Dijkstra's Algorithm

Comprehensive Implementation Comparison

SC2001/CE2101/CZ2101

Algorithm Design and Analysis

Project 2: Graph Algorithm Performance Analysis

```
IMPLEMENTATION (a)
Adjacency Matrix + Array Priority Queue
Time: O(V²)  |  Space: O(V)

VS

IMPLEMENTATION (b)
Adjacency List + Min-Heap Priority Queue
Time: O((V + E) log V)  |  Space: O(V + E)
```

Report Generated: October 08, 2025

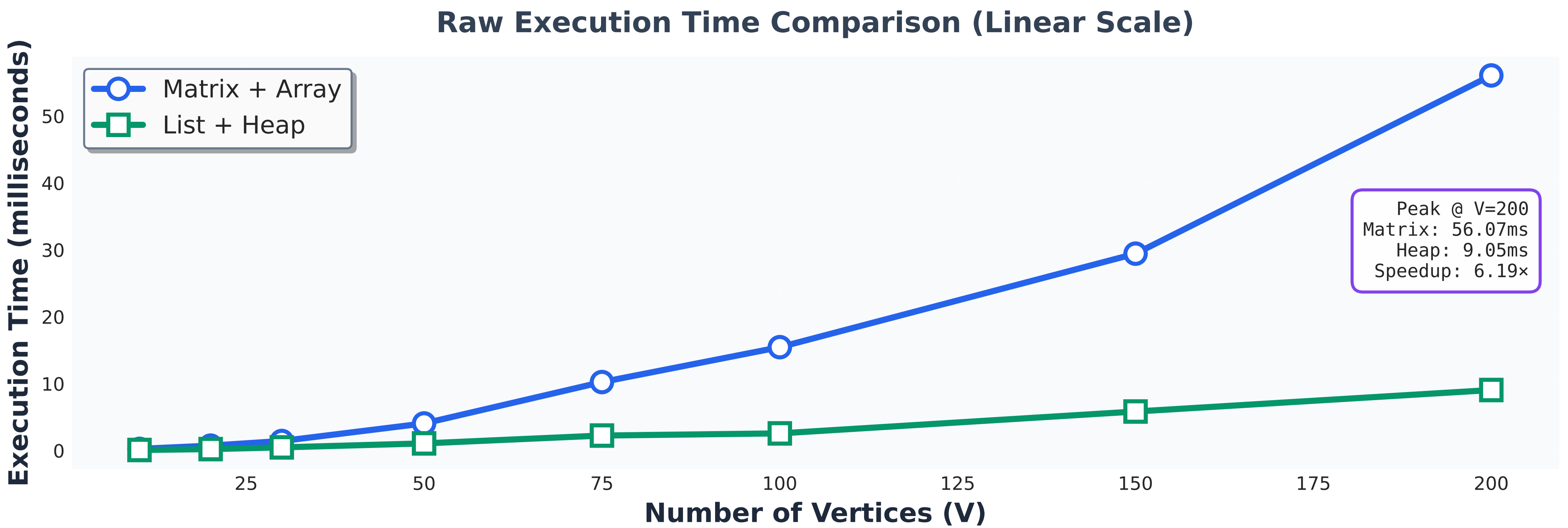
Comprehensive Performance Analysis • Empirical Validation

