1. Solving 8 puzzle problems:

Pseudocode:

Class node:

```
    Function_init_(state, parent=none, action=none,
    path_cost=0):
        SET self.state = state
        SET self.parent = parent
        SET self.action = action
        SET self.path_cost = path_cost
    Function expand():
        CREATE children
        SET row, col = find_blank()
        CREATE possible_actions
        IF row>0 THEN ADD 'Up' to possible_actions
        IF row<2 THEN ADD 'Down' to possibleaction
        IF col>0 THEN ADD 'left' to possible_action
        FOR action IN possible_actions:
            CREATE new_state as a copy of selfstate
```

```
IF action == 'Up' THEN SWAP new_state[row],
[col] with new_state[row-1][col]
ELSE IF action == 'Down' THEN SWAP
new_state[row][col] with new_state[row+1]
[col]
ELSE IF action == 'Left' THEN SWAP new_state
[row][col] with new_state[row][col-1]
ELSE IF action == 'Right' THEN SWAP new_
state[row][col] with new_state[row][col+1]
APPEND newnode(new_state, self.action,
self.path_cost+1) to children.
RETURN children
FUNCTION find_blank():
FOR row FROM 0 TO 2:
FOR col FROM 0 to 2:
IF self.state[row][col]==0 THEN
RETURN row, col
FUNCTION depth_first_search(initial_state,
goal_state):
SET frontier = [node(initial_state)]
SET explored = empty set
WHILE frontier is not empty:
SET node = frontier.pop()
IF node.state == goal_state
THEN RETURN node
ADD tuple of node, state to explored
FOR child IN node.expand():
IF tuple of child.state NOT IN explored
THEN APPEND child to frontier
RETURN none
FUNCTION print_solution(node):
CREATE path
WHILE node is not None:
```

```
        APPEND (node.action, node.state) to path
        SET node = node.parent
    REVERSE path
    FOR (action, state) IN path:
        IF action is not none THEN
        PRINT "Action:", action
        PRINT state
        PRINT ""
SET initial_state = [[1,2,3],[0,4,6],[7,5,8]]
SET goal_state = [[1,2,3],[4,5,6],[7,8,0]]
SET solution = depth_first_search(initial_state,
 goal_state)
IF solution is not none THEN PRINT "Solution
 found:"
CALL print_solution(solution)
ELSE
    PRINT "Solution not found"
```

) Implementing Iterative Deepening Search.

Pseudocode:

```
FUNCTION iterative_deepening_search(initial_state,
 goal_state, depth):
    FOR depth FROM 0 TO max_depth:
        SET result = depth_limited_search(initial_state,
        goal_state, depth)
            IF result is not none THEN
        RETURN result
            RETURN none
FUNCTION depth_limited_search(node, goal_state,
limit):
    IF node.state == goal_state THEN RETURN node
    IF node.depth >= limit THEN RETURN none
        FOR each child IN expand(node):
```

Date __/__/__

Page _____

SET result = depth-limited_search (child.goal_state, limit)

IF result is not None THEN RETURN result

RETURN none.

SET initial_state

SET goal_state

SET max_depth

SET solution = iterative_deepening_search (initial_state goal_state, max_depth)

IF solution is not none THEN PRINT solution

ELSE PRINT "no solution found"