1. Discuss the importance of visualizing the solutions of the N-Queens Problem to understand the placement of queens better. Use a graphical representation to show how queens are placed on the board for different values of N. Explain how visual tools can help in debugging the algorithm and gaining insights into the problem's complexity. Provide examples of visual representations for N = 4, N = 5, and N = 8, showing different valid solutions.
   a. Visualization for 4-Queens:
      Input: N = 4
      Output:
      Explanation: Each 'Q' represents a queen, and '.' represents an empty space.
   b. Visualization for 5-Queens:
      Input: N = 5
      Output:
   c. Visualization for 8-Queens:
      Input: N = 8
      Output:

```python
def print_board(board):
    for row in board:
        print(" ".join(row))
    print()
def solve_n_queens(n):
    def is_safe(board, row, col):
        for i in range(row):
            if board[i][col] == 'Q':
                return False
            if col - (row - i) >= 0 and board[i][col - (row - i)] == 'Q':
                return False
            if col + (row - i) < n and board[i][col + (row - i)] == 'Q':
                return False
        return True
    def solve(board, row):
        if row == n:
            print_board(board)
            return
        for col in range(n):
            if is_safe(board, row, col):
                board[row][col] = 'Q'
                solve(board, row + 1)
                board[row][col] = '.'
    board = [['.' for _ in range(n)] for _ in range(n)]
    solve(board, 0)
print("N = 4:")
solve_n_queens(4)
print("N = 5:")
solve_n_queens(5)
print("N = 8:")
solve_n_queens(8)
```

```python
            return
        for col in range(n):
            if is_safe(board, row, col):
                board[row][col] = 'Q'
                solve(board, row + 1)
                board[row][col] = '.'
    board = [['.' for _ in range(n)] for _ in range(n)]
    solve(board, 0)
print("N = 4:")
solve_n_queens(4)
print("N = 5:")
solve_n_queens(5)
print("N = 8:")
solve_n_queens(8)
```

```
. . . . . . . Q
. . . Q . . . .
Q . . . . . . .
. . Q . . . . .
. . . . . Q . .
. Q . . . . . .
. . . . . . Q .
. . . . Q . . .
```

2. Discuss the generalization of the N-Queens Problem to other board sizes and shapes, such as rectangular boards or boards with obstacles. Explain how the algorithm can be adapted to handle these variations and the additional constraints they introduce. Provide examples of solving generalized N-Queens Problems for different board configurations, such as an 8×10 board, a 5×5 board with obstacles, and a 6×6 board with restricted positions.
   a. 8×10 Board:
      8 rows and 10 columns
      Output: Possible solution [1, 3, 5, 7, 9, 2, 4, 6]
      Explanation: Adapt the algorithm to place 8 queens on an 8×10 board, ensuring no two queens threaten each other.
   b. 5×5 Board with Obstacles:
      Input: N = 5, Obstacles at positions [(2, 2), (4, 4)]
      Output: Possible solution [1, 3, 5, 2, 4]
      Explanation: Modify the algorithm to avoid placing queens on obstacle positions, ensuring a valid solution that respects the constraints.
   c. 6×6 Board with Restricted Positions:
      Input: N = 6, Restricted positions at columns 2 and 4 for the first queen
      Output: Possible solution [1, 3, 5, 2, 4, 6]
      Explanation: Adjust the algorithm to handle restricted positions, ensuring the queens are placed without conflicts and within allowed columns.

