Peer Analysis Report – Shell Sort (Knuth Sequence)

1. Introduction

This report presents a peer analysis of the Shell Sort implementation using the Knuth gap sequence. The purpose is to evaluate its theoretical complexity, memory usage, inefficiencies, and potential optimizations, and to compare it against Heap Sort (my partner's algorithm). The analysis is based on both theoretical background and empirical benchmark results collected with array sizes ranging from 100 to 100,000 under various distributions.

2. Theoretical Analysis of Shell Sort

Worst Case – $O(n^2)$: When the gap sequence fails to reduce the problem efficiently, Shell Sort degrades to quadratic time.

Best Case – $\Omega(n \log n)$: With partially sorted or favorable distributions, Shell Sort performs close to n log n.

Average Case – $\Theta(n^{(3/2)})$: Empirically and in theory, Shell Sort with Knuth gaps often shows about n^1.5 comparisons.

3. Memory Usage

Shell Sort is an in-place algorithm: memory complexity O(1). Only a few auxiliary variables are needed. Cache performance is decent due to sequential subarray access.

4. Empirical Results (Benchmarks)

Example for N=100:

trial=0 n=100 time=1ms comps=671 swaps=376 accesses=2107 trial=1 n=100 time=0ms comps=755 swaps=466 accesses=2371 trial=2 n=100 time=0ms comps=762 swaps=468 accesses=2382

Observations: Small inputs often report 0ms due to coarse measurement. Larger inputs show clear growth in comparisons and array accesses.

5. Identified Inefficiencies

1. Unnecessary Swaps 2. Redundant Array Accesses 3. Fixed Knuth Gaps 4. Timing Precision Issue

6. Suggested Optimizations

• Gap Sequence Improvements (e.g., Sedgewick, Tokuda) • Adaptive Swapping • Precision Timing (microseconds) • Vectorization possibilities

7. Comparison with Heap Sort

Aspect	Shell Sort (Knuth)	Heap Sort
Worst-case time	$O(n^2)$	O(n log n)
Average-case time	Θ(n^1.5)	Θ(n log n)
Best-case time	Ω(n log n)	Ω(n log n)
Memory usage	O(1), in-place	O(1), in-place
Cache behavior	Better (subarray locality)	Worse (tree jumps)
Predictability	Varies (input dependent)	Deterministic O(n log n)

8. Conclusion

Shell Sort (Knuth) shows practical efficiency but has theoretical drawbacks compared to Heap Sort. It is in-place, but inefficiencies such as redundant swaps and fixed gap limitations reduce its competitiveness for very large inputs. Shell Sort remains a useful teaching algorithm but Heap Sort or Quick Sort are preferable for production. Optimizations like better gap sequences and microsecond timing improve implementation quality.