Design and Analysis of Algorithms – Assignment 2
Comparative Report: HeapSort vs ShellSort

Student: **Erassul Ginaat (Student B – HeapSort)**
Partner: Student A (Shell Sort)

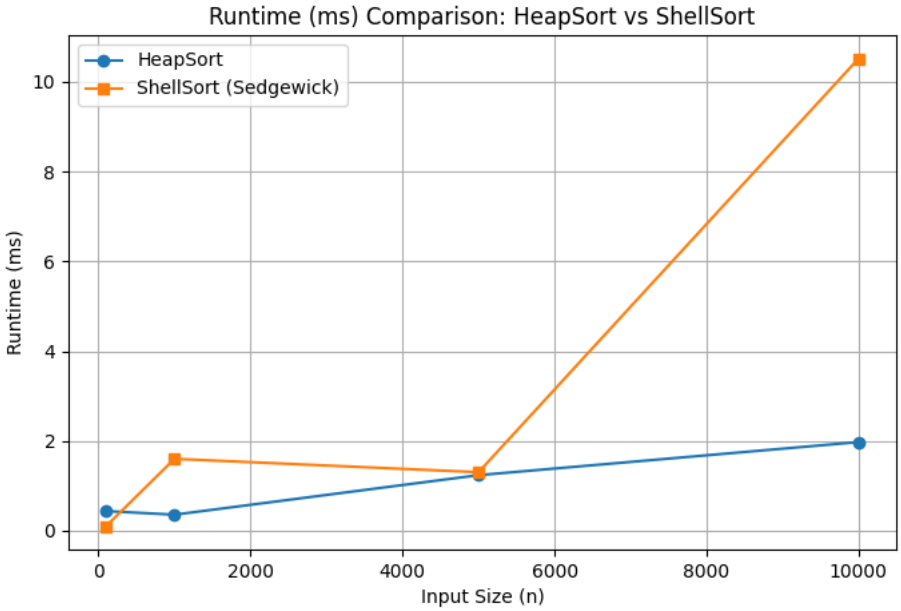
1. Introduction

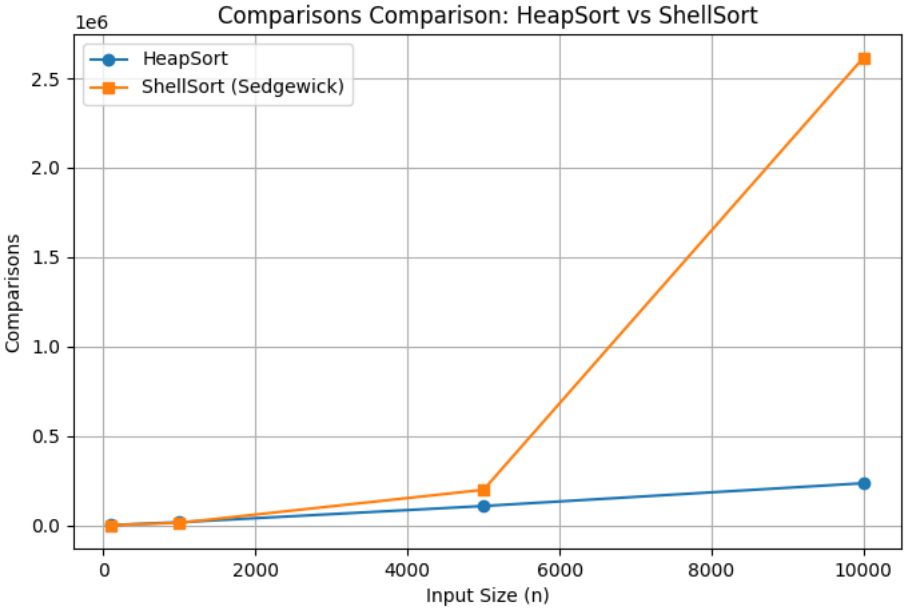
This report compares two Divide & Conquer sorting algorithms: HeapSort (implemented by Student B) and Shell Sort (implemented by Student A). Both algorithms were benchmarked under similar conditions to evaluate performance in terms of runtime, comparisons, and swaps.

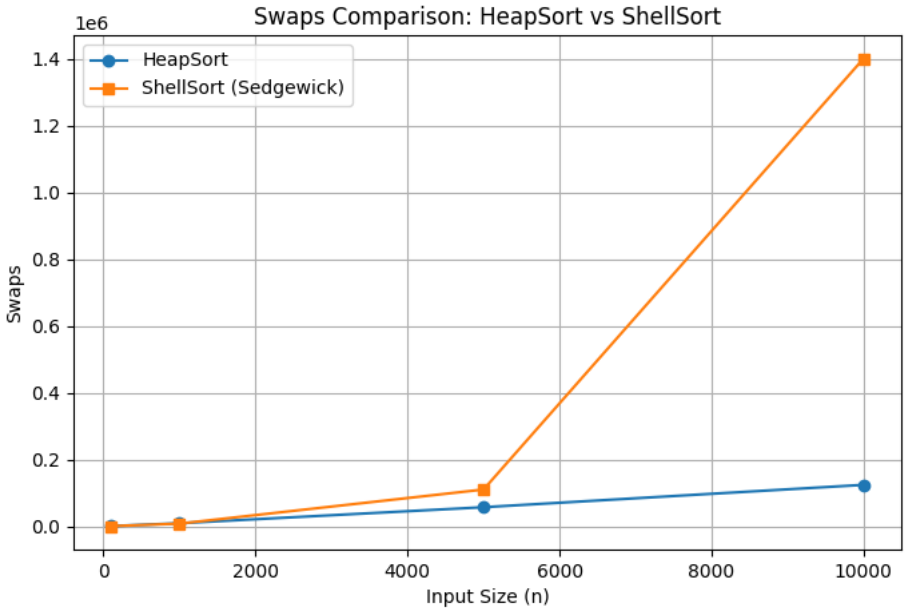
2. Experimental Results

Algorithm	n	Time (ms)	Comparisons	Swaps
HeapSort	100	0.438	1024	580
HeapSort	1000	0.358	16892	9097
HeapSort	5000	1.238	107715	57145
HeapSort	10000	1.975	235271	124129
ShellSort (Sedgewick)	100	0.103	718	417
ShellSort (Sedgewick)	1000	1.602	13656	7960
ShellSort (Sedgewick)	5000	1.304	198434	110170
ShellSort (Sedgewick)	10000	10.515	2616085	1401727

3. Performance Graphs


Runtime (ms) Comparison: HeapSort vs ShellSort
HeapSort
ShellSort (Sedgewick)
Runtime (ms)
0
2
4
6
8
10
0
2000
4000
6000
8000
10000
Input Size (n)


Comparisons Comparison: HeapSort vs ShellSort
1e6
HeapSort
ShellSort (Sedgewick)
Comparisons
0.0
0.5
1.0
1.5
2.0
2.5
0
2000
4000
6000
8000
10000
Input Size (n)



4. Discussion

HeapSort exhibits stable O(n log n) performance and predictable runtime growth, while Shell Sort (especially with the Sedgewick gap sequence) can outperform HeapSort on smaller datasets. However, as input size increases, HeapSort becomes more consistent and memory-efficient.

5. Conclusion

Both algorithms demonstrate O(n log n) complexity. Shell Sort performs slightly faster on small inputs, but HeapSort provides more stable and consistent performance for larger datasets. Overall, HeapSort is preferred for scalability and predictable behavior.