IMPASSABLE GATE - RESULT ANALYSIS

```
================================================================================
IMPASSABLE GATE - RESULT ANALYSIS
================================================================================

This analysis examines the time and space complexity of three search algorithms
based on results tested from 16 puzzles.

================================================================================
THEORETICAL TIME COMPLEXITY REFERENCE
================================================================================

For a puzzle with:
- n = number of pieces
- k = number of steps (solution depth)
- q = number of open squares on the board
- w = width of search (for Algorithm 3)

ALGORITHM 1 (Naive Approach):
  Time Complexity: O((4n)^k)
  - Branching factor of 4n (n pieces x 4 directions per piece)
  - Exponential in solution depth k

ALGORITHM 2 (Duplicate Checking with Radix Tree):
  Time Complexity: O(n^q)
  - Bounded board: polynomial, as each square can have at most one piece,
    limiting total states to n^q
  - Duplicate detection with radix tree avoids re-exploring seen states

ALGORITHM 3 (Iterative Width / Novelty Checking):
  Time Complexity: O(nq^w) where w is the search width
  - Best case: w << n, giving much better complexity than O(n^q) from Algo2
  - Worst case: w = n, giving O(n^(q+1)) due to re-exploring the puzzle w times
  - Novelty checking reduces exploration space

================================================================================
1. TIME COMPLEXITY ANALYSIS
================================================================================

1.1 OBSERVED TIME COMPLEXITY
------------------------------------

From the performance graphs (time_complexity.png)
and statistical summary (RESULTS_SUMMARY.txt):

ALGORITHM 1 (No Duplicate Detection):
- Generated nodes: min=1, max=3,043,088, avg=284,566
- Expanded nodes: min=2, max=564,372, avg=52,493
- Only completed 11/16 puzzles (simple puzzles only)
- Shows EXPONENTIAL GROWTH with in time complexity
- Failed to solve complex puzzles (impassable1, impassable2, impassable3,
  and harder capability puzzles)

ALGORITHM 2 (Radix Tree Duplicate Detection):
- Generated nodes: min=2, max=11,218,541, avg=708,516
- Expanded nodes: min=2, max=10,352,979, avg=653,439
- Completed all 16/16 puzzles
- Shows POLYNOMIAL growth
- Duplicate detection efficiency: 45.6% average (54.4% of attempts are duplicates)

ALGORITHM 3 (Iterative Width):
- Generated nodes: min=2, max=157,621, avg=12,167
- Expanded nodes: min=2, max=155,254, avg=11,917
- Completed all 16/16 puzzles
- Shows REDUCED GROWTH compared to both Algorithm 1 and 2
```

- Most efficient: average 12,167 generated nodes vs 708,516 (Algo2) vs 284,566 (Algo1)
- For impassable3: only 68.77 seconds vs 1011 seconds (Algo2)
- Finds solutions at lower widths (IW(1) to IW(4) for most puzzles)

1.2 COMPARISON WITH THEORETICAL TIME COMPLEXITY

ALGORITHM 1 - Theoretical: O((4n)^k) where n = pieces, k = steps
- Observed: EXPONENTIAL growth confirmed
- Data shows exponential increase in generated nodes with puzzle complexity
- Performance degrades: could not solve puzzles with k > 13 steps

ALGORITHM 2 - Theoretical: O(n^q) where n = pieces, q = open squares
- Observed: POLYNOMIAL growth confirmed
- Strong correlation between theoretical complexity and actual nodes generated
- Successfully handles large state spaces (11M nodes for impassable3: 8^14 ~= 4.4M)
- Duplicate detection prevents timeout
- Avoids re-exploring seen states, avoiding immediate prior state

ALGORITHM 3 - Theoretical: O(nq^w) where w = width of search
- Observed: SIGNIFICANTLY BETTER than both algorithms
- Novelty checking dramatically reduces exploration space

CONCLUSION ON TIME COMPLEXITY:
The data clearly shows (matching theoretical predictions from reference section):
1. Algorithm 1: Exponential O((4n)^k) - matches theory
   * Average 284K nodes for simple puzzles only
2. Algorithm 2: Polynomial O(n^q) - matches theory
   * Average 709K nodes, handles all puzzles
3. Algorithm 3: O(nq^w) where w << n - matches theory
   * Average only 12K nodes, better than Algo2

The time_complexity.png graph (Pieces x Steps x Empty x Width) shows the clearest correlation, with all three algorithms following their theoretical complexity curves.

