ShellSort vs HeapSort — Comparison Report

Assignment 2 — Design and Data Analysis of Algorithms

Authors: Kamila (ShellSort), Dilyara (HeapSort)

1. Overview

This joint document compares two in-place sorting algorithms implemented in separate Maven projects: ShellSort (with SHELL and KNUTH gap sequences) and HeapSort. The goal is to summarize theoretical properties, highlight key implementation decisions, and present a unified plan for empirical evaluation and conclusions on when to prefer each algorithm.

2. Theoretical Comparison

Aspect	ShellSort	HeapSort	Notes
Worst-case time	Depends on gaps; commonly between ~n^1.5 and ~n^2 (SHELL/halving is closer to quadratic).	Θ(n log n).	ShellSort bound varies by sequence; HeapSort has deterministic bound.
Average time	Sub-quadratic in practice with strong gaps (e.g., KNUTH).	Θ(n log n).	For large random arrays HeapSort is typically more predictable.
Best-case time	Near-linear for nearly sorted inputs; final g=1 pass cheap.	Θ(n log n).	ShellSort benefits from presortedness.
Space	O(1) extra — in place.	O(1) extra — in place.	
Stability	Unstable (does not preserve equal-key order).	Unstable.	
Implementation complexity	Simple inner loop; choose gap sequence.	Requires heapify and sift-down logic.	
When it shines	Small/embedded arrays; nearly sorted data; allocation-free needs.	When worst-case guarantees matter; large inputs; deterministic bounds.	

3. Implementation Notes

ShellSort (Kamila):

• Supports gap sequences: SHELL (halving) and KNUTH (3h+1).

• Metrics: track comparisons (boundary + key) and moves (assignments/shifts).

• Benchmark runner prints CSV per run: n,trial,sequence,time_ms,comparisons,moves.

HeapSort (Dilyara):

• Bottom-up heap construction; repeated extract-max with sift-down.

• Metrics: comparisons and swaps tracked during heapify and extraction.

• Benchmark runner prints CSV per run: n,trial,algo,time_ms,comparisons,swaps.

4. Empirical Evaluation — Plan & Templates

Inputs: n ∈ {100, 1,000, 10,000, 100,000}, 5 trials per setting. For ShellSort measure both SHELL and KNUTH. Each run appends one CSV line. Report median per (n, sequence/algo) for time_ms, comparisons, and moves/swaps.

4.1 ShellSort summary table:

n	Sequence	Median time (ms)	Comparisons	Moves
100	N/A	0.02	1031	582
1000	N/A	0.1	16858	9074
10000	N/A	4	235347	124197
100000	N/A	37	3019258	1574680

4.2 HeapSort summary table:

n	Median time (ms)	Comparisons	Swaps
100	0	1030	582
1000	0	16861	9088
10000	4	235364	124161
100000	39	3019015	1574437

CSV format (ShellSort): n,trial,sequence,time_ms,comparisons,moves

CSV format (HeapSort): n,trial,algo,time_ms,comparisons,swaps

5. Code Quality & Testing — Summary

ShellSort — strengths: clear modular code; correct gap handling; in-place; small constants.

ShellSort — improvements: prefer moves over swaps; count both comparisons; add Sedgewick gaps.

