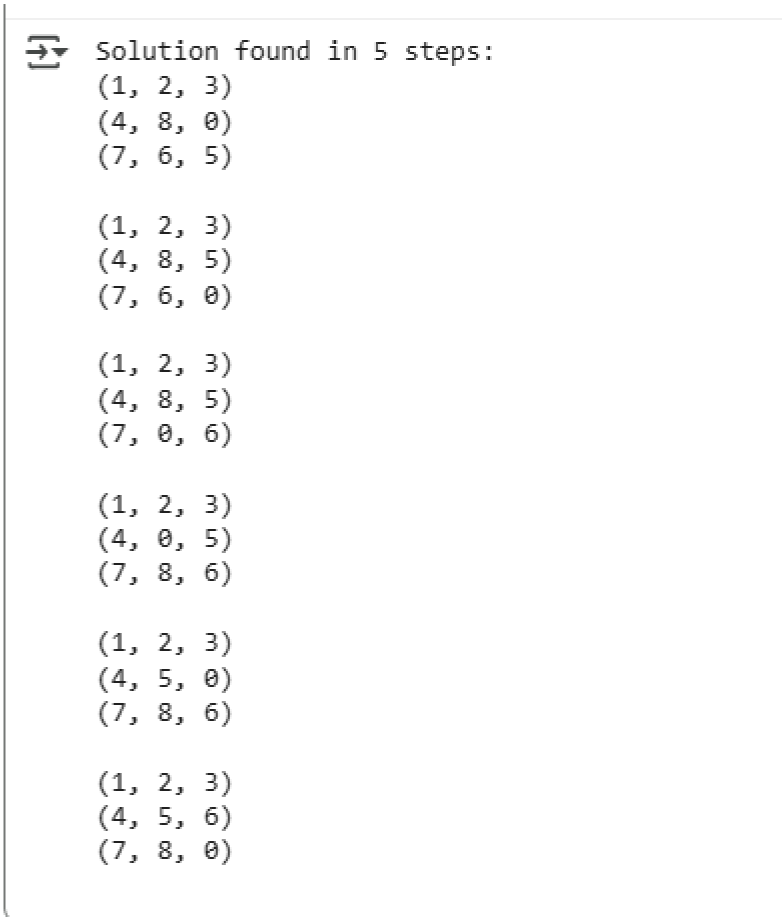
AI LAB 3

A* Algorithm for Misplaced Tiles

```
import heapq

# Define the goal state of the 8-puzzle
goal_state = (1, 2, 3, 4, 5, 6, 7, 8, 0)

# Directions for moving the blank space: (row, col) changes for [up, down,
left, right]
moves = [(-1, 0), (1, 0), (0, -1), (0, 1)]

# Function to calculate the number of misplaced tiles
def misplaced_tiles(state):
    return sum(1 for i in range(9) if state[i] != goal_state[i] and
state[i] != 0)

# Function to get the neighbors of the current state
def get_neighbors(state):
    zero_pos = state.index(0)
    row, col = divmod(zero_pos, 3)
    neighbors = []

    for move in moves:
        new_row, new_col = row + move[0], col + move[1]
        if 0 <= new_row < 3 and 0 <= new_col < 3:
            new_zero_pos = new_row * 3 + new_col
            # Swap the blank space with the adjacent tile
            new_state = list(state)
            new_state[zero_pos], new_state[new_zero_pos] =
new_state[new_zero_pos], new_state[zero_pos]
            neighbors.append((tuple(new_state), new_zero_pos))

    return neighbors

# A* algorithm for solving the 8-puzzle problem
def a_star(initial_state):
    # Priority queue for the A* search: stores tuples of (f, g, state,
parent_state)
    open_list = []
    heapq.heappush(open_list, (misplaced_tiles(initial_state), 0,
initial_state, None))

    # Set to keep track of visited states
    closed_set = set()

    # Dictionary to store the parent state for each state
    parent_map = {initial_state: None}

    while open_list:
```

```python
        f, g, current_state, parent_state = heapq.heappop(open_list)

        # If the goal state is reached
        if current_state == goal_state:
            solution = []
            while current_state is not None:
                solution.append(current_state)
                current_state = parent_map[current_state]
            return solution[::-1]  # Reverse to get the solution from
start to goal

        closed_set.add(current_state)

        # Get all possible neighbors of the current state
        for neighbor, _ in get_neighbors(current_state):
            if neighbor not in closed_set:
                g_new = g + 1
                h_new = misplaced_tiles(neighbor)
                f_new = g_new + h_new
                if all(neighbor != state[0] for state in open_list):  #
Only add to the open list if not already present
                    heapq.heappush(open_list, (f_new, g_new, neighbor,
current_state))
                    parent_map[neighbor] = current_state

    return None  # No solution found

# Function to print the solution in a readable format
def print_solution(solution):
    for state in solution:
        for i in range(0, 9, 3):
            print(state[i:i+3])
        print()

# Example of solving the 8-puzzle from an initial state
initial_state = (1, 2, 3, 4, 8, 0, 7, 6, 5)  # Example initial state
solution = a_star(initial_state)
if solution:
    print("Solution found in", len(solution) - 1, "steps:")
    print_solution(solution)
else:
    print("No solution found.")
```

OUTPUT



```
Solution found in 5 steps:
(1, 2, 3)
(4, 8, 0)
(7, 6, 5)

(1, 2, 3)
(4, 8, 5)
(7, 6, 0)

(1, 2, 3)
(4, 8, 5)
(7, 0, 6)

(1, 2, 3)
(4, 0, 5)
(7, 8, 6)

(1, 2, 3)
(4, 5, 0)
(7, 8, 6)

(1, 2, 3)
(4, 5, 6)
(7, 8, 0)
```