DAY 10 programs

```python
def is_safe(board, row, col):
    for i in range(row):
        if board[i] == col or \
           board[i] - i == col - row or \
           board[i] + i == col + row:
            return False
    return True
def solve_n_queens(n, board=None, row=0):
    if board is None:
        board = [-1] * n
    if row == n:
        return [board.copy()]
    solutions = []
    for col in range(n):
        if is_safe(board, row, col):
            board[row] = col
            solutions += solve_n_queens(n, board, row + 1)
            board[row] = -1
    return solutions
def solve_8x10():
    n = 8
    board = [0] * n
    solutions = []
    for col in range(10):
        if is_safe(board, 0, col):
            board[0] = col
            solutions += solve_n_queens(n, board, 1)
            board[0] = -1
    return solutions
def solve_5x5_with_obstacles(obstacles):
    n = 5
    board = [-1] * n
    solutions = []
    def is_safe_with_obstacles(board, row, col):
        if (row, col) in obstacles:
            return False
        return is_safe(board, row, col)
```

DAY 10 programs

```python
    def solve_with_obstacles(n, board=None, row=0):
        if board is None:
            board = [-1] * n
        if row == n:
            return [board.copy()]
        solutions = []
        for col in range(n):
            if is_safe_with_obstacles(board, row, col):
                board[row] = col
                solutions += solve_with_obstacles(n, board, row + 1)
                board[row] = -1
        return solutions
    return solve_with_obstacles(n, board)
def solve_6x6_with_restricted_positions(restricted):
    n = 6
    board = [-1] * n
    solutions = []
    def is_safe_with_restrictions(board, row, col):
        if col in restricted:
            return False
        return is_safe(board, row, col)
    def solve_with_restrictions(n, board=None, row=0):
        if board is None:
            board = [-1] * n
        if row == n:
            return [board.copy()]
        solutions = []
        for col in range(n):
            if is_safe_with_restrictions(board, row, col):
                board[row] = col
                solutions += solve_with_restrictions(n, board, row + 1)
                board[row] = -1
        return solutions
    return solve_with_restrictions(n, board)
print("8x10 Board Solutions:", solve_8x10())
print("5x5 Board with Obstacles Solutions:", solve_5x5_with_obstacles([(2, 2), (4, 4)]))
print("6x6 Board with Restricted Positions Solutions:", solve_6x6_with_restricted_positions([2, 4]))
```

```python
    board = [-1] * n
    solutions = []
    def is_safe_with_restrictions(board, row, col):
        if col in restricted:
            return False
        return is_safe(board, row, col)
    def solve_with_restrictions(n, board=None, row=0):
        if board is None:
```

input

```
 4, 7, 0, 2, 5], [9, 3, 1, 7, 2, 0, 6, 4], [9, 3, 1, 7, 4, 2, 0, 5], [9, 3, 1, 7, 4, 6, 0, 5], [9, 3, 5, 7, 2, 0, 6, 1], [9, 3, 5, 7, 2,
0, 6, 4], [9, 3, 6, 2, 7, 1, 4, 0], [9, 3, 6, 4, 2, 0, 5, 7], [9, 4, 0, 3, 6, 2, 5, 1], [9, 4, 0, 3, 6, 2, 7, 1], [9, 4, 1, 3, 0, 2, 7, 5
], [9, 4, 1, 3, 6, 2, 7, 5], [9, 4, 2, 0, 3, 1, 7, 5], [9, 4, 2, 0, 6, 1, 7, 5], [9, 4, 2, 7, 3, 6, 0, 5], [9, 4, 6, 0, 2, 7, 1, 3], [9,
4, 6, 0, 2, 7, 5, 3], [9, 4, 6, 0, 3, 1, 7, 5], [9, 4, 6, 3, 0, 2, 7, 5], [9, 5, 0, 2, 4, 6, 1, 3], [9, 5, 0, 2, 4, 7, 1, 3], [9, 5, 0, 4
, 1, 7, 2, 6], [9, 5, 1, 4, 6, 0, 2, 7], [9, 5, 1, 4, 7, 0, 6, 3], [9, 5, 2, 0, 3, 6, 4, 1], [9, 5, 2, 0, 3, 7, 4, 1], [9, 5, 2, 0, 7, 3,
 1, 6], [9, 5, 3, 0, 4, 7, 1, 6], [9, 6, 1, 3, 7, 0, 2, 5], [9, 6, 1, 5, 2, 0, 7, 3], [9, 6, 1, 5, 2, 0, 7, 4], [9, 6, 3, 0, 2, 7, 5, 1],
 [9, 6, 4, 2, 0, 5, 7, 1], [9, 7, 1, 4, 2, 0, 6, 3], [9, 7, 3, 0, 2, 5, 1, 6], [9, 7, 4, 2, 0, 6, 1, 5]]
5x5 Board with Obstacles Solutions: [[0, 2, 4, 1, 3], [0, 3, 1, 4, 2], [2, 4, 1, 3, 0], [3, 1, 4, 2, 0], [4, 1, 3, 0, 2], [4, 2, 0, 3, 1]
]
6x6 Board with Restricted Positions Solutions: []
```

