Approach

- I have use the A-star search algorithm to search the state space for the solution.
- The heuristic used is sum of Manhattan distances of each block from the position of that block in goal state.

Optimizations

- Instead of re-calculating heuristic value after every move, I have made a function to just update the heuristic value. When a move is played, it affects only 2 position on the board, so we can just update the heuristic value based on the change in those 2 places rather than going through the entire board again. This is handled by the *‘getNewHeuristic’* method in my code.
- I have created a separate *‘parent’* dictionary which saves the parent of every explored node using hashing by converting NumPy array to a string which make it faster to access. It also avoids the burden to store parent as a part of the node. Because if we keep saving parents as a part of the nodes, then pushing and popping the nodes from priority queue becomes a heavy operation.
- I have combined the explored set and the minimum path cost mapping into one data structure using a dictionary. So the *‘explored’* dictionary in my code functions as the explored set as well as the container for all minimum path costs encountered till now. Since dictionary uses hashing, updating and accessing from it becomes very fast

Optimality

- A-star search always gives optimal answer if the heuristic used in admissible. In this case, I have used the sum of Manhattan distances of the blocks from their location in the goal state as the heuristic, which always underestimates the actual path cost, hence it is an admissible heuristic

- Since the heuristic used is admissible, A-star search will always find the optimal path.

Comparing runtime and number of nodes generated in different approaches

- Out of the 3 optimizations mentioned in slide-1, the most significant improvement in runtime was caused by the first one (updating the old heuristic after every move instead of re-calculating). This does not affect the number of nodes generated

	Re-calculating heuristic after every move	Updating heuristic after every move
initial_state1	Time: 0.0 seconds Nodes Generated: 31	Time: 0.0 seconds Nodes Generated: 31
initial_state2	Time: 6.0455 seconds Nodes Generated: 164900	Time: 1.8235 seconds Nodes Generated: 164900
initial_state3	Time: 28.6652 Nodes Generated: 775878	Time: 9.2355 seconds Nodes Generated: 775878
initial_state4	Time: 58.6325 seconds Nodes Generated: 1575611	Time: 18.9653 seconds Nodes Generated: 1575611