Bubble Sort: A Brute Force Approach

Sumanth

June 18, 2025

Abstract

Bubble Sort is a simple **brute force** comparison-based sorting algorithm that repeatedly swaps adjacent elements if they are in the wrong order until the list is sorted. Known for its educational value, it is stable, in-place, and adaptive with optimizations. This document details the algorithm's formulation, pseudocode, complexity analysis, practical applications, and implementation considerations, including early termination optimization and its role as a foundational sorting algorithm, as noted by the contributor.

1 Introduction

Bubble Sort is a fundamental **comparison-based** sorting algorithm that sorts an array by repeatedly comparing and swapping adjacent elements if they are in the wrong order. Named for the way larger elements "bubble" to the end of the array, it is a **brute force** approach due to its straightforward pairwise comparisons. Despite its inefficiency for large datasets, Bubble Sort is widely used in education to teach sorting concepts, algorithm tracing, and debugging, as highlighted by the contributor.

2 Problem Formulation

Given an array A of n elements, arrange the elements in non-decreasing order. The output is the sorted array A, with elements satisfying $A[i] \leq A[i+1]$ for all $i \in \{1,2,\ldots,n-1\}$.

2.1 Key Idea

The algorithm performs multiple passes over the array, comparing adjacent elements and swapping them if they are out of order. After each pass, the largest unsorted element is placed at its correct position at the end of the array. An optimization checks if any swaps occurred in a pass; if none, the array is sorted, allowing early termination.

3 Bubble Sort Algorithm

The algorithm uses a nested loop structure with an early termination optimization. Below is the pseudocode:

Algorithm 1 Bubble Sort Algorithm

```
Input: Array $A[1 \dots n]$
Output: Sorted array $A$
for $i = 1$ to $n - 1$ do
    swapped $\leftarrow$ false
    for $j = 1$ to $n - i$ do
        if $A[j] > A[j+1]$ then
            Swap $A[j]$ and $A[j+1]$
            swapped $\leftarrow$ true
        end if
    end for
    if swapped = false then
        break                                                    ▷ Early termination
    end if
end for
Return $A$
```

3.1 Stability

Bubble Sort is **stable**, preserving the relative order of equal elements, as it only swaps adjacent elements when strictly necessary ($A[j] > A[j+1]$).

3.2 Early Termination

The `swapped` flag enables the algorithm to terminate early if the array is already sorted, achieving $O(n)$ time complexity in the best case.

4 Complexity Analysis

- **Time Complexity**:
  - Best Case: $O(n)$, when the array is already sorted (one pass with no swaps).
  - Average Case: $O(n^2)$, due to pairwise comparisons across multiple passes.
  - Worst Case: $O(n^2)$, when the array is sorted in reverse order, requiring maximum swaps.
- **Space Complexity**: $O(1)$, as it is an in-place algorithm, using only a constant amount of extra memory.

5 Practical Applications

Bubble Sort is used in:

- **Education**: Teaching sorting algorithms, algorithm tracing, and debugging.
- **Small Datasets**: Sorting tiny arrays where simplicity outweighs performance.

- **Algorithm Analysis**: Serving as a baseline for comparing more advanced sorting algorithms.

6 Example

Given the array $A = [64, 34, 25, 12, 22]$, Bubble Sort proceeds as follows:

Pass	Array State				
Initial	64	34	25	12	22
Pass 1	34	25	12	22	64
Pass 2	25	12	22	34	64
Pass 3	12	22	25	34	64
Pass 4	12	22	25	34	64

After Pass 4, no swaps occur, and the algorithm terminates with the sorted array $[12, 22, 25, 34, 64]$.

7 Limitations and Extensions

- **Limitations**:
  - Inefficient for large datasets due to $O(n^2)$ complexity.
  - Outperformed by advanced algorithms like Quick Sort or Merge Sort for practical use.
- **Extensions**:
  - **Cocktail Shaker Sort**: A bidirectional variant that sorts from both ends, reducing passes for certain inputs.
  - **Adaptive Variants**: Enhance adaptivity by tracking the last swap position to reduce unnecessary comparisons.
  - **Hybrid Approaches**: Combine with other algorithms for small subarrays in larger sorting tasks.

8 Implementation Considerations

- **Early Termination**: Use a flag to detect sorted arrays, achieving $O(n)$ best-case performance.
- **Stability**: Ensure swaps only occur when strictly necessary to maintain stability.
- **Numeric Stability**: Handle large or floating-point numbers with appropriate data types to avoid overflow or precision issues.
- **Debugging Aid**: Implement with verbose output for educational purposes to trace comparisons and swaps.
- **Optimization**: Track the last swap position to reduce the inner loop's range in subsequent passes.

9 References

- GeeksForGeeks: Bubble Sort
- Wikipedia: Bubble Sort