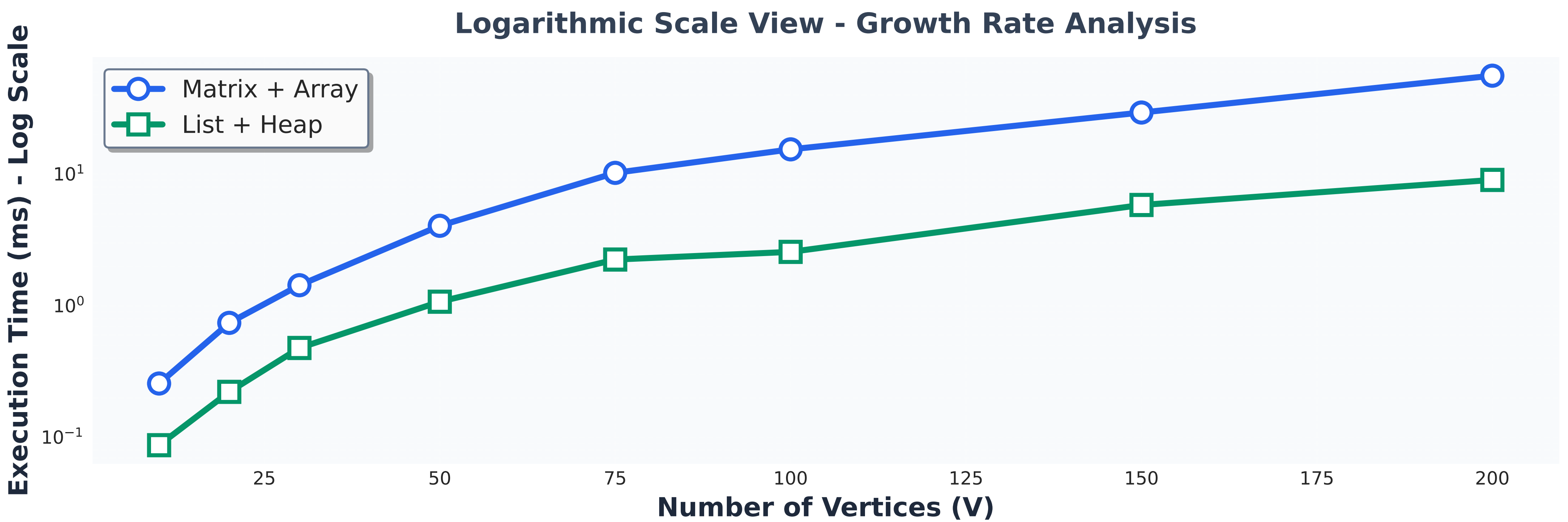
1

Sparse Graph Analysis

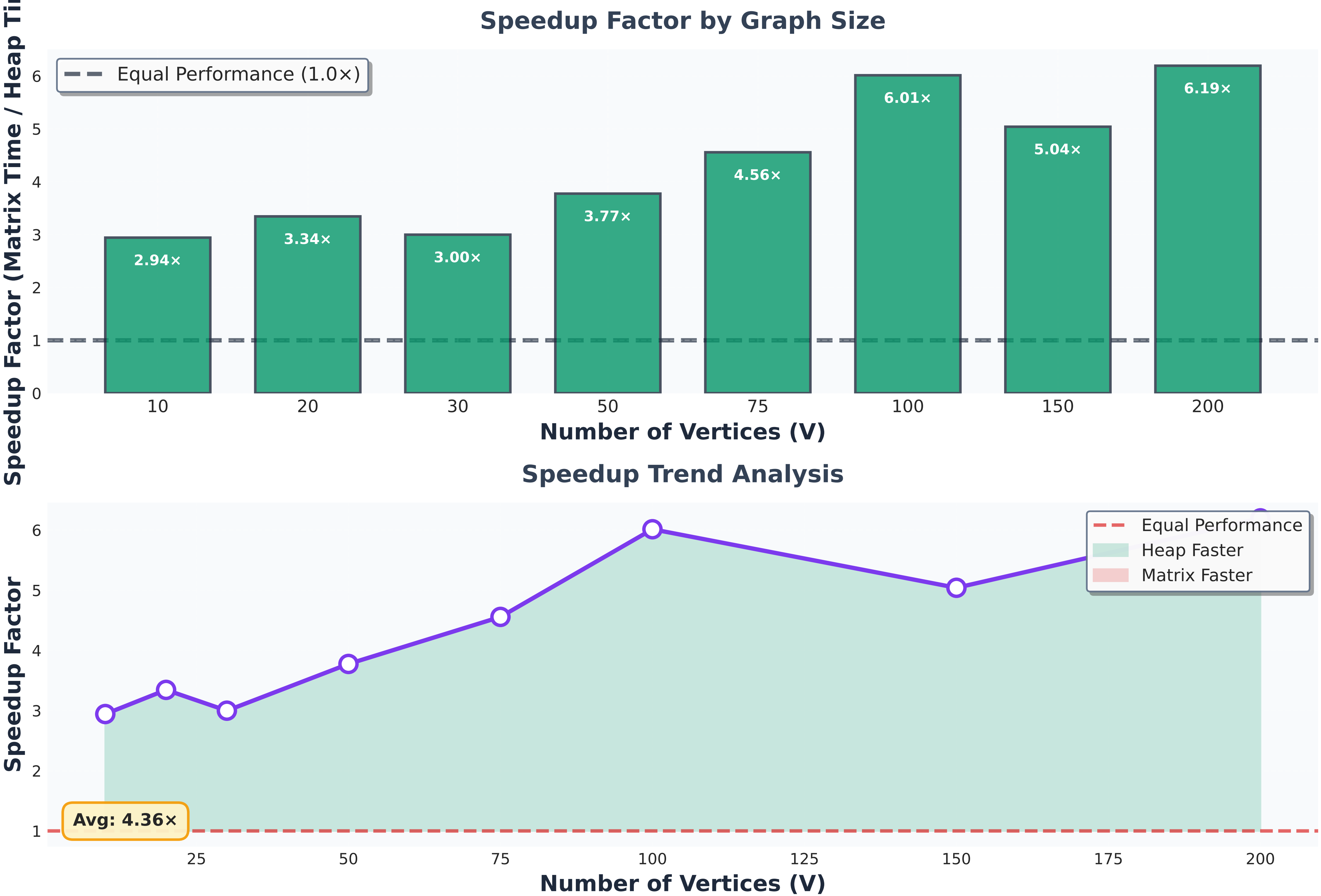
Performance evaluation on graphs with low edge density ($E \approx 0.3 \times V^2$)