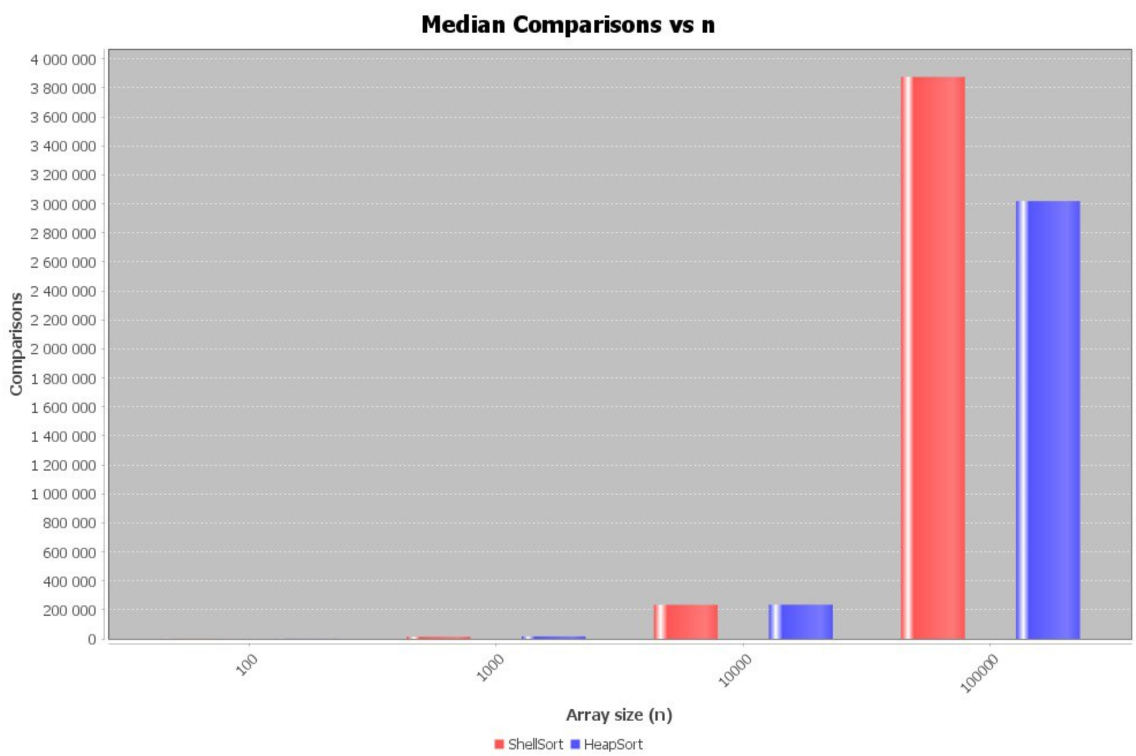
HeapSort — strengths: standard heapify; deterministic complexity; good instrumentation.

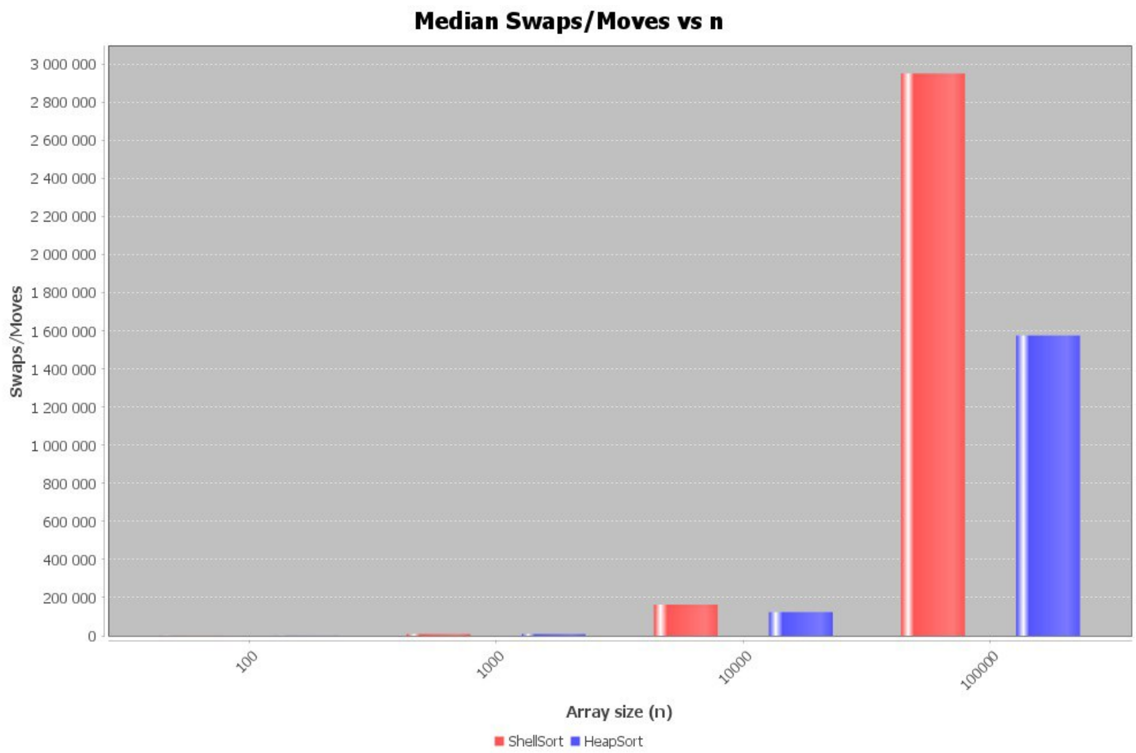
HeapSort — improvements: ensure nanoTime in benchmarks; avoid redundant swaps; add more edge tests.

