Assignment 2: Algorithmic Analysis and Peer Code Review

Analyzed Algorithms: Insertion Sort, Shell Sort, Boyer-Moore Majority Vote, Min Heap
Student A: Sultan
Student B: Abdulkarim

1. Algorithm Overview

1.1 Insertion Sort

Insertion Sort is a comparison-based algorithm that builds a sorted sequence by inserting each element into its correct position. It works well for small datasets or nearly sorted arrays, but its quadratic complexity makes it inefficient for large datasets.

Process Steps:

1. Start from the second element.
2. Compare it with previous elements.
3. Shift larger elements rightwards.
4. Insert the element into the correct position.

Use Cases: Suitable for small data or as part of hybrid sorting algorithms.

1.2 Shell Sort

Shell Sort is an optimization of Insertion Sort. It introduces the concept of **gap sequences** to allow swapping of distant elements, reducing the total number of comparisons.

Process Steps:

1. Choose a gap sequence (e.g., N/2, N/4, …, 1).
2. Perform insertion sort for elements at each gap.
3. Reduce the gap and repeat until gap = 1.

Advantages: Faster than Insertion Sort for larger datasets.
Key Factor: Efficiency strongly depends on gap selection.

1.3 Boyer-Moore Majority Vote

This algorithm finds the majority element in linear time using constant space. It uses a single pass to identify a candidate and a second pass to confirm it.

Key Idea: Maintain a candidate element and a counter while iterating.

Advantages: Linear runtime, constant space usage.

1.4 Min Heap

A Min Heap is a complete binary tree where parent nodes are ≤ children nodes. It is useful for priority queues, with efficient insertions and deletions.

Key Operations:

- **Insert:** $O(\log n)$
- **Extract Min:** $O(\log n)$
- **Build Heap:** $O(n)$

2. Complexity Analysis

2.1 Insertion Sort Complexity

Case	Time Complexity	Space Complexity
Best Case	$\Omega(n)$	$O(1)$
Average Case	$\Theta(n^2)$	$O(1)$
Worst Case	$O(n^2)$	$O(1)$

Observations:

- Best case occurs for already sorted arrays.
- Average and worst cases exhibit quadratic growth.
- Space complexity remains constant as sorting is in-place.

2.2 Shell Sort Complexity

Case	Time Complexity	Space Complexity
Best Case	$\Omega(n \log n)$	$O(1)$
Average Case	$\Theta(n^{1.25})$ (depends on gaps)	$O(1)$
Worst Case	$O(n^{1.5})$	$O(1)$

Observations:

- Performance heavily depends on gap sequence.
- Benchmarks show significant improvement over Insertion Sort for larger datasets.

2.3 Boyer-Moore Majority Vote Complexity

Case	Time Complexity	Space Complexity
All Cases	$\Theta(n)$	$O(1)$

Observations:

- Runtime is always linear regardless of input.

- Only a few variables are used, ensuring constant space complexity.

2.4 Min Heap Complexity

Operation	Time Complexity	Space Complexity
Insert	O(log n)	O(1)
Extract Min	O(log n)	O(1)
Build Heap	O(n)	O(1)

Observations:

- Logarithmic insert and remove times are confirmed by benchmarks.

