```python
def print_solution(queens, N):
    for i in range(N):
        row = ""
        for j in range(N):
            if queens[i] == j:
                row += 'Q '
            else:
                row += '. '
        print(row)
    print("\n")
def is_safe(queens, row, col):
    for i in range(row):
        if queens[i] == col or abs(queens[i] - col) == row - i:
            return False
    return True
def solve_nqueens(N):
    queens = [-1] * N
    solutions = []
    row = 0
    col = 0
    while row >= 0:
        while col < N:
            if is_safe(queens, row, col):
                queens[row] = col
                if row == N - 1:
                    solutions.append(queens[:])
                    col = N
                else:
                    row += 1
                    col = 0
                    break
            else:
                col += 1
        else:
            row -= 1
            if row >= 0:
                col = queens[row] + 1
    return solutions
def show_solutions_for_nqueens(N):
    solutions = solve_nqueens(N)
    print(f"Solutions for {N}-Queens problem:")
    for sol in solutions:
        print_solution(sol, N)
show_solutions_for_nqueens(4)
```

```
Solutions for 4-Queens problem:
. Q . .
. . . Q
Q . . .
. . Q .


. . Q .
Q . . .
. . . Q
. Q . .


=== Code Execution Successful ===
```

```python
def print_board(queens, N, M):
    for i in range(N):
        row = ""
        for j in range(M):
            if queens[i] == j:
                row += 'Q '
            else:
                row += '. '
        print(row)
    print("\n")
def is_safe(queens, row, col, N, M):
    for i in range(row):
        if queens[i] == col or abs(queens[i] - col) == row - i:
            return False
    return True
def solve_nqueens(N, M):
    queens = [-1] * N
    solutions = []
    row = 0
    col = 0
    while row >= 0:
        while col < M:
            if is_safe(queens, row, col, N, M):
                queens[row] = col
                if row == N - 1:
                    solutions.append(queens[:])
                    col = M
                else:
                    row += 1
                    col = 0
                    break
            else:
                col += 1
        else:
            row -= 1
            if row >= 0:
                col = queens[row] + 1
    return solutions
solutions = solve_nqueens(8, 10)
print("Solutions for 8x10 board:")
for sol in solutions:
    print_board(sol, 8,10)
```

```
Solutions for 8x10 board:
Q . . . .
. . Q . .
. . . . Q
. Q . . .


Q . . . .
. . Q . .
. . . . Q
. . . . .


Q . . . .
. . Q . .
. . . . Q
. . . . .


Q . . . .
. . Q . .
. . . . Q
. . . . .


Q . . . .
. . Q . .
. . . . Q
. . . . .


Q . . . .
. . Q . .
. . . . Q
. . . . .


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

main.py

Share Run

```python
def print_board(board):
    for row in board:
        print(" ".join(row))
    print("\n")
def is_valid(board, row, col, num):
    for i in range(9):
        if board[row][i] == num:
            return False
    for i in range(9):
        if board[i][col] == num:
            return False
    start_row = (row // 3) * 3
    start_col = (col // 3) * 3
    for i in range(3):
        for j in range(3):
            if board[start_row + i][start_col + j] == num:
                return False
    return True
def solve_sudoku(board):
    for row in range(9):
        for col in range(9):
            if board[row][col] == '.':
                for num in '123456789':
                    if is_valid(board, row, col, num):
                        board[row][col] = num
                        if solve_sudoku(board):
                            return True
                        board[row][col] = '.'
                return False
    return True
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("Solved Sudoku:")
print_board(board)

```

Output

```
Solved Sudoku:
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


=== Code Execution Successful ===
```

main.py

Share Run

```python
def print_board(board):
    for row in board:
        print(" ".join(row))
    print("\n")
def is_valid(board, row, col, num):
    for i in range(9):
        if board[row][i] == num:
            return False
    for i in range(9):
        if board[i][col] == num:
            return False
    start_row = (row // 3) * 3
    start_col = (col // 3) * 3
    for i in range(3):
        for j in range(3):
            if board[start_row + i][start_col + j] == num:
                return False
    return True
def solve_sudoku(board):
    for row in range(9):
        for col in range(9):
            if board[row][col] == '.':
                for num in '123456789':
                    if is_valid(board, row, col, num):
                        board[row][col] = num
                        if solve_sudoku(board):
                            return True
                        board[row][col] = '.'
                return False
    return True
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("Solved Sudoku:")
print_board(board)
```

Output