3. Write a program to solve a Sudoku puzzle by filling the empty cells. A sudoku solution must satisfy all of the following rules: Each of the digits 1-9 must occur exactly once in each row. Each of the digits 1-9 must occur exactly once in each column. Each of the digits 1-9 must occur exactly once in each of the 9 3x3 sub-boxes of the grid. The '.' character indicates empty cells.

   Example 1:

   Input: board =

   [["5","3",".",".","7",".",".",".","."],
   ["6",".",".","1","9","5",".",".","."],

   [".","9","8",".",".",".",".","6","."],
   ["8",".",".",".","6",".",".",".","3"],
   ["4",".",".","8",".","3",".",".","1"],
   ["7",".",".",".","2",".",".",".","6"],
   [".","6",".",".",".",".","2","8","."],
   [".",".",".","4","1","9",".",".","5"],
   [".",".",".",".","8",".",".","7","9"]]

   Output:

   [["5","3","4","6","7","8","9","1","2"],
   ["6","7","2","1","9","5","3","4","8"],
   ["1","9","8","3","4","2","5","6","7"],
   ["8","5","9","7","6","1","4","2","3"],
   ["4","2","6","8","5","3","7","9","1"],
   ["7","1","3","9","2","4","8","5","6"],
   ["9","6","1","5","3","7","2","8","4"],
   ["2","8","7","4","1","9","6","3","5"],
   ["3","4","5","2","8","6","1","7","9"]]

DAY 10 programs

```python
def solve_sudoku(board):
    def is_valid(board, row, col, num):
        for i in range(9):
            if board[row][i] == num or board[i][col] == num:
                return False
            if board[3 * (row // 3) + i // 3][3 * (col // 3) + i % 3] == num:
                return False
        return True
    def solve(board):
        for row in range(9):
            for col in range(9):
                if board[row][col] == '.':
                    for num in map(str, range(1, 10)):
                        if is_valid(board, row, col, num):
                            board[row][col] = num
                            if solve(board):
                                return True
                            board[row][col] = '.'
                    return False
        return True
    solve(board)
board = [["5","3",".",".","7",".",".",".","."],
         ["6",".",".","1","9","5",".",".","."],
         [".","9","8",".",".",".",".","6","."],
         ["8",".",".",".","6",".",".",".","3"],
         ["4",".",".","8",".","3",".",".","1"],
         ["7",".",".",".","2",".",".",".","6"],
         [".","6",".",".",".",".","2","8","."],
         [".",".",".","4","1","9",".",".","5"],
         [".",".",".",".","8",".",".","7","9"]]

solve_sudoku(board)
print(board)
```

```python
            for col in range(9):
                if board[row][col] == '.':
                    for num in map(str, range(1, 10)):
                        if is_valid(board, row, col, num):
                            board[row][col] = num
                            if solve(board):
```

input

```
[['5', '3', '4', '6', '7', '8', '9', '1', '2'], ['6', '7', '2', '1', '9', '5', '3', '4', '8'], ['1', '9', '8', '3', '4', '2', '5', '6', '
7'], ['8', '5', '9', '7', '6', '1', '4', '2', '3'], ['4', '2', '6', '8', '5', '3', '7', '9', '1'], ['7', '1', '3', '9', '2', '4', '8', '5
', '6'], ['9', '6', '1', '5', '3', '7', '2', '8', '4'], ['2', '8', '7', '4', '1', '9', '6', '3', '5'], ['3', '4', '5', '2', '8', '6', '1'
, '7', '9']]
```