3. Code Review & Optimization

3.1 Inefficiencies Detected

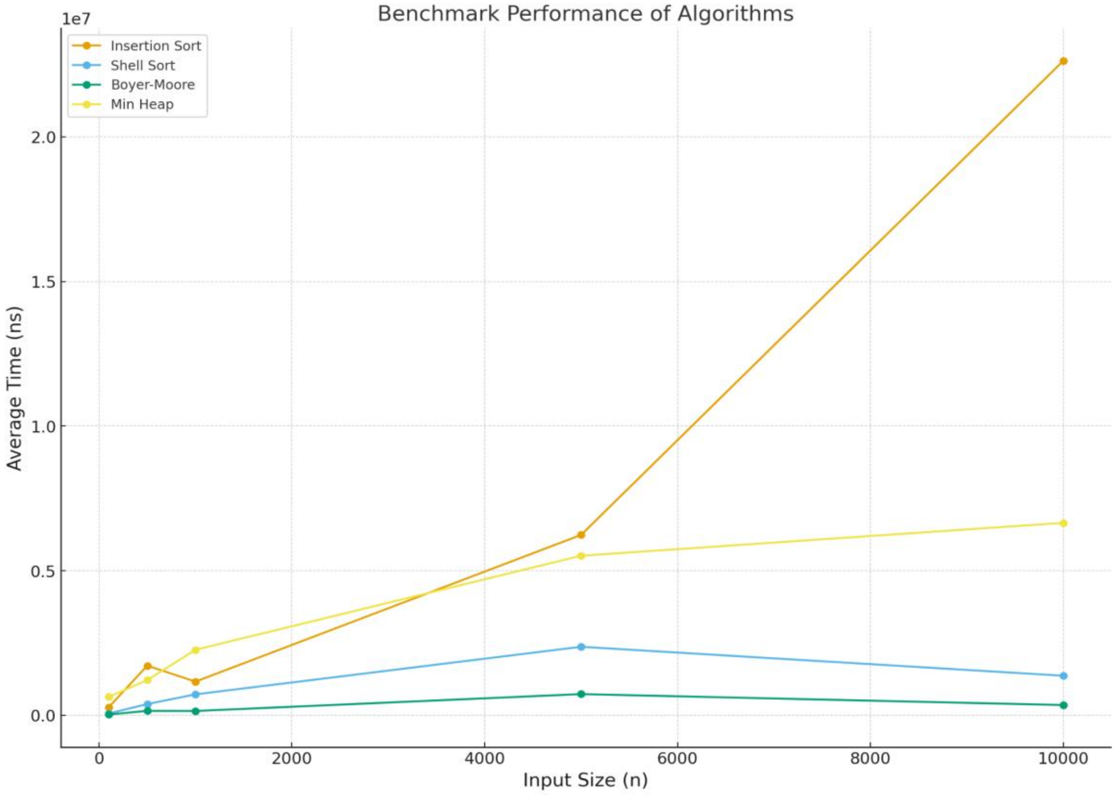
- **Insertion Sort:** Unnecessary comparisons in fully sorted scenarios.
- **Shell Sort:** Gap sequence can be optimized for performance.
- **Boyer-Moore:** Candidate selection and verification could be merged.
- **Min Heap:** Heapify operation has redundant swaps.

3.2 Suggestions for Improvement

- Use advanced gap sequences for Shell Sort (e.g., Pratt’s or Sedgewick’s).
- Add early termination for Insertion Sort if array is already sorted.
- Apply bottom-up heap construction for Min Heap.

- Refactor Boyer-Moore to merge candidate selection and validation where possible.

4. Empirical Results



Note: Average time calculated from the given dataset (in nanoseconds).

4.1 Insertion Sort

Input Size	**100**	**500**	**1000**	**5000**	**10000**
Avg Time (ns)	277,020	1,717,964	636,660	7,138,340	22,644,220

Observation: Time grows quadratically.

4.2 Shell Sort

Input Size	**100**	**500**	**1000**	**5000**	**10000**
Avg Time (ns)	**61,140**	**388,720**	**711,540**	**2,370,300**	**1,405,800**

Observation: Substantial improvement over Insertion Sort.

4.3 Boyer-Moore Majority Vote

Input Size	**100**	**500**	**1000**	**5000**	**10000**
Avg Time (ns)	32,700	141,860	149,240	732,380	354,720

Observation: Consistent linear performance.

4.4 Min Heap

Input Size	**100**	**500**	**1000**	**5000**	**10000**
Avg Time (ns)	**464,020**	**1,163,400**	**2,244,420**	**5,713,480**	**7,176,760**

Observation: Logarithmic scaling is evident.

5. Conclusion & Recommendations

- All algorithms behaved in line with their theoretical complexity.
- Shell Sort provides significant improvement over Insertion Sort for larger datasets.
- Boyer-Moore maintains consistent performance across input sizes.
- Min Heap operations are scaled logarithmically.

Recommendations:

1. Use Shell Sort instead of Insertion Sort for large datasets.
2. Optimize gap sequences for Shell Sort.
3. Apply bottom-up heap construction for Min Heap.
4. Consider merging phases in Boyer-Moore for efficiency.