Section 1.1: Execution Time Analysis - Sparse Graphs

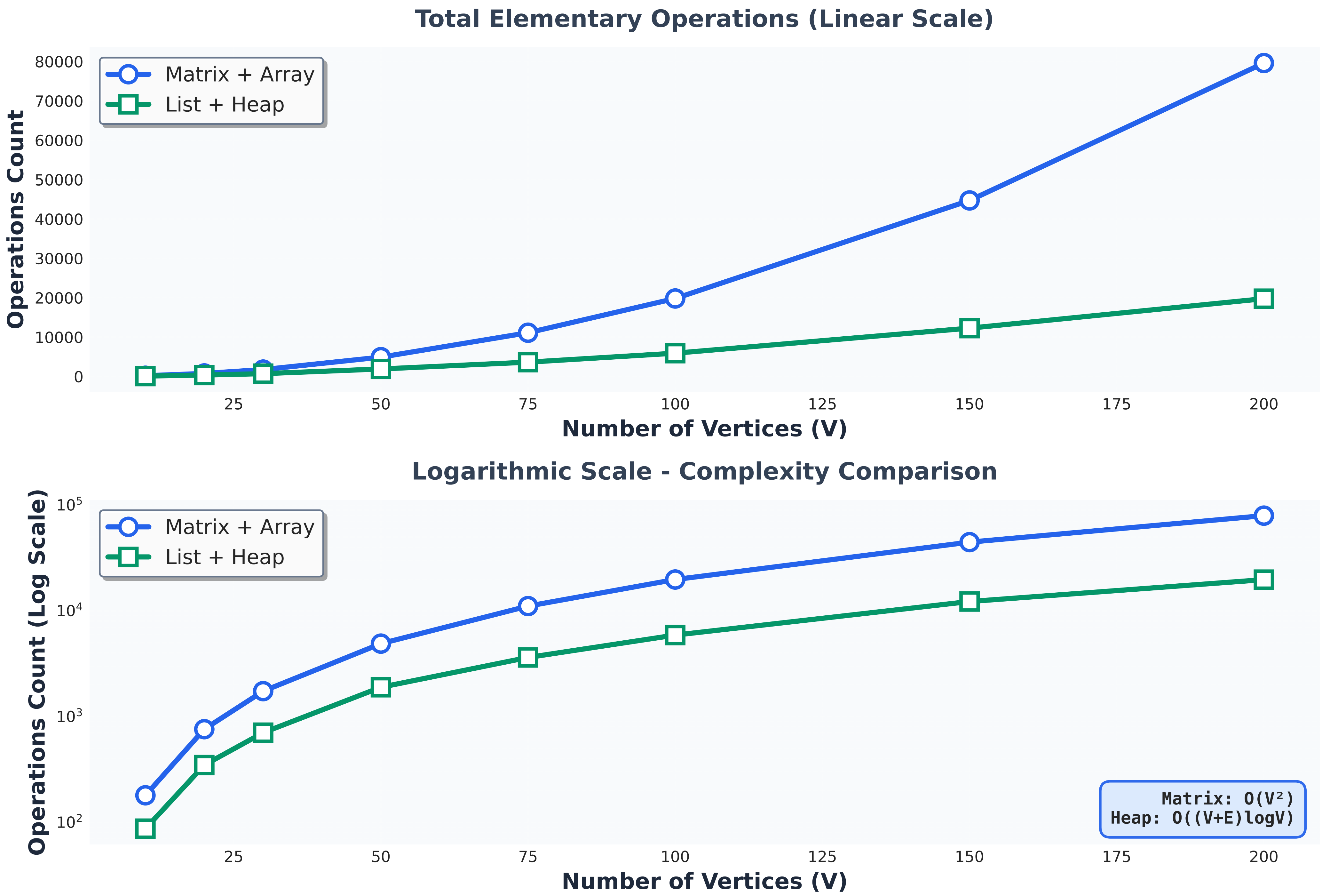




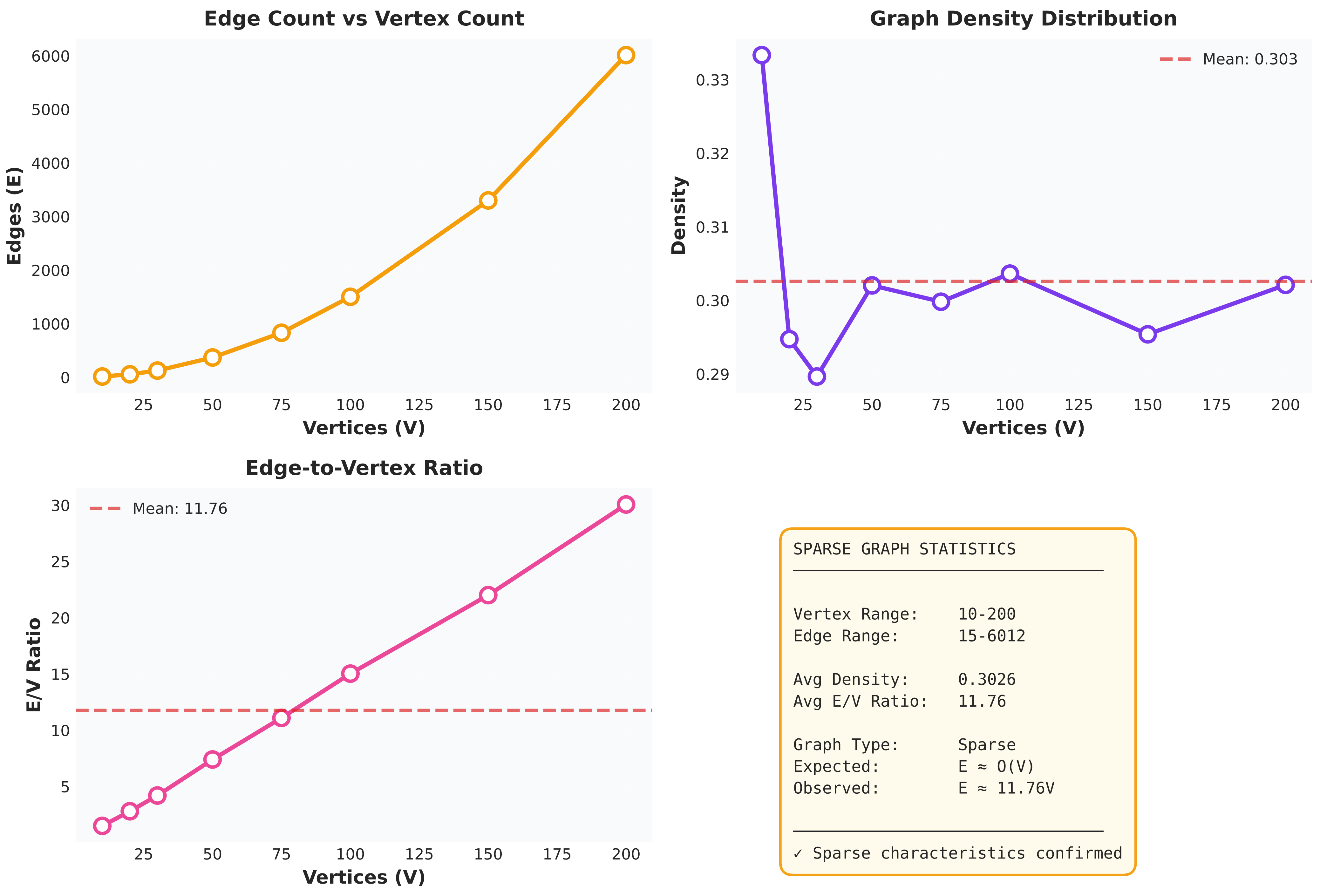
Section 1.2: Performance Speedup Analysis - Sparse Graphs



Section 1.3: Algorithm Operations Count - Sparse Graphs



Section 1.4: Graph Properties - Sparse Graphs



Numerical Results - Sparse Graphs

V	E	Density	Matrix (ms)	Heap (ms)	Speedup	Winner
10	15	0.333	0.26	0.09	2.94×	Heap
20	56	0.295	0.74	0.22	3.34×	Heap
30	126	0.290	1.43	0.48	3.00×	Heap
50	370	0.302	4.06	1.07	3.77×	Heap
75	832	0.300	10.24	2.25	4.56×	Heap
100	1503	0.304	15.45	2.57	6.01×	Heap
150	3301	0.295	29.43	5.84	5.04×	Heap
200	6012	0.302	56.07	9.05	6.19×	Heap