4. Write a program to solve a Sudoku puzzle by filling the empty cells.A sudoku solution must satisfy all of the following rules:Each of the digits 1-9 must occur exactly once in each row.Each of the digits 1-9 must occur exactly once in each column.Each of the digits 1-9 must occur exactly once in each of the 9 3x3 sub-boxes of the grid.The '.' character indicates empty cells.

   Example 1:

   Input: board =
   [["5","3",".",".","7",".",".",".","."],
   ["6",".",".","1","9","5",".",".","."],
   [".","9","8",".",".",".",".","6","."],
   ["8",".",".",".","6",".",".",".","3"],
   ["4",".",".","8",".","3",".",".","1"],
   ["7",".",".",".","2",".",".",".","6"],
   [".","6",".",".",".",".","2","8","."],
   [".",".",".","4","1","9",".",".","5"],
   [".",".",".",".","8",".",".","7","9"]]

   Output:
   [["5","3","4","6","7","8","9","1","2"],
   ["6","7","2","1","9","5","3","4","8"],
   ["1","9","8","3","4","2","5","6","7"],
   ["8","5","9","7","6","1","4","2","3"],
   ["4","2","6","8","5","3","7","9","1"],
   ["7","1","3","9","2","4","8","5","6"],
   ["9","6","1","5","3","7","2","8","4"],
   ["2","8","7","4","1","9","6","3","5"],
   ["3","4","5","2","8","6","1","7","9"]]

```python
def solve_sudoku(board):
    def is_valid(board, row, col, num):
        for i in range(9):
            if board[row][i] == num or board[i][col] == num:
                return False
            if board[3 * (row // 3) + i // 3][3 * (col // 3) + i % 3] == num:
                return False
        return True

    def solve(board):
        for row in range(9):
            for col in range(9):
                if board[row][col] == '.':
                    for num in map(str, range(1, 10)):
                        if is_valid(board, row, col, num):
                            board[row][col] = num
                            if solve(board):
                                return True
                            board[row][col] = '.'
                    return False
        return True

    solve(board)
board = [["5","3",".",".","7",".",".",".","."],
         ["6",".",".","1","9","5",".",".","."],
         [".","9","8",".",".",".",".","6","."],
         ["8",".",".",".","6",".",".",".","3"],
         ["4",".",".","8",".","3",".",".","1"],
         ["7",".",".",".","2",".",".",".","6"],
         [".","6",".",".",".",".","2","8","."],
         [".",".",".","4","1","9",".",".","5"],
         [".",".",".",".","8",".",".","7","9"]]

solve_sudoku(board)
print(board)
```

input

```
[['5', '3', '4', '6', '7', '8', '9', '1', '2'], ['6', '7', '2', '1', '9', '5', '3', '4', '8'], ['1', '9', '8', '3', '4', '2', '5', '6', '
7'], ['8', '5', '9', '7', '6', '1', '4', '2', '3'], ['4', '2', '6', '8', '5', '3', '7', '9', '1'], ['7', '1', '3', '9', '2', '4', '8', '5
', '6'], ['9', '6', '1', '5', '3', '7', '2', '8', '4'], ['2', '8', '7', '4', '1', '9', '6', '3', '5'], ['3', '4', '5', '2', '8', '6', '1'
, '7', '9']]
```

5. You are given an integer array nums and an integer target. You want to build an expression out of nums by adding one of the symbols '+' and '-' before each integer in nums and then concatenate all the integers.For example, if nums = [2, 1], you can add a '+' before 2 and a '-' before 1 and concatenate them to build the expression "+2-1" Return the number of different expressions that you can build, which evaluates to target.

