — guaranteed performance, suitable for large data.
- Quick Sort: $O(n \log n)$ average — fastest in practice with small constants.

3.2. Non-Comparison Sorts (Can beat $O(n \log n)$)

- Counting Sort: Linear $O(n + k)$ when $k = O(n)$.
- Radix Sort: $O(d(n + k))$ — efficient for fixed-length integers.
- Bucket Sort: $O(n + k)$ average — best for uniformly distributed data.

3.3. Space–Time Tradeoffs

- In-place ($O(1)$): Bubble, Selection, Insertion, Heap, Quick Sort.
- Extra space ($O(n)$): Merge, Counting, Radix, Bucket Sort.
- Quick Sort uses $O(\log n)$ stack space but sorts in-place.

3.4. Stability

- Stable: Bubble, Insertion, Merge, Counting, Radix, Bucket.
- Unstable: Selection, Quick Sort, Heap Sort.
- Stability matters when sorting by multiple keys.

4. Detailed Algorithm Analysis

4.1. Bubble Sort

Time: Best $O(n)$ — Average $O(n^2)$ — Worst $O(n^2)$
Space: $O(1)$
Repeatedly swaps adjacent elements if in wrong order. Stable but inefficient.

4.2. Selection Sort

Time: Best $O(n^2)$ — Average $O(n^2)$ — Worst $O(n^2)$
Space: $O(1)$
Finds minimum and places it at beginning. Not stable, minimal swaps.

4.3. Insertion Sort

Time: Best $O(n)$ — Average $O(n^2)$ — Worst $O(n^2)$
Space: $O(1)$
Builds sorted list one element at a time. Stable, good for small/nigh-sorted inputs.

4.4. Merge Sort

Time: $O(n \log n)$ (all cases)
Space: $O(n)$
Divide and conquer. Stable and predictable, ideal for linked lists.

4.5. Quick Sort (Deterministic)

Time: Best/Average $O(n \log n)$ — Worst $O(n^2)$
Space: $O(\log n)$–$O(n)$
Partition-based. Worst on sorted data. Fast in practice.

4.6. Quick Sort (Randomized)

Time: Expected $O(n \log n)$ — Worst rare $O(n^2)$
Space: $O(\log n)$–$O(n)$
Random pivot minimizes worst-case chance. Best average performer.

4.7. Heap Sort

Time: $O(n \log n)$ in all cases
Space: $O(1)$
Uses heap. Guaranteed $O(n \log n)$, but unstable.

4.8. Counting Sort

Time: $O(n + k)$
Space: $O(k)$
Counts occurrences. Stable and linear when $k = O(n)$.

4.9. Radix Sort

Time: $O(d(n + k))$
Space: $O(n + k)$
Sorts by digits using a stable sort. Stable and efficient.

4.10. Bucket Sort

Time: Avg $O(n + k)$ — Worst $O(n^2)$
Space: $O(n + k)$
Distributes into buckets, sorts individually. Excellent for uniform data.

5. Algorithm Selection Guide

5.1. Based on Input Size

- Small ($n < 100$): Insertion Sort
- Medium ($100 < n < 10^5$): Randomized Quick Sort
- Large ($n > 10^5$): Merge or Quick Sort (Rand)
- Very Large ($n > 10^6$): Radix or Counting Sort

5.2. Based on Data Type

- Integers (limited range): Counting Sort
- Integers (any range): Radix or Quick Sort
- Floating-point: Bucket or Merge Sort
- General comparison: Merge or Quick Sort

5.3. Based on Requirements

- Stability: Merge, Counting, Radix, Bucket
- In-place: Heap, Quick Sort
- Worst-case guarantee: Merge or Heap Sort
- Best average: Quick Sort (Randomized)

5.4. Special Cases

- Nearly sorted: Insertion or Bubble Sort
- Many duplicates: Counting Sort
- Linked list: Merge Sort
- External sorting: Merge Sort

6. Complexity Class Hierarchy

$$
\begin{aligned}
O(1) &- \text{Constant (not applicable for sorting)} \\
O(\log n) &- \text{Logarithmic (not applicable)} \\
O(n) &- \text{Linear (Counting Sort when } k = O(1)) \\
O(n \log n) &- \text{Optimal for comparison sorts (Merge, Heap, Quick avg)} \\
O(n^2) &- \text{Quadratic (Bubble, Selection, Insertion, Quick worst)} \\
O(n^3), O(2^n) &- \text{Not practical for sorting.}
\end{aligned}
$$

Lower Bound Theorem:
Any comparison-based sorting algorithm requires $\Omega(n \log n)$ comparisons in the worst case. Non-comparison sorts (Counting, Radix, Bucket) can achieve linear time $O(n)$.

— END OF THEORETICAL ANALYSIS —