Approach- A* Search with Manhattan Distance Heuristic

- I had tried 3 different metrics.
  1. Hamming Distance – Counts the number of misplaced tiles.
  2. Manhattan Distance – Counts the distance between a tile and its correct location.
  3. Manhattan Distance with Linear Conflicts – Linear Conflicts means if any two tiles are in the same row or column and are blocking each others goal states. It means that it would take some extra steps to put them in the right place and normal Manhattan distance would not count this.
- I chose **Manhattan metric** as my final choice since it was the fastest. The Hamming distance metric was exploring too many nodes and hence slower. On the other hand, Linear Conflicts are time consuming to calculate for every node which made the algorithm slower.
- I then implemented the A* search algorithm. I used a Min-Heap data structure to store the nodes and at every step, it pops out the node with the lowest f(n) value (Cost). I used a set data structure to keep track of already visited nodes and a dictionary to store the parent relationships so that the path can be constructed.

Optimality

- We know that A* search finds optimal solution to problems as long as the heuristic is admissible.
- Manhattan distance can be used as a heuristic for this problem because it is a solution for the relaxed 15-puzzle problem in which we can place a tile at its correct place directly.
- I have used Manhattan distance as the heuristic here and Manhattan distance is an admissible metric because it never overestimates the cost of the path from any given node. For every point, it gives the distance from that point to its true position measured along the horizontal and vertical direction.
- Due to these conditions, the algorithm that I have used which is A* Search with Manhattan distance as heuristic will give the optimal solution.

Performance of Algorithm

TEST CASE	A* Search, Misplaced Tile Heuristic	A* Search, Manhattan Distance Heuristic	A* Search, Manhattan + Linear Conflicts Heuristic
INITIAL STATE 1	Nodes – 32 Time – 0.001s	Nodes – 32 Time – 0.001 s	Nodes – 305 Time – 0.018s
INITIAL STATE 2	-	Nodes – 111182 Time – 1.3035 s	Nodes – 201357 Time – 11.0495 s
INITIAL STATE 3	-	Nodes – 1363848 Time – 17.0265 s	Nodes – 1334220 Time – 75.8292 s
INITIAL STATE 4	-	Nodes – 890864 Time – 10.7184 s	Nodes – 1650520 Time – 92.7282 s

Note : These times are recorded on my local machine. Blank values indicate that it took too much time to run to be considered efficient.