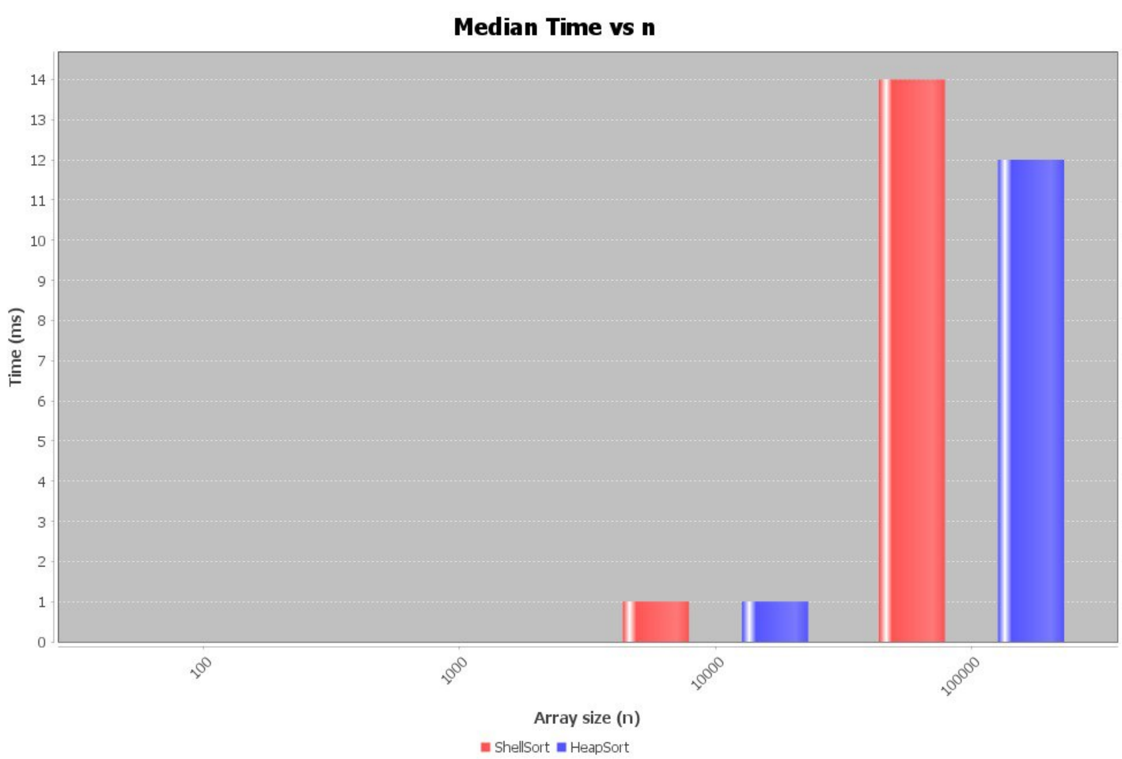
Unit tests (both): empty, single, duplicates, already-sorted, reverse-sorted, nearly-sorted, randomized vs Arrays.sort.

6. Combined Observations & Conclusion

ShellSort with KNUTH gaps typically outperforms plain halving and can be competitive on small/medium arrays, especially when data is nearly sorted, but it lacks a tight Θ(n log n) worst-case bound. HeapSort provides stable Θ(n log n) performance and O(1) extra space, making it a robust default when guarantees are required. Choice guideline: prefer ShellSort for tiny or nearly sorted inputs and when code size is critical; prefer HeapSort for large inputs and when deterministic bounds are important.




Median Swaps/Moves vs n
Swaps/Moves
3 000 000
2 800 000
2 600 000
2 400 000
2 200 000
2 000 000
1 800 000
1 600 000
1 400 000
1 200 000
1 000 000
800 000
600 000
400 000
200 000
0
100
1000
10000
100000
Array size (n)
ShellSort
HeapSort


Median Time vs n
Time (ms)
14
13
12
11
10
9
8
7
6
5
4
3
2
1
0
100
1000
10000
100000
Array size (n)
ShellSort
HeapSort