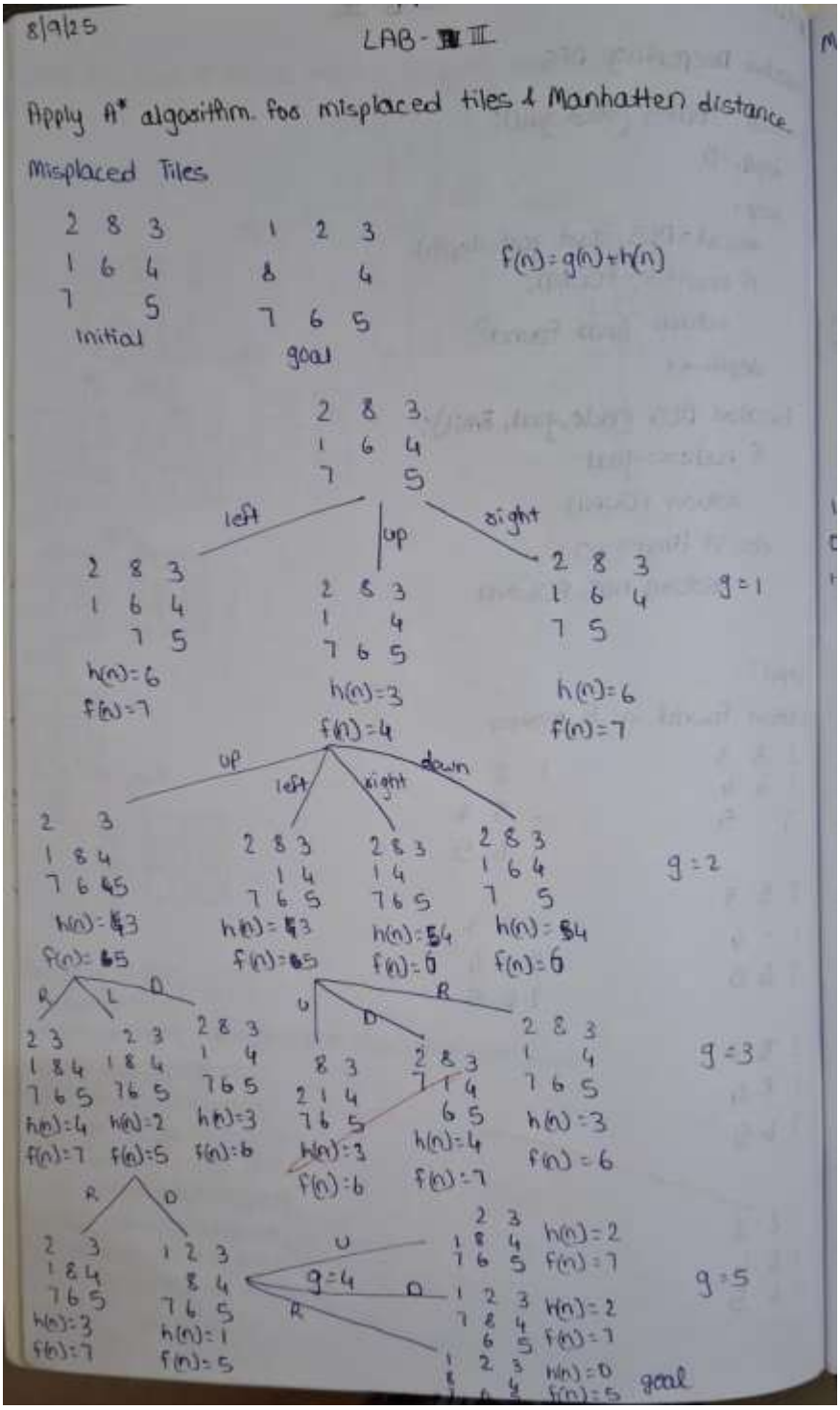
Lab 3

Apply A* algorithm for misplaced tiles.

