main.py

Run

```python
N = 4
board = [
    [1, 0, 0, 0],
    [0, 0, 1, 0],
    [0, 1, 0, 0],
    [0, 0, 0, 1],
]

for row in board:
    print(" ".join("Q" if x else "." for x in
        row))
```

Output

Clear

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

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

main.py | Output

```python
def is_safe(board, row, col):
    for i in range(col):
        if board[row][i] == 1:
            return False
    for i, j in zip(range(row, -1, -1),
        range(col, -1, -1)):
        if board[i][j] == 1:
            return False
    for i, j in zip(range(row, len
        (board)), range(col, -1, -1)):
        if board[i][j] == 1:
            return False
    return True

def solve_n_queens(board, col):
    if col >= len(board[0]):
        return True
    for i in range(len(board)):
        if is_safe(board, i, col):
            board[i][col] = 1
            if solve_n_queens(board,
                col + 1):
                return True
            board[i][col] = 0
    return False

# Example usage for an 8x10 board
board = [[0 for _ in range(10)] for _
    in range(8)]
solve_n_queens(board, 0)
```

Run

main.py

Run

```python
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"]]

def is_valid(board, row, col, num):
    for i in range(9):
        if board[row][i] == num or
            board[i][col] == num or board[row
            //3*3 + i//3][col//3*3 + i%3] ==
            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 map(str, range(1, 10
                    )):
                    if is_valid(board, row, col
                        , num):
                        board[row][col] = num
                        if solve_sudoku(board):
                            return True
                        board[row][col] = '.'
                return False
    return True

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

Run

Output

```python
nums = [1, 1, 1, 1, 1]
target = 3

memo = {}

def dfs(i, current_sum):
    if i == len(nums):
        return 1 if current_sum == target else 0
    if (i, current_sum) in memo:
        return memo[(i, current_sum)]

    add = dfs(i + 1, current_sum + nums[i])
    subtract = dfs(i + 1, current_sum - nums[i])

    memo[(i, current_sum)] = add + subtract
    return memo[(i, current_sum)]

result = dfs(0, 0)
print(result)  # Output will be 5

```

```
5

=== Code Execution
```

Programiz
Python Online Compiler

Progran
PRO ❯

main.py

Run

Output

Clear

```python
arr = [3, 1, 2, 4]
mod = 10**9 + 7
n = len(arr)
stack = []
prev = [0] * n
next = [0] * n
for i in range(n):
    while stack and arr[stack[-1]] >= arr[i]:
        stack.pop()
    prev[i] = i - stack[-1] if stack else i + 1
    stack.append(i)
stack = []
for i in range(n-1, -1, -1):
    while stack and arr[stack[-1]] > arr[i]:
        stack.pop()
    next[i] = stack[-1] - i if stack else n - i
    stack.append(i)
result = sum(arr[i] * prev[i] * next[i] for i
    in range(n)) % mod
print(result)
```

```
17

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

main.py

Run

Output

```python
def combinationSum(candidates, target):
    def backtrack(remaining, combination, start
        ):
        if remaining == 0:
            result.append(list(combination))
            return
        elif remaining < 0:
            return
        for i in range(start, len(candidates)):
            combination.append(candidates[i])
            backtrack(remaining - candidates[i]
                , combination, i)
            combination.pop()
    result = []
    candidates.sort()
    backtrack(target, [], 0)
    return result
candidates = [2, 3, 6, 7]
target = 7
print(combinationSum(candidates, target))
```

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

=== Code Execution Successfu
```

main.py

Run

Output

Clear

```python
candidates = [10, 1, 2, 7, 6, 1, 5]
target = 8
candidates.sort()
res = []

def backtrack(start, path, target):
    if target == 0:
        res.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])

backtrack(0, [], target)
print(res)
```

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

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

main.py

Run

```python
import itertools
nums = [1, 2, 3]
result = list(itertools.permutations(nums))
print(result)

```

Output

Clear

```
[(1, 2, 3), (1, 3, 2), (2, 1, 3), (2, 3, 1), (3, 1, 2
    ), (3, 2, 1)]

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

main.py

Run

Output

Clear

```python
from itertools import permutations
nums = [1, 1, 2]
unique_permutations = list(set(permutations(nums)))
print(unique_permutations)
```

```
[(1, 2, 1), (2, 1, 1), (1, 1, 2)]

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