   Example 1:
   Input: nums = [1,1,1,1,1], target = 3
   Output: 5
   Explanation: There are 5 ways to assign symbols to make the sum of nums be target 3.
   -1 + 1 + 1 + 1 + 1 = 3
   +1 - 1 + 1 + 1 + 1 = 3
   +1 + 1 - 1 + 1 + 1 = 3
   +1 + 1 + 1 - 1 + 1 = 3
   +1 + 1 + 1 + 1 - 1 = 3
   Example 2:
   Input: nums = [1], target = 1
   Output: 1

```python
def findTargetSumWays(nums, target):
    from collections import defaultdict
    def dfs(index, current_sum):
        if index == len(nums):
            return 1 if current_sum == target else 0
        return dfs(index + 1, current_sum + nums[index]) + dfs(index + 1, current_sum - nums[index])
    return dfs(0, 0)
print(findTargetSumWays([1, 1, 1, 1, 1], 3))
print(findTargetSumWays([1], 1))
```

input

```
5
1
```

6. Given an array of integers arr, find the sum of min(b), where b ranges over every (contiguous) subarray of arr. Since the answer may be large, return the answer modulo 109 + 7.

   Example 1:
   Input: arr = [3,1,2,4]
   Output: 17
   Explanation:
   Subarrays are [3], [1], [2], [4], [3,1], [1,2], [2,4], [3,1,2], [1,2,4], [3,1,2,4].
   Minimums are 3, 1, 2, 4, 1, 1, 2, 1, 1, 1.
   Sum is 17.
   Example 2:
   Input: arr = [11,81,94,43,3]
   Output: 444

```python
def sum_of_min_subarrays(arr):
    MOD = 10**9 + 7
    n = len(arr)
    total_sum = 0
    stack = []

    for i in range(n):
        while stack and arr[stack[-1]] > arr[i]:
            j = stack.pop()
            k = stack[-1] if stack else -1
            total_sum += arr[j] * (i - j) * (j - k) % MOD
            total_sum %= MOD
        stack.append(i)

    while stack:
        j = stack.pop()
        k = stack[-1] if stack else -1
        total_sum += arr[j] * (n - j) * (j - k) % MOD
        total_sum %= MOD

    return total_sum

print(sum_of_min_subarrays([3, 1, 2, 4]))
print(sum_of_min_subarrays([11, 81, 94, 43, 3]))
```

input

```
17
444
```

7. Given an array of distinct integers candidates and a target integer target, return a list of all unique combinations of candidates where the chosen numbers sum to target. You may return the combinations in any order.The same number may be chosen from candidates an unlimited number of times. Two combinations are unique if the frequency of at least one of the chosen numbers is different.The test cases are generated such that the number of unique combinations that sum up to target is less than 150 combinations for the given input.

   Example 1:
   Input: candidates = [2,3,6,7], target = 7
   Output: [[2,2,3],[7]]
   Explanation:
   2 and 3 are candidates, and 2 + 2 + 3 = 7. Note that 2 can be used multiple times.
   7 is a candidate, and 7 = 7.
   These are the only two combinations.
   Example 2:
   Input: candidates = [2,3,5], target = 8
   Output: [[2,2,2,2],[2,3,3],[3,5]]

```python
def combinationSum(candidates, target):
    result = []

    def backtrack(remaining, combo, start):
        if remaining == 0:
            result.append(list(combo))
            return
        elif remaining < 0:
            return

        for i in range(start, len(candidates)):
            combo.append(candidates[i])
            backtrack(remaining - candidates[i], combo, i)
            combo.pop()

    backtrack(target, [], 0)
    return result


candidates1 = [2, 3, 6, 7]
target1 = 7
print(combinationSum(candidates1, target1))  # Output: [[2, 2, 3], [7]]
candidates2 = [2, 3, 5]
target2 = 8
print(combinationSum(candidates2, target2))  # Output: [[2, 2, 2, 2], [2, 3, 3], [3, 5]]
```

input

```
[[2, 2, 3], [7]]
[[2, 2, 2, 2], [2, 3, 3], [3, 5]]
```

8. Given a collection of candidate numbers (candidates) and a target number (target), find all unique combinations in candidates where the candidate numbers sum to target. Each number in candidates may only be used once in the combination. The solution set must not contain duplicate combinations.

   Example 1:
   Input: candidates = [10,1,2,7,6,1,5], target = 8
   Output:
   [
   [1,1,6],
   [1,2,5],
   [1,7],
   [2,6]
   ]
   Example 2:
   Input: candidates = [2,5,2,1,2], target = 5
   Output:
   [
   [1,2,2],
   [5]
   ]

```python
def combination_sum2(candidates, target):
    def backtrack(start, path, target):
        if target == 0:
            result.append(path)
            return
        for i in range(start, len(candidates)):
            if i > start and candidates[i] == candidates[i - 1]:
                continue
            if candidates[i] > target:
                break
            backtrack(i + 1, path + [candidates[i]], target - candidates[i])

    candidates.sort()
    result = []
    backtrack(0, [], target)
    return result

# Example 1
candidates1 = [10, 1, 2, 7, 6, 1, 5]
target1 = 8
print(combination_sum2(candidates1, target1))

# Example 2
candidates2 = [2, 5, 2, 1, 2]
target2 = 5
print(combination_sum2(candidates2, target2))
```

input

```
[[1, 1, 6], [1, 2, 5], [1, 7], [2, 6]]
[[1, 2, 2], [5]]
```

9. Given an array nums of distinct integers, return all the possible permutations. You can return
the answer in any order.

Example 1:

Input: nums = [1,2,3]

Output: [[1,2,3],[1,3,2],[2,1,3],[2,3,1],[3,1,2],[3,2,1]]

Example 2:

Input: nums = [0,1]

Output: [[0,1],[1,0]]

Example 3:

Input: nums = [1]

Output: [[1]]

```python
from itertools import permutations

def permute(nums):
    return [list(p) for p in permutations(nums)]

print(permute([1, 2, 3]))
print(permute([0, 1]))
print(permute([1]))

```

```
[[1, 2, 3], [1, 3, 2], [2, 1, 3], [2, 3, 1], [3, 1, 2], [3, 2, 1]]
[[0, 1], [1, 0]]
[[1]]
```

10. Given a collection of numbers, nums, that might contain duplicates, return all possible unique
permutations in any order.
Example 1:
Input: nums = [1,1,2]
Output:
[[1,1,2],
[1,2,1],
[2,1,1]]
Example 2:
Input: nums = [1,2,3]
Output: [[1,2,3],[1,3,2],[2,1,3],[2,3,1],[3,1,2],[3,2,1]]

```python
from typing import List

def permuteUnique(nums: List[int]) -> List[List[int]]:
    def backtrack(start):
        if start == len(nums):
            result.append(nums[:])
            return
        seen = set()
        for i in range(start, len(nums)):
            if nums[i] in seen:
                continue
            seen.add(nums[i])
            nums[start], nums[i] = nums[i], nums[start]
            backtrack(start + 1)
            nums[start], nums[i] = nums[i], nums[start]

    result = []
    nums.sort()
    backtrack(0)
    return result
print(permuteUnique([1, 1, 2]))
print(permuteUnique([1, 2, 3]))

```

```
[[1, 1, 2], [1, 2, 1], [2, 1, 1]]
[[1, 2, 3], [1, 3, 2], [2, 1, 3], [2, 3, 1], [3, 2, 1], [3, 1, 2]]
```