Algorithm:



Code:

```
print("Shreya Raj 1BM23CS317")
import heapq

class PuzzleState:
    def __init__(self, board, goal, parent=None, g=0):
        self.board = board
        self.goal = goal
        self.parent = parent
        self.g = g
        self.h = self.misplaced_tiles()
        self.f = self.g + self.h

    def misplaced_tiles(self):
        """Count misplaced tiles (excluding 0)."""
        return sum(1 for i in range(9) if self.board[i] != 0 and self.board[i] != self.goal[i])

    def get_neighbors(self):
        """Generate possible moves by sliding the blank (0)."""
        neighbors = []
        idx = self.board.index(0)
        x, y = divmod(idx, 3)   # row, col
        moves = [(-1,0),(1,0),(0,-1),(0,1)]  # up, down, left, right

        for dx, dy in moves:
            nx, ny = x+dx, y+dy
            if 0 <= nx < 3 and 0 <= ny < 3:
```

```
                new_idx = nx*3 + ny
                new_board = self.board[:]
                new_board[idx], new_board[new_idx] = new_board[new_idx], new_board[idx]
                neighbors.append(PuzzleState(new_board, self.goal, self, self.g+1))
        return neighbors

    def __lt__(self, other):
        return self.f < other.f  # priority queue uses f value

def reconstruct_path(state):
    path = []
    while state:
        path.append(state.board)
        state = state.parent
    return path[::-1]

def astar(start, goal):
    start_state = PuzzleState(start, goal)
    open_list = []
    heapq.heappush(open_list, start_state)
    closed_set = set()

    while open_list:
        current = heapq.heappop(open_list)

        if current.board == goal:
            return reconstruct_path(current)
```

```
        closed_set.add(tuple(current.board))

        for neighbor in current.get_neighbors():
            if tuple(neighbor.board) in closed_set:
                continue
            heapq.heappush(open_list, neighbor)
    return None

print("Enter the 8-puzzle START state (use 0 for blank).")
start_input = list(map(int, input("Enter 9 numbers separated by spaces: ").split()))

print("\nEnter the GOAL state (use 0 for blank).")
goal_input = list(map(int, input("Enter 9 numbers separated by spaces: ").split()))

if len(start_input) != 9 or len(goal_input) != 9:
    print("Invalid input! Please enter exactly 9 numbers for each state.")
else:
    solution = astar(start_input, goal_input)

    if solution:
        print("\n Steps to solve:")
        for step in solution:
            for i in range(0,9,3):
                print(step[i:i+3])
            print("------")
    else:
```

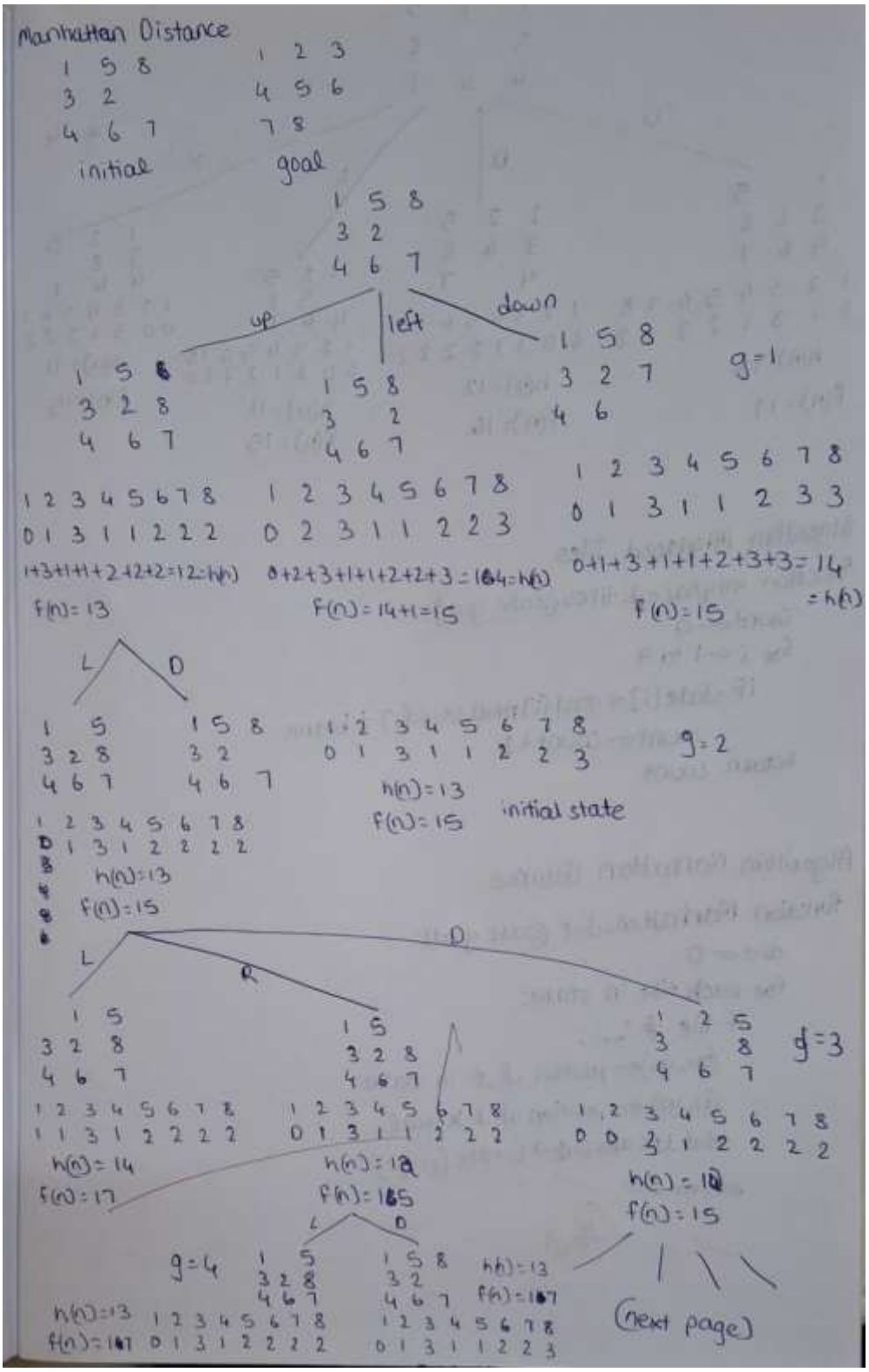
```
    print(" No solution found!")
```