2. SPACE COMPLEXITY ANALYSIS

2.1 OBSERVED SPACE COMPLEXITY GROWTH

From space complexity graphs (space_algorithm_*.png) and statistical summary:

ALGORITHM 1 (No Duplicate Detection):
- Expanded nodes: min=2, max=564,372, avg=52,493
- Auxiliary memory: 0 MB (no duplicate tracking structures)
- Total space: Queue size only ~= expanded nodes
- Space grows EXPONENTIALLY

ALGORITHM 2 (Radix Tree):
- Expanded nodes: min=2, max=10,352,979, avg=653,439
- Auxiliary memory: min=0, max=358.06 MB, avg=22.61 MB
- Total space: Queue + Radix Tree ~= expanded nodes + stored states
- Space grows POLYNOMIALLY bounded by n^q

ALGORITHM 3 (Iterative Width):
- Expanded nodes: min=2, max=155,254, avg=11,917
- Auxiliary memory: 0 MB (trees freed after each width iteration)
- Total space: Current width's queue and trees only
- Space is bounded by current width.
- Trees freed between width iterations keeps memory low

IMPASSABLE GATE - RESULT ANALYSIS

2.2 DO ALGORITHMS 2 AND 3 DECREASE SPACE GROWTH RATE vs ALGORITHM 1?

From space_complexity.png graph:
- Algorithm 3 (green) consistently below Algorithm 2 (blue) and Algorithm 1 (red)
- For similar theoretical baseline complexity:
* Algo1 uses ~500K nodes (when it can solve but fails to solve complex as it runs out of memory)
  * Algo2 uses ~10M nodes for hard puzzles
  * Algo3 uses ~150K nodes maximum

Specific Example - impassable3 (8 pieces, 78 steps, 14 empty spaces):
- Algorithm 1: FAILED (failed to solve)
- Algorithm 2: 358 MB auxiliary + 10.3M expanded = ~375 MB total
- Algorithm 3: 0 MB auxiliary + 155K expanded = ~5 MB total
  -> Algorithm 3 uses LESS memory than Algorithm 2

==

3. SUMMARY AND CONCLUSIONS

==

TIME COMPLEXITY RESULTS:

- Algorithm 1: Exponential O((4n)^k) - confirmed by data
  * Only solved 11/16 puzzles (simple puzzles with small k)
  * Average 285K nodes for puzzles it could solve
- Algorithm 2: Polynomial O(n^q) - confirmed by data
  * Solved all 16/16 puzzles
  * Average 709K nodes, max 11.2M for impassable3
- Algorithm 3: Polynomial O(nq^w) where w << n - confirmed by data
  * Worst case when w=n: O(n^(q+1))
  * Solved all 16/16 puzzles at much lower widths
  * Average only 12K nodes
  * Example: puzzle11 actual (3,012) ~= theoretical (2,916) at w=2

-> Execution time: Algo3 (4.7s avg) << Algo2 (63.5s avg) << Algo1 (1.1s avg on simple, but timeout for complex puzzles)

SPACE COMPLEXITY RESULTS:
- Algorithm 1: Exponential O((4n)^k) - only solves 11/16 puzzles
  * Failed on all complex puzzles due to exponential memory timeout
- Algorithm 2: Polynomial O(n^q) - REDUCES growth rate
  * Reduces from EXPONENTIAL to POLYNOMIAL
  * Average 22.6 MB auxiliary memory, max 358 MB for impassable3
- Algorithm 3: Polynomial O(nq^w) where w << n
  * Reduces polynomial degree: O(n^q) -> O(nq^w) where w << n
  * Average ~0 MB auxiliary (trees freed between widths)
  * For impassable3: 5 MB vs Algo2's 375 MB

-> For hardest puzzle (n=8, q=14, k=78):
  * Algo1: O((32)^78) - INFEASIBLE (10^117 nodes)
  * Algo2: O(8^14) ~= 4.4M - FEASIBLE (11.2M actual nodes, 375 MB)
  * Algo3: O(8x14^w) - EFFICIENT (155K actual nodes, 5 MB)

BEST ALGORITHM: Algorithm 3 (Iterative Width)
- Fastest execution time (4.7s average vs 63.5s for Algo2)
- Lowest memory usage (~5 MB max vs 375 MB for Algo2)