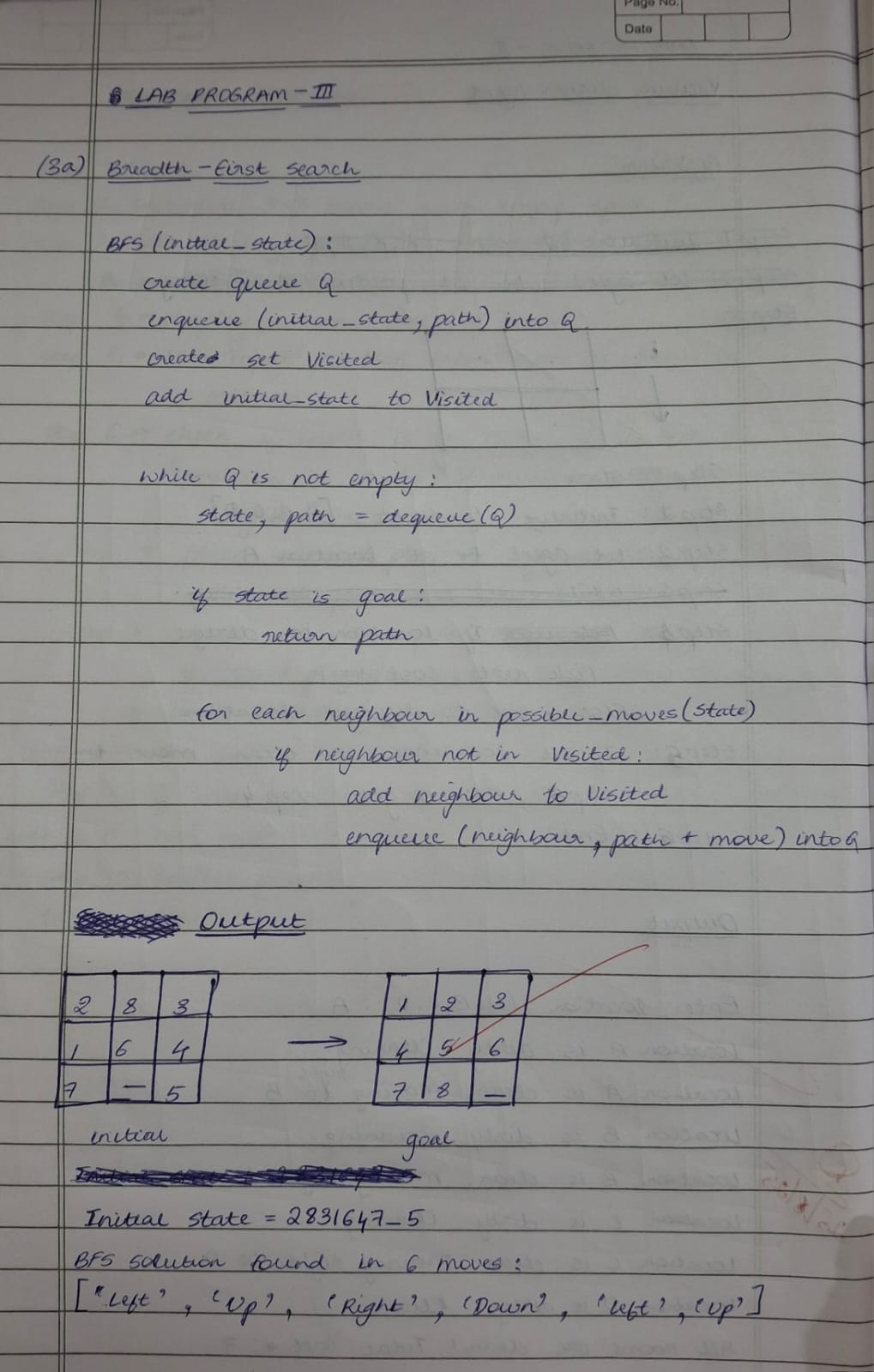
LAB PROGRAM 3

1. Solve 8 puzzle by Breadth First Search method (non-heuristic approach)

PSEUDOCODE:





CODE:

from collections import deque

GOAL\_STATE = (1, 2, 3, 8, 0, 4, 7, 6, 5)

MOVES = {

    'left': -1,

    'right': 1,

    'up': -3,

    'down': 3,

}

def is\_valid\_move(blank\_idx, move):

    if move == 'left' and blank\_idx % 3 == 0:

        return False

    if move == 'right' and blank\_idx % 3 == 2:

        return False

    if move == 'up' and blank\_idx < 3:

        return False

    if move == 'down' and blank\_idx > 5:

        return False

    return True

def get\_neighbors(state):

    neighbors = []

    blank\_idx = state.index(0)

    for move, delta in MOVES.items():

        if is\_valid\_move(blank\_idx, move):

            new\_idx = blank\_idx + delta

            new\_state = list(state)

            new\_state[blank\_idx], new\_state[new\_idx] = new\_state[new\_idx], new\_state[blank\_idx]

            neighbors.append(tuple(new\_state))

    return neighbors

def bfs(start\_state):

    queue = deque([start\_state])

    visited = set([start\_state])

    parent = {start\_state: None}

    explored\_count = 0

    while queue:

        current = queue.popleft()

        explored\_count += 1

        if current == GOAL\_STATE:

            path = []

            while current:

                path.append(current)

                current = parent[current]

            path.reverse()

            print(f"Total states explored (breadth-wise): {explored\_count}")

            return path

        for neighbor in get\_neighbors(current):

            if neighbor not in visited:

                visited.add(neighbor)

                parent[neighbor] = current

                queue.append(neighbor)

    print(f"Total states explored (breadth-wise): {explored\_count}")

    return None

def print\_state(state):

    for i in range(0, 9, 3):

        print(state[i:i+3])

    print()

if \_\_name\_\_ == "\_\_main\_\_":

    start = (2, 8, 3,

             1, 6, 4,

             7, 0, 5)

    print("Starting BFS 8-puzzle solver...\nInitial state:")

    print\_state(start)

    print("Sinchana Hemanth (1BM23CS330)")

    solution = bfs(start)

    if solution:

        print(f"Solution found in {len(solution)-1} moves:\n")

        for step\_num, state in enumerate(solution):

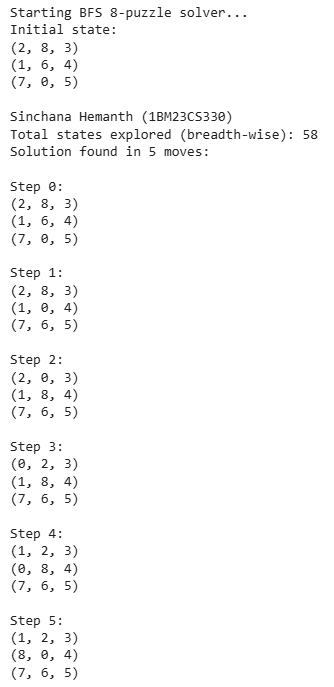
            print(f"Step {step\_num}:")

            print\_state(state)

    else:

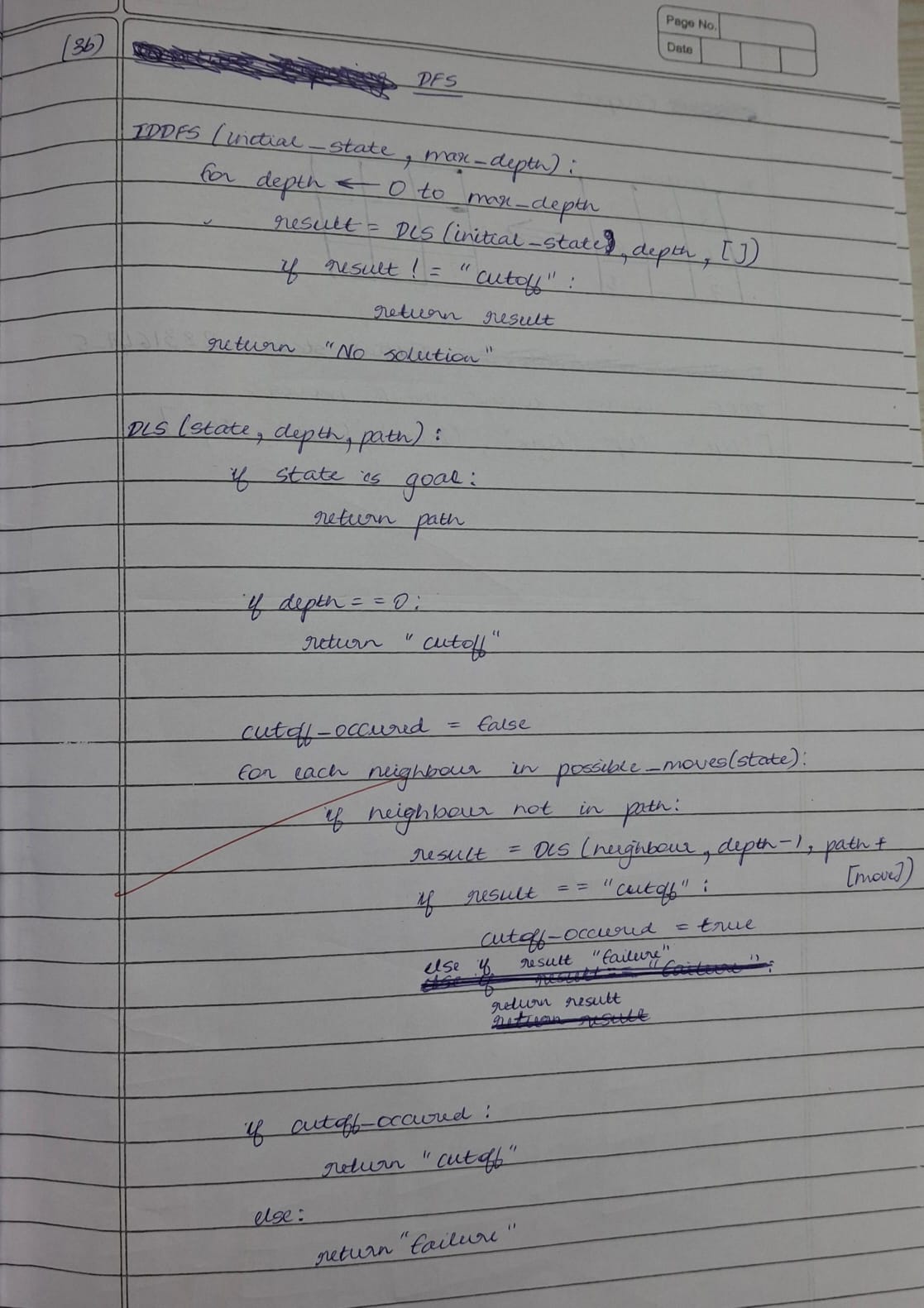
        print("No solution found.")

