Sliding Puzzle algorithm overview

Author : Samuel Cucicea
Student ID : W1873364
Course: 5SENG003W.2 Algorithms: Theory, Design, and Implementation

a) Choice of Data Structure and Algorithm

- **Algorithm Choice:**
- Choosing the Breadth-First Search (BFS) algorithm was a strategic decision. Its strength in identifying the shortest path in an unweighted graph directly aligns with the structure of the sliding puzzle. This adaptation was key to ensuring the algorithm could effectively navigate the puzzle's unique sliding cardinal directions.

- **Data Structure Utilization:**

2D Array for the Puzzle Board: The choice here was clear. A 2D array mirrors the grid layout naturally (cells,row,col ..etc) making it easier to manage navigation and update the puzzle state.

- **ArrayDeque for Queue Management:** BFS requires efficient queue management for its breadth-first exploration. ArrayDeque was ideal for this, thanks to its quick FIFO (First In, First Out) capabilities, providing O(1).

- **HashSet for Tracking Visited Nodes:** To avoid going over the same spot multiple times, a HashSet keeps track of which nodes we've already visited. This not only saves time but also boosts algorithm's performance by making sure every move counts and is not a redundant move.

b) Benchmark Example Run

Ran the algorithm on a 10x10 puzzle as a small benchmark test. The solution correctly identified the shortest path, demonstrating the algorithm's ability to navigate the puzzle rules effectively. The steps taken from 'S' to 'F' included sliding actions that were logically consistent with the puzzle's constraints, showcasing the practical application of the chosen algorithm and data structures.

The path shown represent a grid which rows and columns are counted from 0;

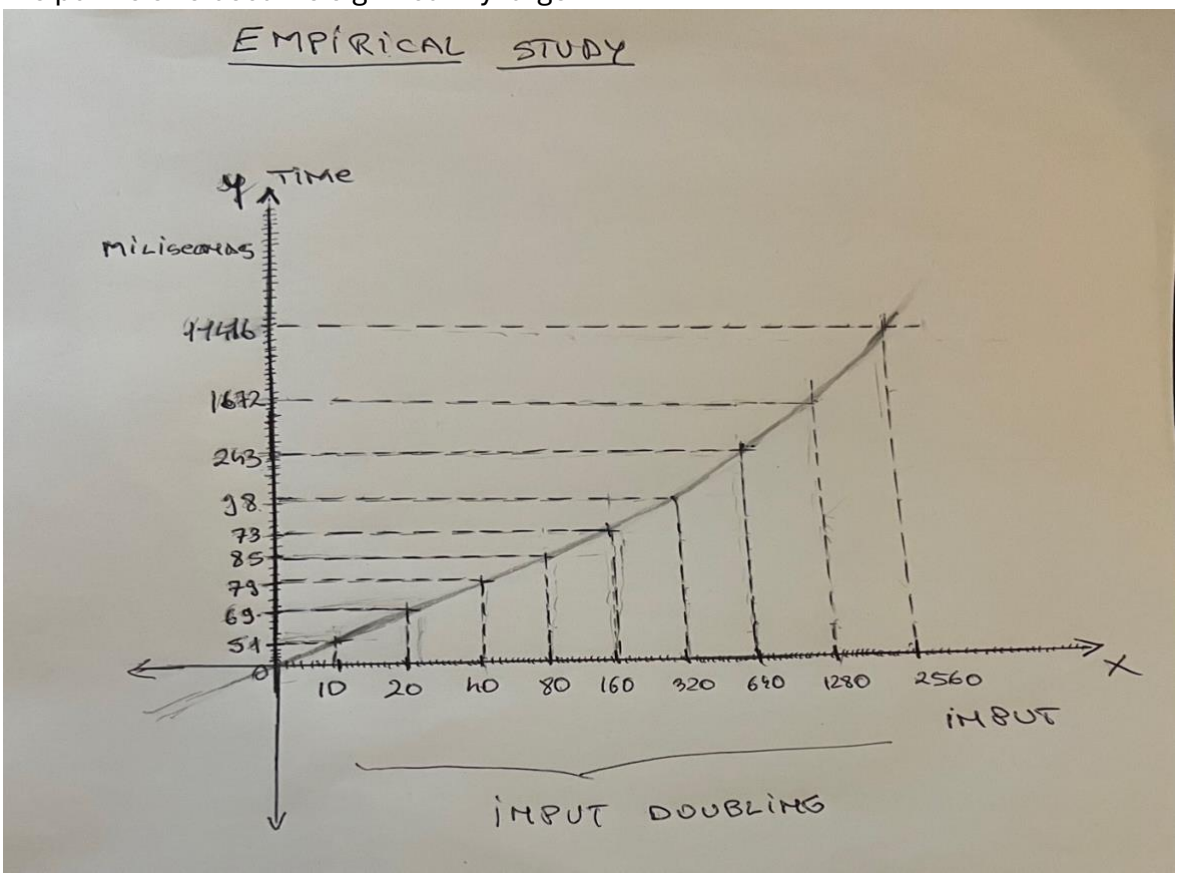
- **Example Path**
- 1. Move right to (9, 1)
- 2. Move up to (8, 1)
- 3. Move right to (8, 8)
- 4. Move up to (7, 8)
- 5. Move left to (7, 4)
- 6. Move down to (9, 4)
- 7. Move right to (9, 7)

- 8. Move up to (0, 7)
- 9. Move right to (0, 9)
- 10. Move down to (2, 9)
- 11. Move left to (2, 6)
- 12. Move up to (1, 6)
- 13. Move left to (1, 4)
- 14. Move up to (0, 4)
- 15. Move left to (0, 2)
- 16. Move down to (1, 2)
- Done!
- Time taken :45

- **Time Taken**: Time taken : 45

c) Performance Analysis

- **Empirical Study**: Conducted tests by doubling the puzzle size each time, observing how the execution time increased. Notably, the time growth was not strictly linear but showed a pattern suggesting more complexity, especially as the puzzle size became significantly larger.



Considerations:Base on the observed performance , the time complexity seem to be linearithmic, However, for extremely larger puzzles, the execution time is not purely linearithmic, suggesting for quadratic time with extremely big mazes.

Classification: The empirical data suggests a transition from linearithmic for smaller puzzles to potentially quadratic (O(n^2)) for extremely larger puzzles.