Assignment 1

Student Name: Rizwan Amir

Student ID: SP21-BSCS-0052

Course: Artificial Intelligence

Course Instructor: Sir Asim Imdad Wagan

1) **DFS Implementation:**

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

    while queue:
        current_state = queue.pop()

        if current_state.solved():
            moves = []
            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())
                queue.append(new_state)
                prev_states[new_state.serialize()] = current_state
                actions[new_state.serialize()] = direction

    return []
```

2) Heuristic Code (Sum of Misplaced Tiles and Manhattan Distance of misplaced tiles):

```
def misplaced_tiles(board):
    correct_board = [[1, 2, 3], [4, 5, 6], [7, 8, None]]
    misplaced_tiles = 0
    n = len(board)
    for i in range(n):
        for j in range(n):
            if board[i][j] != correct_board[i][j]:
                misplaced_tiles += 1

    return misplaced_tiles
if __name__ == "__main__":
    incorrect_board = [[7, 1, 4], [5, 2, 3], [None, 6, 8]]
print("Total misplaced tiles are:", misplaced_tiles(incorrect_board))

def manhattan distance(board):
    manhattan_distance = 0
    correct_board = [[1, 2, 3], [4, 5, 6], [7, 8, -1]]
    n = len(board)
    for i in range(n):
        for j in range(n):
            if board[i][j] != 0:
                correct_i, correct_j = divmod(correct_board[i][j], n)
                manhattan_distance += abs(i - correct_i) + abs(j - correct_j)
    return manhattan_distance

if __name__ == "__main__":
    incorrect_board = [[7, 1, 4], [5, 2, 3], [-1, 6, 8]]
    print("Manhatten Distance of Tiles
Are:",manhattan_distance(incorrect_board))
```

3) A* Implementation:

```
4) import heapq

   class Slidepuzzle:
       def   init  (self, incorrect state, desired state):
           self.incorrect_state = incorrect_state
           self.desired_state = desired_state

       def solvable(self, state):
           inversions = 0
           n = len(state)

           for i in range(n * n - 1):
               for j in range(i + 1, n):
                   if state[i] > state[j] and state[i] != 0 and state[j]
   != 0:
                       inversions += 1
           return inversions % 2 == 0

       def manhattan_distance(self, state):
           n = len(state)
           distance = 0
```

```
        for i in range(n):
                for j in range(n):
                    tile = state[i - n + j]
                    if tile != 0:
                        goal_i, goal_j =
divmod(self.desired_state.index(tile), n)
                        distance += abs(i - goal_i) + abs(j - goal_j)
        return distance

    def solve(self):
        if not self.solvable(self.incorrect_state):
            return None
        open_set = []
        heapq.heappush(open_set, (0 +
self.manhattan_distance(self.incorrect_state), 0, self.incorrect_state,
[]))
        closed_set = set()

        while open_set:_, cost, current_state, path =
heapq.heappop(open_set)
        if current_state == self.desired_state:
            return path
        closed_set.add(tuple(current_state))
        n = int(len(current_state) ** 0.5)
        zero_index = current_state.index(0)
        i, j = divmod(zero_index, n)

        for move in [(0, -1), (-1, 0), (0, 1), (1, 0)]:
            new_i, new_j = i + move[0], j + move[1]
            if 0 <= new_i < n and 0 <= new_j < n:
                new_zero_index = new_i * n + new_j
                new_state = list(current_state)
                new_state[zero_index], new_state[new_zero_index] =
new_state[new_zero_index], new_state[zero_index]
            if tuple(new_state) not in closed_set:
                updated_cost = cost + 1
                heapq.heappush(open_set, (updated_cost +
self.manhattan_distance(new_state), updated_cost, new_state, path +
[new_state]))
            if not open_set:
                return None
            else:
                return path
if __name__ == "__main__":
    incorrect_state = [1, 2, 3, 4, 0, 5, 6, 7, 8]
    desired_state = [1, 2, 3, 4, 5, 6, 7, 8, 0]
    slide_puzzle_solver = Slidepuzzle(incorrect_state, desired_state)
    solution = slide_puzzle_solver.solve()
    if solution is not None:
        print("Solution has been found:")
        for step, state in enumerate(solution):
            print(f"Step {step + 1}:")
            for i in range(0, len(state), 3):
                print(state[i:i + 3])
    else:
        print("No solution has been found.")
```

4) COMPARISION:

The sum of misplaced tiles heuristic is simpler and faster to calculate than the Manhattan distance heuristic. This is because the sum of misplaced tiles heuristic only counts the number of tiles that are not in their correct positions, while the Manhattan distance heuristic calculates the distance between each misplaced tile and its correct position. The sum of misplaced tiles heuristic has a time complexity of O(n), while the Manhattan distance heuristic has a time complexity of O(n^2), where n is the number of tiles in the puzzle.

In other words, the sum of misplaced tiles heuristic is a less accurate but faster heuristic, while the Manhattan distance heuristic is a more accurate but slower heuristic.