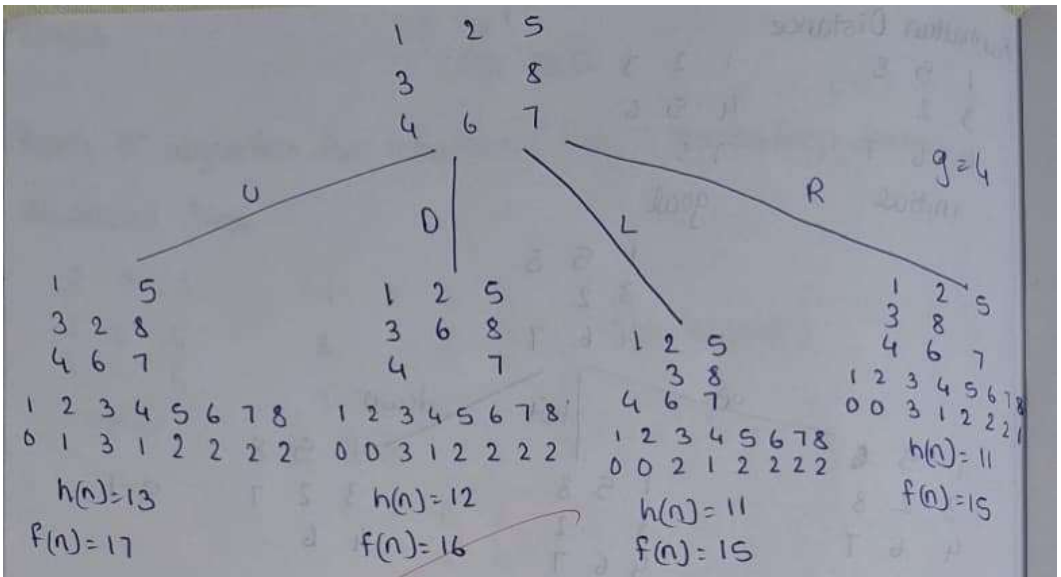
Output:

```
Shreya Raj 1BM23CS317
Enter the 8-puzzle START state (use 0 for blank).
Enter 9 numbers separated by spaces: 2 8 3 1 6 4 7 0 5

Enter the GOAL state (use 0 for blank).
Enter 9 numbers separated by spaces: 1 2 3 8 0 4 7 6 5

Steps to solve:
[2, 8, 3]
[1, 6, 4]
[7, 0, 5]
------
[2, 8, 3]
[1, 0, 4]
[7, 6, 5]
------
[2, 0, 3]
[1, 8, 4]
[7, 6, 5]
------
[0, 2, 3]
[1, 8, 4]
[7, 6, 5]
------
[1, 2, 3]
[0, 8, 4]
[7, 6, 5]
------
[1, 2, 3]
[8, 0, 4]
[7, 6, 5]
------
```

Apply A* algorithm for Manhattan Distance.