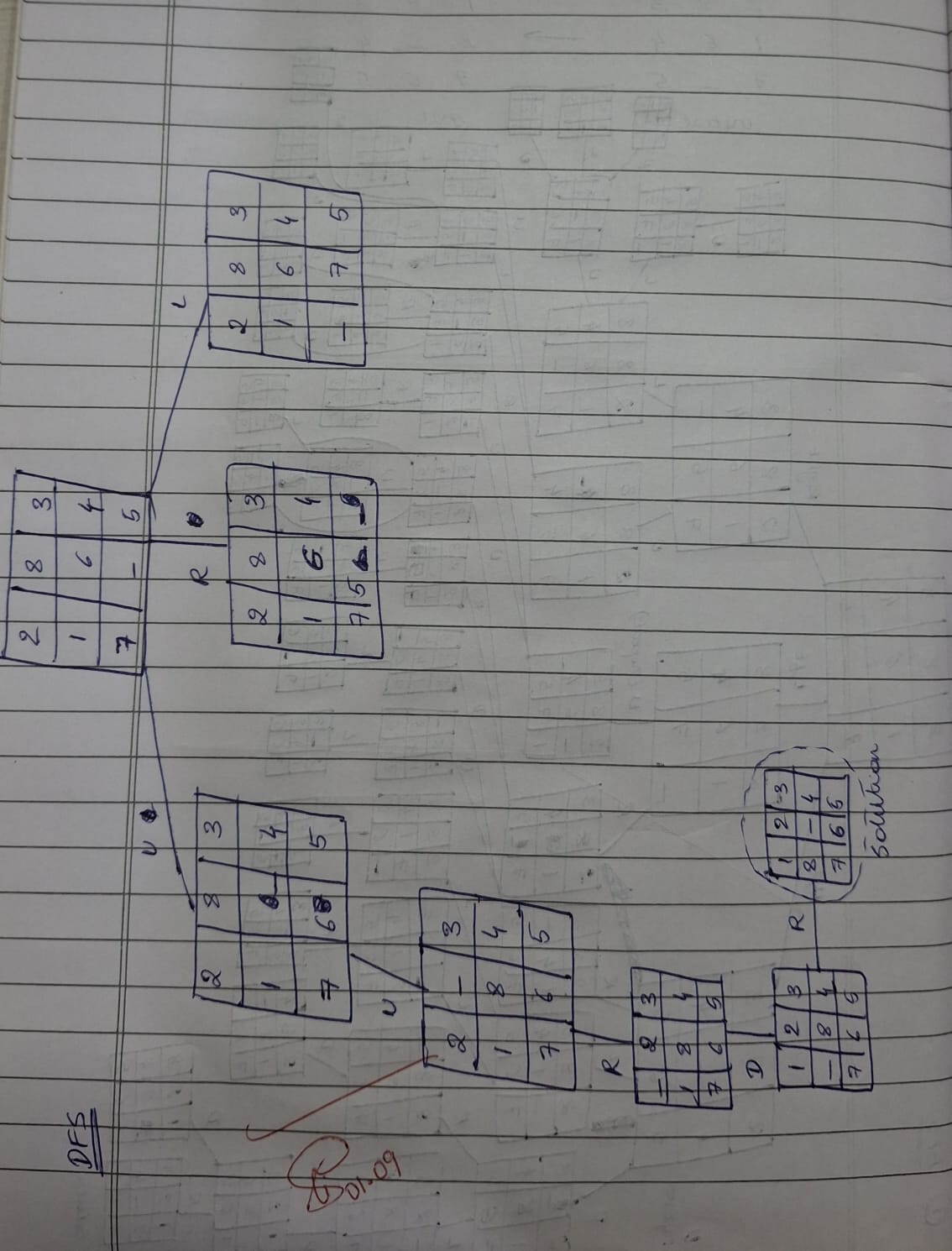
OUTPUT:



1. Solve 8 puzzle by Depth First Search method (non-heuristic approach)

PSEUDOCODE:





CODE:

GOAL\_STATE = (1, 2, 3, 4, 5, 6, 7, 8, 0)

