main.py

Share Run

Output Clear

```python
def sum_of_subarray_mins(arr):
    MOD, n = 10**9 + 7, len(arr)
    prev_less, next_less, stack = [-1] * n, [n] * n, []
    for i in range(n):
        while stack and arr[stack[-1]] >= arr[i]: stack.pop()
        prev_less[i] = stack[-1] if stack else -1
        stack.append(i)
    stack.clear()
    for i in range(n - 1, -1, -1):
        while stack and arr[stack[-1]] > arr[i]: stack.pop()
        next_less[i] = stack[-1] if stack else n
        stack.append(i)
    return sum(arr[i] * (i - prev_less[i]) * (next_less[i] - i)
        for i in range(n)) % MOD
print(sum_of_subarray_mins([3, 1, 2, 4]))

```

```
17

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

main.py

Share Run

```python
def solve_sudoku(board):
    def valid(n, r, c):
        return all(n not in (board[r][i], board[i][c]) for i in
            range(9)) and \
                all(n != board[i][j] for i in range(r//3*3, r//3
                    *3+3) for j in range(c//3*3, c//3*3+3))
    for r in range(9):
        for c in range(9):
            if board[r][c] == '.':
                for n in map(str, range(1, 10)):
                    if valid(n, r, c):
                        board[r][c] = n
                        if solve_sudoku(board): return True
                        board[r][c] = '.'
                return False
    return True
board = [
    ["5","3",".",".","7",".",".",".",".","."],
    ["6",".",".","1","9","5",".",".",".","."],
    [".","9","8",".",".",".",".",".",".","6",".","."],
```

Output Clear

```
['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

Output Clear

```python
candidates, target = [2, 3, 6, 7], 7
res, stack = [], [(0, [], 0)]
while stack:
    start, path, total = stack.pop()
    if total == target:
        res.append(path)
        continue
    for i in range(start, len(candidates)):
        if total + candidates[i] <= target:
            stack.append((i, path + [candidates[i]], total +
                candidates[i]))
print(res)
```

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

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

main.py

Share Run

Output Clear

```python
def solve_sudoku(board):
    def valid(n, r, c):
        return all(n not in (board[r][i], board[i][c]) for i in range(9)) and \
               all(n != board[i][j] for i in range(r//3*3, r//3*3+3) for j in range(c//3*3, c//3*3+3))
    for r in range(9):
        for c in range(9):
            if board[r][c] == '.':
                for n in map(str, range(1, 10)):
                    if valid(n, r, c):
                        board[r][c] = n
                        if solve_sudoku(board): return True
                        board[r][c] = '.'
                return False
    return True
board = [
    ["5","3",".",".","7",".",".",".",".","."],
    ["6",".",".","1","9","5",".",".",".","."],
    [".","9","8",".",".",".",".",".",".","6",".","."],
```

```
['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

Output Clear

```python
def solve_n_queens(M, N, obstacles=[], restricted=[]):
    solutions, board = [], [["."] * N for _ in range(M)]
    def place(row=0):
        if row == M:
            solutions.append(["".join(r) for r in board])
            return
        for col in (restricted if row == 0 and restricted else
            range(N)):
            if (row, col) not in obstacles and all(
                board[i][col] != 'Q' and
                (col - (row - i) < 0 or board[i][col - (row - i
                    )] != 'Q') and
                (col + (row - i) >= N or board[i][col + (row -
                    i)] != 'Q')
                for i in range(row)):
                board[row][col] = 'Q'
                place(row + 1)
                board[row][col] = '.'
    place()
    return solutions
```

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

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

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

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

main.py

Share Run

```python
candidates, target = [10, 1, 2, 7, 6, 1, 5], 8
candidates.sort()
res, stack = [], [(0, [], 0)]
while stack:
    start, path, total = stack.pop()
    if total == target:
        res.append(path)
        continue
    for i in range(start, len(candidates)):
        if i > start and candidates[i] == candidates[i - 1]:
            continue
        if total + candidates[i] <= target:
            stack.append((i + 1, path + [candidates[i]], total +
                candidates[i]))
print(res)
```

Output Clear

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

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

main.py

Share Run

```
                            board[r][c] = n
                            if solve_sudoku(board): return True
                            board[r][c] = '.'
                    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)
for row in board:
    print(row)
```

Output Clear

```
['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

Output Clear

```python
nums = [1, 1, 2]
nums.sort()
res, stack = [], [(nums, [])]
while stack:
    remaining, path = stack.pop()
    if not remaining:
        res.append(path)
        continue
    for i in range(len(remaining)):
        if i > 0 and remaining[i] == remaining[i - 1]:
            continue
        stack.append((remaining[:i] + remaining[i + 1:], path +
            [remaining[i]]))
print(res)
```

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

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

main.py

Share Run

```python
def find_ways(nums, target):
    def backtrack(index, total):
        if index == len(nums):
            return 1 if total == target else 0
        return backtrack(index + 1, total + nums[index]) + \
               backtrack(index + 1, total - nums[index])
    return backtrack(0, 0)
nums1, target1 = [1, 1, 1, 1, 1], 3
print(find_ways(nums1, target1))
```

Output Clear

```
5

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

main.py

Share Run

Output Clear

```
            solutions.append(["".join(r) for r in board])
            return
        for col in (restricted if row == 0 and restricted else
            range(N)):
            if (row, col) not in obstacles and all(
                board[i][col] != 'Q' and
                (col - (row - i) < 0 or board[i][col - (row - i
                    )] != 'Q') and
                (col + (row - i) >= N or board[i][col + (row -
                    i)] != 'Q')
                for i in range(row)):
                board[row][col] = 'Q'
                place(row + 1)
                board[row][col] = '.'
    place()
    return solutions
for sol in solve_n_queens(5, 5, obstacles=[(1, 1), (3, 3)]):
    print("\n".join(sol), "\n")

```

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

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

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

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

main.py

Share Run

```
                            board[r][c] = n
                            if solve_sudoku(board): return True
                            board[r][c] = '.'
                    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)
for row in board: print(row)

```

Output Clear

```
['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
N = 4
board = [["."] * N for _ in range(N)]
solutions = []
def solve(row=0):
    if row == N:
        solutions.append(["".join(r) for r in board])
        return
    for col in range(N):
        if all(board[r][col] != 'Q' and
               (col - (row - r) < 0 or board[r][col - (row - r
                   )] != 'Q') and
               (col + (row - r) >= N or board[r][col + (row - r
                   )] != 'Q')
               for r in range(row)):
            board[row][col] = 'Q'
            solve(row + 1)
            board[row][col] = '.'
solve()
for solution in solutions:
    print("\n".join(solution), "\n")
```

Output

Clear

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

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


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

main.py

Share Run

```python
nums = [1, 2, 3]
res, stack = [], [(nums, [])]
while stack:
    remaining, path = stack.pop()
    if not remaining:
        res.append(path)
        continue
    for i in range(len(remaining)):
        stack.append((remaining[:i] + remaining[i+1:], path +
            [remaining[i]]))
print(res)
```

Output Clear

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

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