Algorithm:





Algorithm Misplaced Tiles

```
function misplaced_tiles(state, goal):
    count ← 0
    for i ← 1 to 9
        if state[i] ≠ goal[i] AND state[i] ≠ blank
            count ← count + 1
    return count
```

Algorithm Manhattan Distance

```
function Manhatten_dist (state, goal):
    dist ← 0
    for each tile in state:
        if tile ≠ '_':
            (x1, y1) ← position of t in state
            (x2, y2) ← position of t in goal
            dist += abs(x1-x2) + abs(y1-y2)
    return dist
```

8/9

Code:

```
import heapq


class PuzzleState:
    def __init__(self, board, goal, parent=None, g=0):
        self.board = board
        self.goal = goal
        self.parent = parent
        self.g = g
        self.h = self.manhattan_distance()
        self.f = self.g + self.h

    def manhattan_distance(self):
        """Heuristic: Manhattan Distance."""
        distance = 0
        for i, tile in enumerate(self.board):
            if tile != 0:  # skip blank
                goal_index = self.goal.index(tile)
                x1, y1 = divmod(i, 3)
                x2, y2 = divmod(goal_index, 3)
                distance += abs(x1 - x2) + abs(y1 - y2)
        return distance

    def get_neighbors(self):
        """Generate possible moves by sliding the blank (0)."""
        neighbors = []
        idx = self.board.index(0)
```

```
        x, y = divmod(idx, 3)
        moves = [(-1,0),(1,0),(0,-1),(0,1)]  # up, down, left, right

        for dx, dy in moves:
            nx, ny = x + dx, y + dy
            if 0 <= nx < 3 and 0 <= ny < 3:
                new_idx = nx * 3 + ny
                new_board = self.board[:]
                new_board[idx], new_board[new_idx] = new_board[new_idx], new_board[idx]
                neighbors.append(PuzzleState(new_board, self.goal, self, self.g + 1))
        return neighbors

    def __lt__(self, other):
        return self.f < other.f  # priority queue uses f value

def reconstruct_path(state):
    path = []
    while state:
        path.append(state.board)
        state = state.parent
    return path[::-1]

def astar(start, goal):
    start_state = PuzzleState(start, goal)
    open_list = []
    heapq.heappush(open_list, start_state)
    closed_set = set()
```

```
    while open_list:
        current = heapq.heappop(open_list)

        if current.board == goal:
            return reconstruct_path(current)

        closed_set.add(tuple(current.board))

        for neighbor in current.get_neighbors():
            if tuple(neighbor.board) in closed_set:
                continue
            heapq.heappush(open_list, neighbor)
    return None
print("Shreya Raj 1BM23CS317")
print("Enter the 8-puzzle START state (use 0 for blank).")
start_input = list(map(int, input("Enter 9 numbers separated by spaces: ").split()))

print("\nEnter the GOAL state (use 0 for blank).")
goal_input = list(map(int, input("Enter 9 numbers separated by spaces: ").split()))

if len(start_input) != 9 or len(goal_input) != 9:
    print("Invalid input! Please enter exactly 9 numbers for each state.")
else:
    solution = astar(start_input, goal_input)

    if solution:
```

```
    print("\nSteps to solve:")
    for step in solution:
        for i in range(0, 9, 3):
            print(step[i:i+3])
        print("------")
    print(f"Total moves: {len(solution)-1}")
else:
    print("No solution found!")
```

Output:

```
Shreya Raj 1BM23CS317
Enter the 8-puzzle START state (use 0 for blank).
Enter 9 numbers separated by spaces: 2 8 3 1 6 4 7 0 5

Enter the GOAL state (use 0 for blank).
Enter 9 numbers separated by spaces: 1 2 3 8 0 4 7 6 5

Steps to solve:
[2, 8, 3]
[1, 6, 4]
[7, 0, 5]
------
[2, 8, 3]
[1, 0, 4]
[7, 6, 5]
------
[2, 0, 3]
[1, 8, 4]
[7, 6, 5]
------
[0, 2, 3]
[1, 8, 4]
[7, 6, 5]
------
[1, 2, 3]
[0, 8, 4]
[7, 6, 5]
------
[1, 2, 3]
[8, 0, 4]
[7, 6, 5]
------
Total moves: 5
```