MOVES = {

    'left': -1,

    'right': 1,

    'up': -3,

    'down': 3,

}

def is\_valid\_move(blank\_idx, move):

    if move == 'left' and blank\_idx % 3 == 0:

        return False

    if move == 'right' and blank\_idx % 3 == 2:

        return False

    if move == 'up' and blank\_idx < 3:

        return False

    if move == 'down' and blank\_idx > 5:

        return False

    return True

def get\_neighbors(state):

    neighbors = []

    blank\_idx = state.index(0)

    for move, delta in MOVES.items():

        if is\_valid\_move(blank\_idx, move):

            new\_idx = blank\_idx + delta

            new\_state = list(state)

            new\_state[blank\_idx], new\_state[new\_idx] = new\_state[new\_idx], new\_state[blank\_idx]

            neighbors.append(tuple(new\_state))

    return neighbors

def dfs(start\_state, max\_depth=50):

    stack = [(start\_state, 0)]

    visited = set([start\_state])

    parent = {start\_state: None}

    while stack:

        current, depth = stack.pop()

        if current == GOAL\_STATE:

            path = []

            while current:

                path.append(current)

                current = parent[current]

            path.reverse()

            return path

        if depth < max\_depth:

            for neighbor in get\_neighbors(current):

                if neighbor not in visited:

                    visited.add(neighbor)

                    parent[neighbor] = current

                    stack.append((neighbor, depth + 1))

    return None

def print\_state(state):

    for i in range(0, 9, 3):

        print(state[i:i+3])

    print()

if \_\_name\_\_ == "\_\_main\_\_":

    start = (1, 2, 3,

             4, 0, 6,

             7, 5, 8)

    print("Starting DFS 8-puzzle solver...\nInitial state:")

    print\_state(start)

    print("Sinchana Hemanth (1BM23CS330)")

    solution = dfs(start, max\_depth=20)

    if solution:

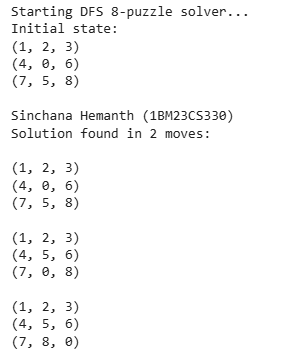
        print(f"Solution found in {len(solution)-1} moves:\n")

        for step in solution:

            print\_state(step)

