Week 3:

A*_MisplaceTiles

CODE:

```
#Heauristic approach to 8-puzzle problem

import heapq

def solve_8puzzle(initial_state):
    goal_state = [[1, 2, 3], [8, 0, 4], [7, 6, 5]]
    priority_queue = [(heuristic(initial_state, goal_state), 0, initial_state, [])]
    visited = set()

    while priority_queue:
        f_cost, g_cost, current_state, current_path = heapq.heappop(priority_queue)

        if current_state == goal_state:
            return current_path + [current_state]

        if tuple(map(tuple, current_state)) in visited:
            continue
        visited.add(tuple(map(tuple, current_state)))

        for next_state, action in get_possible_moves(current_state):
            new_g_cost = g_cost + 1
            new_f_cost = new_g_cost + heuristic(next_state, goal_state)
            heapq.heappush(priority_queue, (new_f_cost, new_g_cost, next_state,
current_path + [(current_state, action)]))

    return None


def heuristic(state, goal_state):
    misplaced_tiles = 0
    for i in range(3):
        for j in range(3):
            if state[i][j] != goal_state[i][j] and state[i][j] != 0:
                misplaced_tiles += 1
```

```
    return misplaced_tiles


def find_position(state, tile):
    for i in range(3):
        for j in range(3):
            if state[i][j] == tile:
                return i, j


def get_possible_moves(state):
    row, col = find_position(state, 0)
    possible_moves = []

    if row > 0:
        new_state = [list(row) for row in state]
        new_state[row][col], new_state[row - 1][col] = new_state[row - 1][col],
new_state[row][col]
        possible_moves.append((new_state, 'Up'))
    if row < 2:
        new_state = [list(row) for row in state]
        new_state[row][col], new_state[row + 1][col] = new_state[row + 1][col],
new_state[row][col]
        possible_moves.append((new_state, 'Down'))
    if col > 0:
        new_state = [list(row) for row in state]
        new_state[row][col], new_state[row][col - 1] = new_state[row][col - 1],
new_state[row][col]
        possible_moves.append((new_state, 'Left'))
    if col < 2:
        new_state = [list(row) for row in state]
        new_state[row][col], new_state[row][col + 1] = new_state[row][col + 1],
new_state[row][col]
        possible_moves.append((new_state, 'Right'))

    return possible_moves


initial_state = [[2, 8, 3], [1, 6, 4], [0, 7, 5]]
```

```python
solution = solve_8puzzle(initial_state)

if solution:
    print("Solution found:")
    for state, action in solution[:-1]:
        print("--------------------")
        for row in state:
            print(row)
        print("Move:", action)
    print("--------------------")
    for row in solution[-1]:
        print(row)
else:
    print("No solution found.")
```

Output:

```
Solution found:
--------------------
[2, 8, 3]
[1, 6, 4]
[0, 7, 5]
Move: Right
--------------------
[2, 8, 3]
[1, 6, 4]
[7, 0, 5]
Move: Up
--------------------
[2, 8, 3]
[1, 0, 4]
[7, 6, 5]
Move: Up
--------------------
[2, 0, 3]
[1, 8, 4]
[7, 6, 5]
Move: Left
--------------------
[0, 2, 3]
[1, 8, 4]
[7, 6, 5]
Move: Down
--------------------
[1, 2, 3]
[0, 8, 4]
[7, 6, 5]
Move: Right
--------------------
[1, 2, 3]
[8, 0, 4]
[7, 6, 5]
```

15/10/24.

WEEK-3

A* implementation

(i) Misplace Tiles

$g(n)$: Depth of the node

$h(n)$: Number of missplaced tiles.

Initial state:

2	8	3
1	6	4
7		5

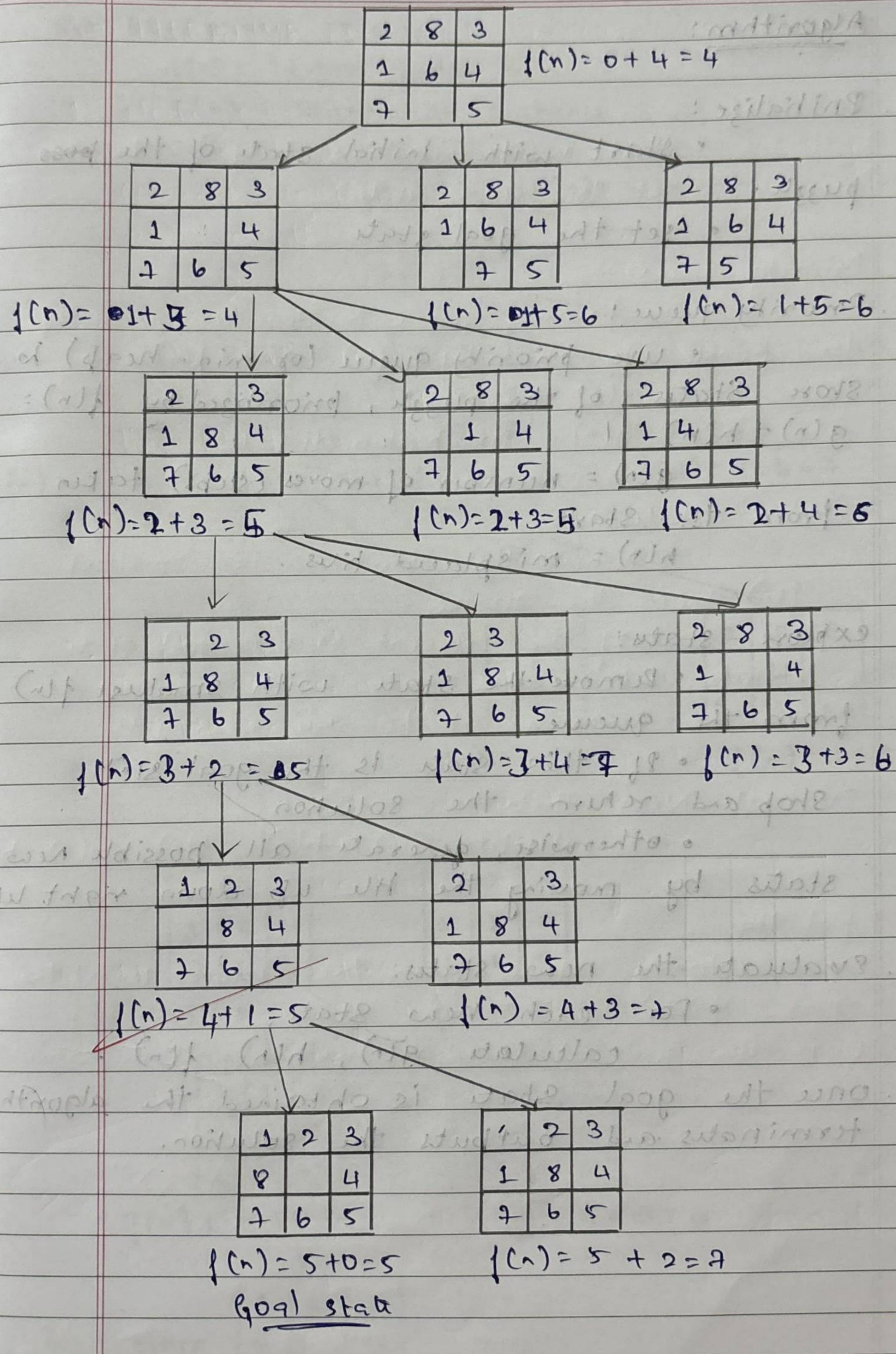
Final state:

1	2	3
8		4
7	6	5

$g(n) = 0$

$h(n) = 4$ (Misplaced 1, 2, 8, 6)

$f(n) = g(n) + h(n) = 0 + 4 = 4$


f(n)= 0+4 = 4
f(n)= 1+3 = 4
f(n)= 1+5=6
f(n)= 1+5 =6
f(n)= 2+3 = 5
f(n)= 2+3=5
f(n)= 2+4 =6
f(n)= 3+2 = 5
f(n)= 3+4 =7
f(n) = 3+3= 6
f(n)= 4+1 =5
f(n) = 4+3 =7
f(n)= 5+0=5
Goal state
f(n)= 5 + 2=7

Algorithm:

1. Initialize:
   - Start with initial state of the ~~poss~~ puzzle.
   - set the goal state

2. Priority queue:
   - use priority queue (or min-heap) to store states of the puzzle, prioritized by $f(n) = g(n) + h(n)$

     $g(n)$ = number of moves (steps) taken from the start

     $h(n)$ = misplaced tiles.

3. explore states:
   - Remove the state with smallest $f(n)$ from the queue
   - If this state is the goal state, stop and return the solution
   - otherwise, generate all possible new states by moving the tile up down right left.

4. Evaluate the new states:
   - For each new state, calculate $g(n)$, $h(n)$ $f(n)$.

5. once the goal state is obtained the algorithm terminates and outputs the solution.