Artificial Intelligence
Assignment # 1
FA-23
3rd November, 2023
Mohammad Ali Jinnah University

1. DFS Implementation:

```
def dfs_solver(initial_state):
    stack = [initial_state]
    visited = set([initial_state.serialize()])
    prev_states = {initial_state.serialize(): None}
    actions = {initial_state.serialize(): None}

    while stack:
        current_state = stack.pop()
        i = 0

        if current_state.solved():
            moves = []
            print(i)
            i = i + 1
            while current_state:
                move = actions[current_state.serialize()]
                if move is not None:
                    moves.append(move)
                current_state = prev_states[current_state.serialize()]
            moves.reverse()
            return moves

        for direction in ["left", "right", "up", "down"]:
            new_state = current_state.move(direction)
            if new_state and new_state.serialize() not in visited:
                visited.add(new_state.serialize())
                stack.append(new_state)
                prev_states[new_state.serialize()] = current_state
                actions[new_state.serialize()] = direction

    return []
```

2. Manhattan Distance:

```
def manhattan_dist(state, target):

    return sum(
        abs(i // 3 - target[i] // 3) + abs(i % 3 - target[i] % 3)
        for i in range(9)
        if state[i] != target[i] and target[i] is not None
    )
```

3. Sum of Misplaced Tiles

```
def sum_of_misplaced_tiles(state, target):
    return sum(1 for i in range(9) if state[i] != target[i])
```

4. A* Implementation:

```
def astar_solver(initial_state, target_state):
    open_set = [(initial_state, 0)]  # Initialize the open set with the
initial state and cost
    g_costs = {initial_state: 0}     # Dictionary to keep track of the "g"
cost for each state
    actions = {initial_state: None}  # Dictionary to store actions taken
to reach each state

    while open_set:
        # Find the state with the lowest f-score (g + heuristic)
        current_state, _ = min(open_set, key=lambda x: g_costs[x[0]] +
heuristic(x[0], target_state))

        if current_state == target_state:
            # Reconstruct and return the list of moves
            moves = []
            while current_state:
                move = actions[current_state]
                if move is not None:
                    moves.append(move)
                current_state = current_state.get_previous_state()
            moves.reverse()
            return moves

        for direction in ["left", "right", "up", "down"]:
            new_state = current_state.move(direction)
            if new_state:
                tentative_g_cost = g_costs[current_state] + 1

                if new_state not in g_costs or tentative_g_cost <
g_costs[new_state]:
                    g_costs[new_state] = tentative_g_cost
                    open_set.append((new_state, tentative_g_cost))
                    actions[new_state] = direction

    return []
```

5. Comparison between both heuristics:

- The Manhattan Distance is usually more accurate and fast due to the spatial relationship of each tile.

- The Misplaced Tiles Heuristic is less accurate but much simpler than the Manhattan Distance.