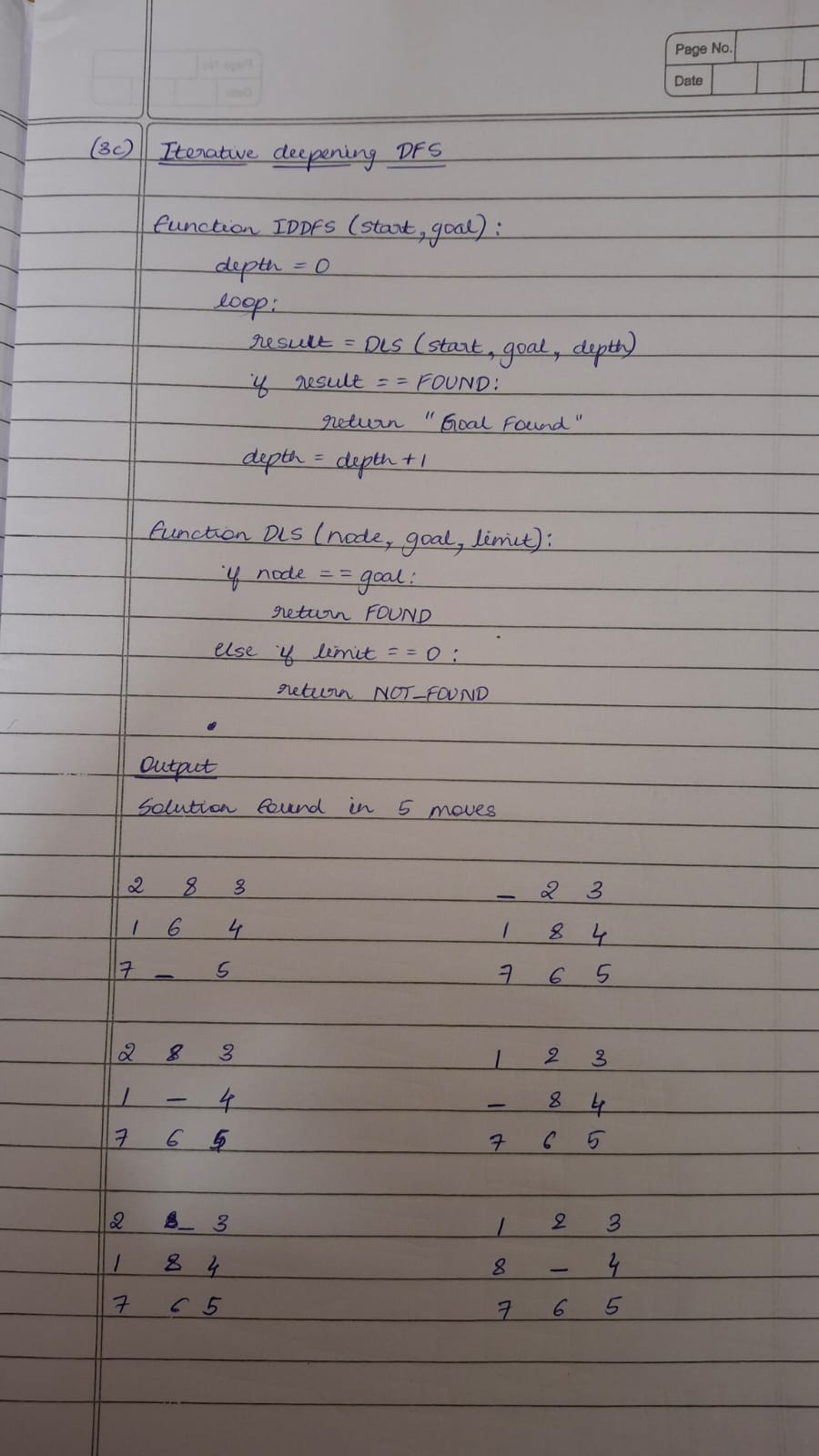
    else:

        print("No solution found or max depth exceeded.")

OUTPUT:  


1. Solve 8 puzzles by Iterative Deepening Depth First Search (IDDFS)

PSEUDOCODE:



CODE:

from collections import deque

N = 3

moves = [(-1,0),(1,0),(0,-1),(0,1)]

def find\_blank(state):

    idx = state.index("\_")

    return divmod(idx, N)

def swap(state, i1, j1, i2, j2):

    s = list(state)

    idx1, idx2 = i1\*N+j1, i2\*N+j2

    s[idx1], s[idx2] = s[idx2], s[idx1]

    return tuple(s)

def expand(state):

    x, y = find\_blank(state)

    children = []

    for dx, dy in moves:

        nx, ny = x+dx, y+dy

        if 0 <= nx < N and 0 <= ny < N:

            children.append(swap(state, x, y, nx, ny))

    return children

def dls(state, goal, limit, path, visited):

    if state == goal:

        return path

    if limit == 0:

        return None

    visited.add(state)

    for child in expand(state):

        if child not in visited:

            result = dls(child, goal, limit-1, path+[child], visited)

            if result is not None:

                return result

    visited.remove(state)

    return None

def iddfs(start, goal, max\_depth=20):

    for depth in range(max\_depth):

        visited = set()

        result = dls(start, goal, depth, [start], visited)

        if result is not None:

            return result

    return None

print("Sinchana Hemanth (1BM23CS330)")

initial = (2,8,3,1,6,4,7,"\_",5)

goal    = (1,2,3,8,"\_",4,7,6,5)

solution = iddfs(initial, goal, max\_depth=30)

if solution:

    print("Solution found in", len(solution)-1, "moves:\n")

    for step in solution:

        for i in range(0, 9, 3):

            print(step[i:i+3])

        print()

else:

    print("No solution found within depth limit")

OUTPUT:

