Assignment 6

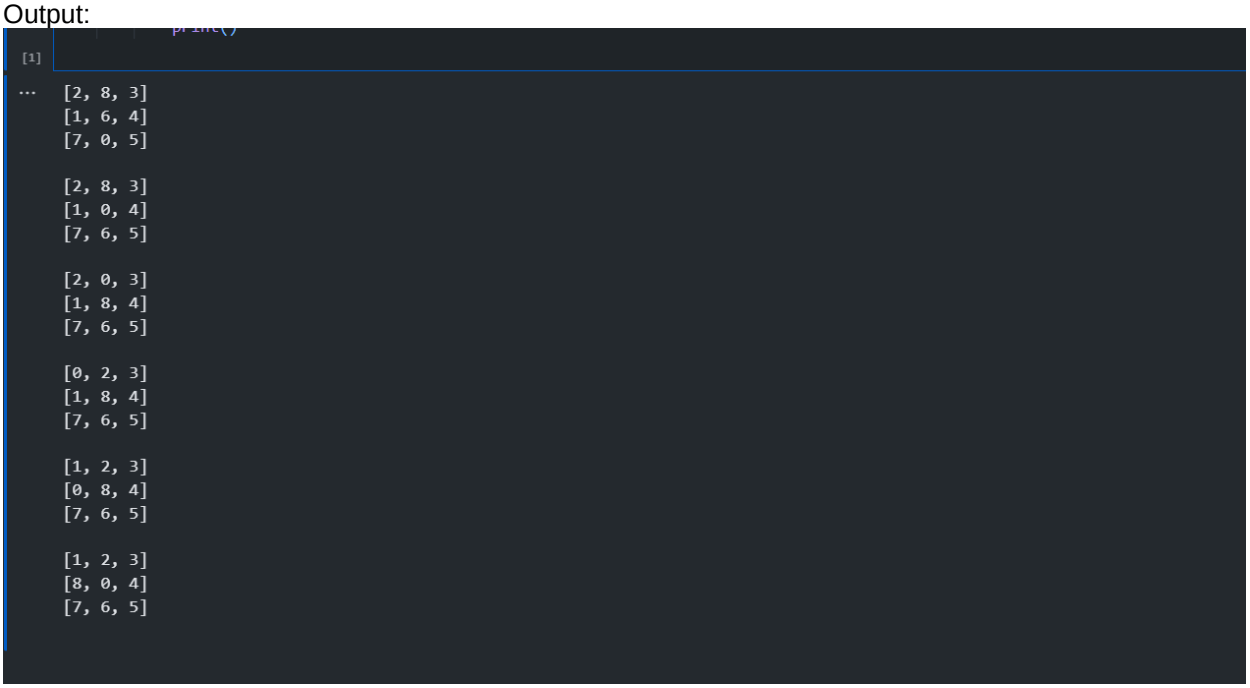
Sarthak Kapaliya
20BCP072
G2

A* is generally more efficient than Hill climbing in finding the optimal solution to the n-puzzle problem. However, Hill climbing can be faster in some cases and requires less memory to store the search tree.

```
import heapq

class Node:
    def __init__(self, state, parent=None, g=0, h=0):
        self.state = state
        self.parent = parent
        self.g = g
        self.h = h

    def f(self):
        return self.g + self.h

    def __lt__(self, other):
        return self.f() < other.f()

    def __eq__(self, other):
        return self.state == other.state

def misplaced_tiles(state):
    goal_state = [1, 2, 3, 8,0,4,7,6,5]
    return sum([1 for i in range(9) if state[i] != goal_state[i]])

def get_children(node):
    children = []
    blank_index = node.state.index(0)
    if blank_index % 3 > 0:
        left_child_state = node.state[:]
        left_child_state[blank_index], left_child_state[blank_index - 1] =
left_child_state[blank_index - 1], left_child_state[blank_index]
        children.append(Node(left_child_state, node, node.g + 1,
misplaced_tiles(left_child_state)))
    if blank_index % 3 < 2:
        right_child_state = node.state[:]
        right_child_state[blank_index], right_child_state[blank_index + 1] =
right_child_state[blank_index + 1], right_child_state[blank_index]
        children.append(Node(right_child_state, node, node.g + 1,
misplaced_tiles(right_child_state)))
    if blank_index // 3 > 0:
```

```
        up_child_state = node.state[:]
        up_child_state[blank_index], up_child_state[blank_index - 3] =
up_child_state[blank_index - 3], up_child_state[blank_index]
        children.append(Node(up_child_state, node, node.g + 1,
misplaced_tiles(up_child_state)))
    if blank_index // 3 < 2:
        down_child_state = node.state[:]
        down_child_state[blank_index], down_child_state[blank_index + 3] =
down_child_state[blank_index + 3], down_child_state[blank_index]
        children.append(Node(down_child_state, node, node.g + 1,
misplaced_tiles(down_child_state)))
    return children

def a_star(initial_state):
    start_node = Node(initial_state, None, 0, misplaced_tiles(initial_state))
    open_list = [start_node]
    closed_list = []
    while open_list:
        current_node = heapq.heappop(open_list)
        if current_node.h == 0:
            path = []
            while current_node:
                path.append(current_node.state)
                current_node = current_node.parent
            return list(reversed(path))
        closed_list.append(current_node)
        for child in get_children(current_node):
            if child in closed_list:
                continue
            if child not in open_list:
                heapq.heappush(open_list, child)
            else:
                existing_node = open_list[open_list.index(child)]
                if child.g < existing_node.g:
                    existing_node.g = child.g
                    existing_node.parent = child.parent
    return None

if __name__ == "__main__":
    initial_state = [2,8,3,1,6,4,7,0,5]
    path = a_star(initial_state)
    for state in path:
        print(state[0:3]) # print the first three elements
        print(state[3:6]) # print the next three elements
        print(state[6:9])
        print()
```

Output:


[1]
[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]