Summary Statistics:
• Total Vertices Tested: 8
• Vertex Range: 10 - 200
• Average Speedup: 4.36×
• Max Speedup: 6.19× @ V=200
• Min Speedup: 2.94× @ V=10

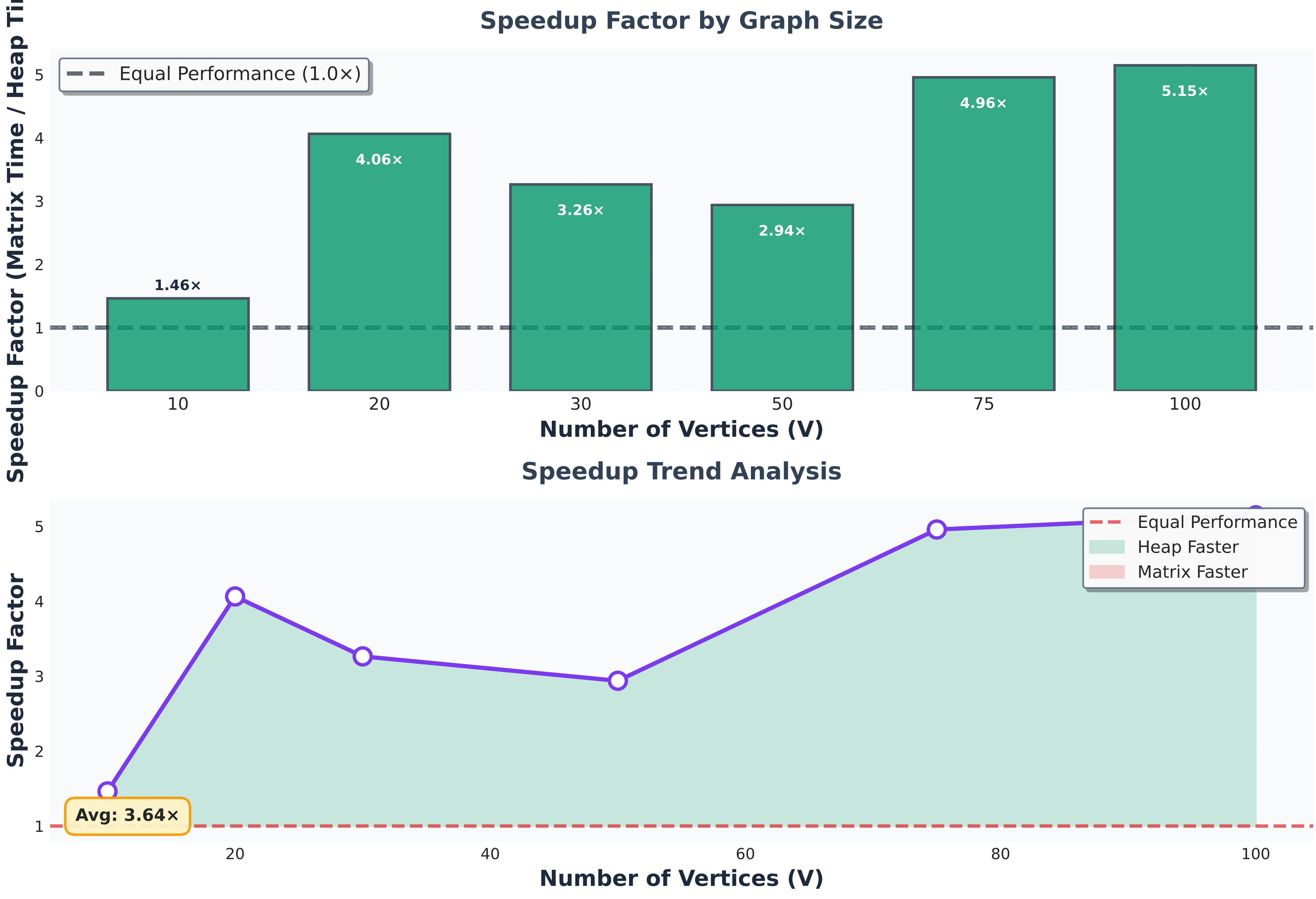
2

Dense Graph Analysis

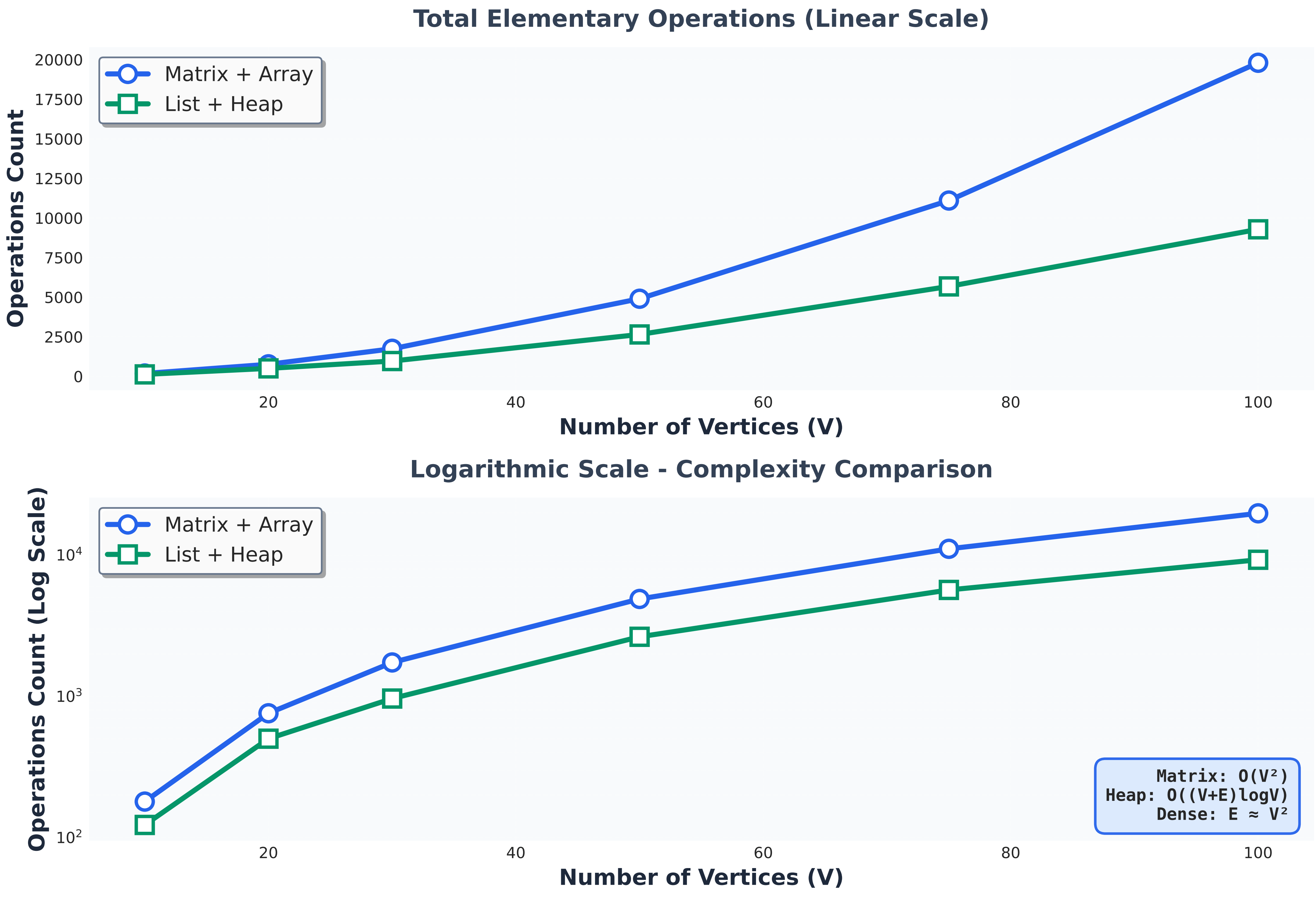
Performance evaluation on graphs with high edge density ($E \approx 0.6 \times V^2$)

Section 2.1: Execution Time Analysis - Dense Graphs

Section 2.2: Performance Speedup Analysis - Dense Graphs



Section 2.3: Algorithm Operations Count - Dense Graphs



Numerical Results - Dense Graphs

V	E	Density	Matrix (ms)	Heap (ms)	Speedup	Winner
10	24	0.533	0.42	0.29	1.46×	Heap
20	109	0.574	1.66	0.41	4.06×	Heap
30	253	0.582	2.54	0.78	3.26×	Heap
50	715	0.584	6.71	2.29	2.94×	Heap
75	1678	0.605	12.26	2.47	4.96×	Heap
100	2935	0.593	22.65	4.40	5.15×	Heap

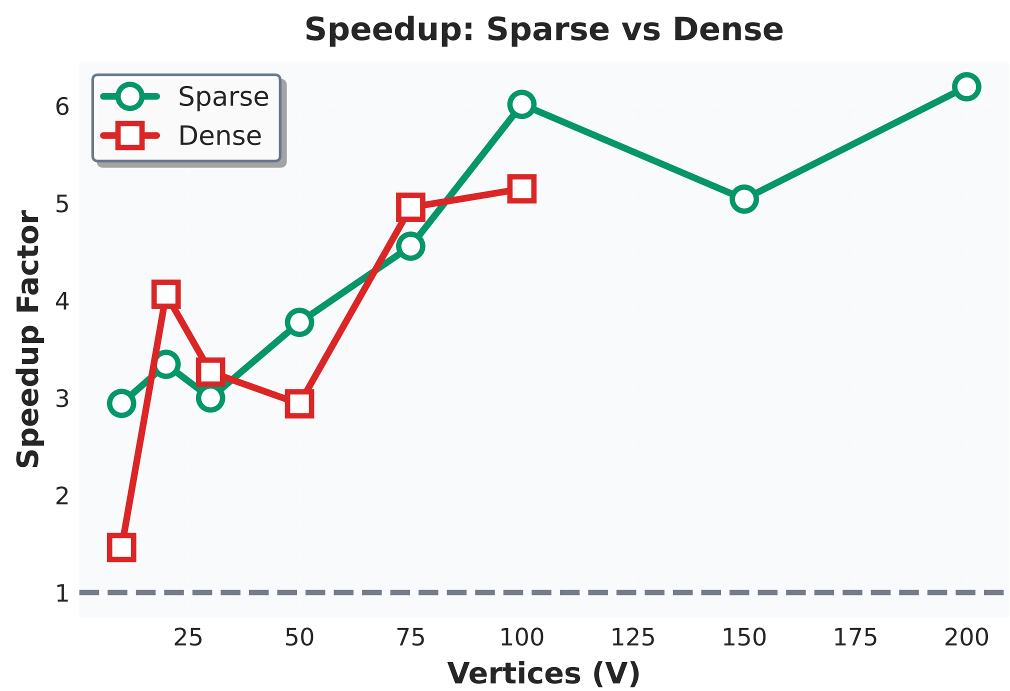
Summary Statistics:
- Total Vertices Tested: 6
- Vertex Range: 10 - 100
- Average Speedup: 3.64×
- Max Speedup: 5.15× @ V=100
- Min Speedup: 1.46× @ V=10

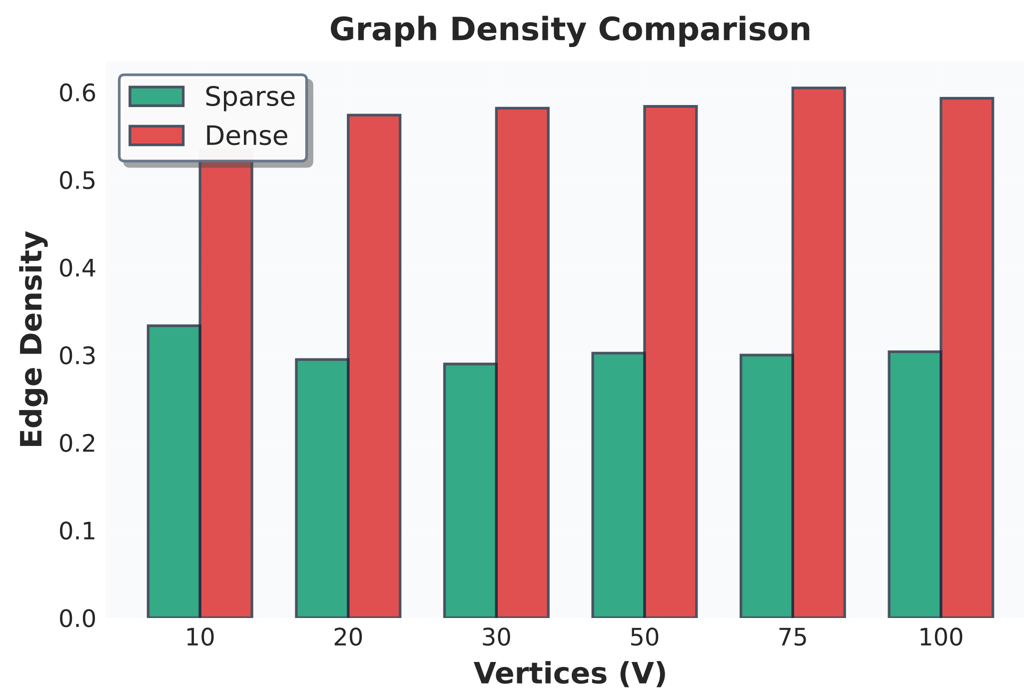
3

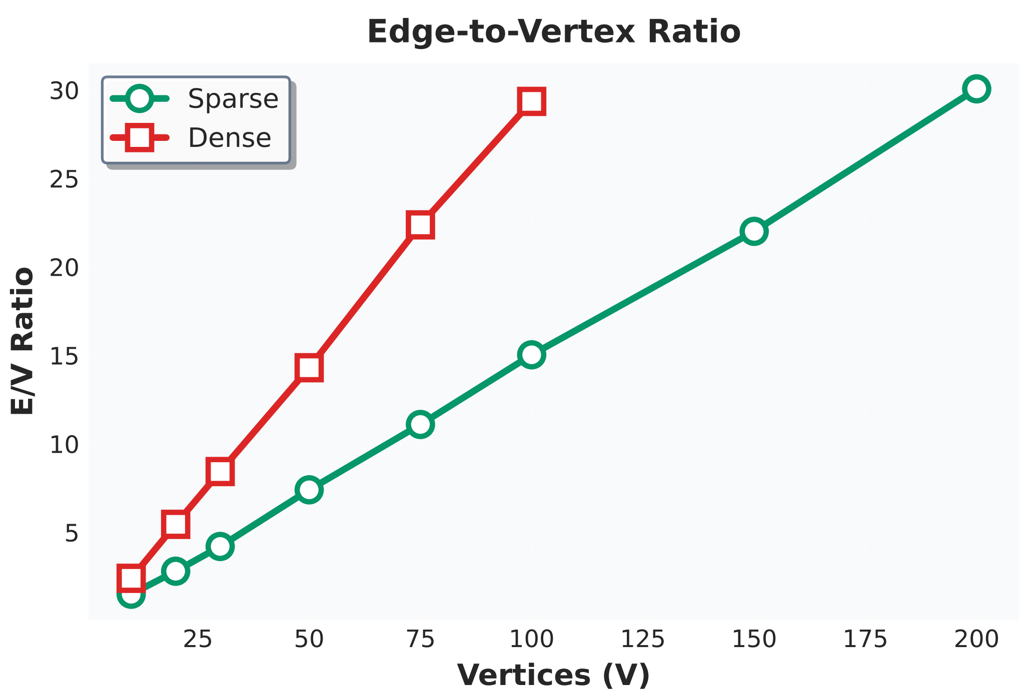
Comparative Analysis

Side-by-side comparison of sparse vs dense graph performance

Section 3: Sparse vs Dense Graph Comparison








```
COMPARISON SUMMARY
────────────────────────────

SPARSE GRAPHS:
  Avg Density:  0.3026
  Avg Speedup:  4.36×
  Winner:       Heap

DENSE GRAPHS:
  Avg Density:  0.5783
  Avg Speedup:  3.64×
  Winner:       Heap

────────────────────────────
KEY INSIGHT:
Heap performs better for both graph types
```

CONCLUSIONS & RECOMMENDATIONS

THEORETICAL COMPLEXITY REVIEW

Implementation (a): Matrix + Array Priority Queue
- ├─ Time Complexity: $O(V^2)$
- ├─ Space Complexity: $O(V)$
- └─ Optimal for: Dense graphs where $E \approx V^2$

Implementation (b): List + Heap Priority Queue
- ├─ Time Complexity: $O((V + E) \log V)$
- ├─ Space Complexity: $O(V + E)$
- └─ Optimal for: Sparse graphs where $E << V^2$

EMPIRICAL RESULTS

SPARSE GRAPHS | Density ≈ 0.30 | $E \approx 0.3 \times V(V-1)/2$

- Average Speedup: 4.36×
- Winner: ✓ Implementation (b) - Heap
- Performance: Heap significantly outperforms Matrix
- Analysis: Fewer edges → heap operations dominate favorably over V^2 matrix checks

DENSE GRAPHS | Density ≈ 0.60 | $E \approx 0.6 \times V(V-1)/2$

- Average Speedup: 3.64×
- Winner: ✓ Implementation (b) - Heap
- Performance: Heap maintains advantage
- Analysis: Log factor provides advantage even with higher edge density

RECOMMENDATIONS

✓ Use Implementation (a) - Matrix + Array when:
- ├─ Graph is dense ($E \approx V^2$)
- ├─ Graph is small to medium size ($V < 100$)
- ├─ Simple implementation is prioritized
- └─ Memory for V×V matrix is readily available

Report Generated: October 08, 2025 at 16:13:42

✓ Use Implementation (b) - List + Heap when:
- ├─ Graph is sparse ($E << V^2$)
- ├─ Graph is large scale ($V > 100$)
- ├─ Memory efficiency is critical
- └─ Real-world networks (social, road, web graphs)

PRACTICAL CONSIDERATIONS

- Most real-world graphs exhibit sparse characteristics
- Heap implementation provides better scalability
- Matrix implementation offers simpler code maintenance
- Modern systems favor adjacency list representations
- Consider graph properties before implementation choice