```
Solved Sudoku:
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


=== Code Execution Successful ===
```

main.py

Share Run

```python
def count_ways(nums, index, current_sum, target):
    if index == len(nums):
        if current_sum == target:
            return 1
        else:
            return 0
    add = count_ways(nums, index + 1, current_sum + nums[index], target)
    subtract = count_ways(nums, index + 1, current_sum - nums[index], target)
    return add + subtract
def find_target_sum_ways(nums, target):
    return count_ways(nums, 0, 0, target)
nums = [1, 1, 1, 1, 1]
target = 3
result = find_target_sum_ways(nums, target)
print("Number of ways to reach target:", result)
```

Output

```
Number of ways to reach target: 5

=== Code Execution Successful ===
```

main.py

Share Run

```python
def find_combinations(candidates, target):
    result = []
    combination = []
    def backtrack(start, remaining_target):
        if remaining_target == 0:
            result.append(combination[:])
            return
        for i in range(start, len(candidates)):
            if i > start and candidates[i] == candidates[i-1]:
                continue
            candidate = candidates[i]
            if candidate > remaining_target:
                continue
            combination.append(candidate)
            backtrack(i + 1, remaining_target - candidate)
            combination.pop()
    candidates.sort()
    backtrack(0, target)
    return result
candidates = [10, 1, 2, 7, 6, 1, 5]
target = 8
result = find_combinations(candidates, target)
print("Unique combinations that sum to target:", result)
```

Output

```
Unique combinations that sum to target: [[1, 1, 6], [1, 2, 5], [1, 7], [2, 6]]

=== Code Execution Successful ===
```

main.py

Share Run

Output

```python
MOD = 10**9 + 7
def sum_of_subarray_mins(arr):
    n = len(arr)
    left = [0] * n
    right = [0] * n
    stack = []
    for i in range(n):
        count = 1
        while stack and arr[stack[-1]] > arr[i]:
            count += left[stack.pop()]
        left[i] = count
        stack.append(i)
    stack = []
    for i in range(n-1, -1, -1):
        count = 1
        while stack and arr[stack[-1]] >= arr[i]:
            count += right[stack.pop()]
        right[i] = count
        stack.append(i)
    result = 0
    for i in range(n):
        result = (result + arr[i] * left[i] * right[i]) % MOD
    return result
arr = [3, 1, 2, 4]
result = sum_of_subarray_mins(arr)
print("Sum of subarray minimums:", result)

```

```
Sum of subarray minimums: 17

=== Code Execution Successful ===
```

main.py

Share Run

```python
def find_permutations(nums):
    result = []
    permutation = []
    n = len(nums)
    used = [False] * n
    def backtrack():
        if len(permutation) == n:
            result.append(permutation[:])
            return
        for i in range(n):
            if not used[i]:
                used[i] = True
                permutation.append(nums[i])
                backtrack()
                permutation.pop()
                used[i] = False
    backtrack()
    return result
nums = [1, 2, 3]
result = find_permutations(nums)
print("All permutations:", result)
```

Output

```
All permutations: [[1, 2, 3], [1, 3, 2], [2, 1, 3], [2, 3, 1], [3, 1, 2], [3, 2, 1]]

=== Code Execution Successful ===
```

main.py

Share Run

```python
def find_unique_permutations(nums):
    result = []
    permutation = []
    n = len(nums)
    used = [False] * n
    def backtrack():
        if len(permutation) == n:
            result.append(permutation[:])
            return
        for i in range(n):
            if used[i] or (i > 0 and nums[i] == nums[i-1] and not used[i-1]):
                continue
            used[i] = True
            permutation.append(nums[i])
            backtrack()
            permutation.pop()
            used[i] = False
    nums.sort()
    backtrack()
    return result
nums = [1, 1, 2]
result = find_unique_permutations(nums)
print("Unique permutations:", result)

```

Output

Cle

```
All permutations: [[1, 2, 3], [1, 3, 2], [2, 1, 3], [2, 3, 1], [3, 1, 2], [3, 2, 1]]

=== Code Execution Successful ===
```

main.py

Share Run

```python
def find_combinations(candidates, target):
    result = []
    combination = []
    def backtrack(start, remaining_target):
        if remaining_target == 0:
            result.append(combination[:])
            return
        for i in range(start, len(candidates)):
            candidate = candidates[i]
            if candidate > remaining_target:
                continue
            combination.append(candidate)
            backtrack(i, remaining_target - candidate)
            combination.pop()
    backtrack(0, target)
    return result
candidates = [2, 3, 6, 7]
target = 7
result = find_combinations(candidates, target)
print("Combinations that sum to target:", result)

```

Output

```
Combinations that sum to target: [[2, 2, 3], [7]]

=== Code Execution Successful ===
```