1. Remove Element

```
def removeElement(nums, val):
    k = 0
    for i in range(len(nums)):
        if nums[i] != val:
            nums[k] = nums[i]
            k += 1
    return k


# Example usage
nums = [3, 2, 2, 3]
val = 3
k = removeElement(nums, val)
print(k)  # Output: 2
print(nums[:k])  # Output: [2, 2]
```

2. Sudoku Solver

```
def solveSudoku(board):
    def is_valid(board, row, col, num):
        # Check the row
        for i in range(9):
            if board[row][i] == num:
                return False
        # Check the column
        for i in range(9):
            if board[i][col] == num:
                return False
        # Check the 3x3 box
        start_row, start_col = 3 * (row // 3), 3 * (col // 3)
        for i in range(start_row, start_row + 3):
            for j in range(start_col, start_col + 3):
                if board[i][j] == 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 '123456789':
                        if is_valid(board, row, col, num):
                            board[row][col] = num
                            if solve(board):
                                return True
                            board[row][col] = '.'  # Backtrack
                    return False
        return True

    solve(board)

# Example usage
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"]]

solveSudoku(board)
for row in board:
    print(row)
```

3. Count and Say

```
def countAndSay(n):
    if n == 1:
        return "1"
```

```
    def next_sequence(sequence):
        result = []
        i = 0
        while i < len(sequence):
            count = 1
            while i + 1 < len(sequence) and sequence[i] == sequence[i + 1]:
                i += 1
                count += 1
            result.append(str(count) + sequence[i])
            i += 1
        return ''.join(result)

    current_sequence = "1"
    for _ in range(2, n + 1):
        current_sequence = next_sequence(current_sequence)

    return current_sequence

# Example usage
n = 1
print(countAndSay(n))  # Output: "1"
```

4. Combination Sum

```
def combinationSum(candidates, target):
    candidates.sort()  # Optional: sort the candidates to help with early termination
    result = []

    def backtrack(remaining, combination, start):
        if remaining == 0:
            result.append(list(combination))  # Found a valid combination
            return
        elif remaining < 0:
            return  # Exceeded the target, no need to proceed

        for i in range(start, len(candidates)):
            candidate = candidates[i]
```

```
            combination.append(candidate)
            backtrack(remaining - candidate, combination, i)  # Not i + 1 because we can reuse the same elements
            combination.pop()  # Backtrack

    backtrack(target, [], 0)
    return result

# Example usage
candidates = [2, 3, 6, 7]
target = 7
print(combinationSum(candidates, target))  # Output: [[2, 2, 3], [7]]
```

5. Combination Sum II

```
def combinationSum2(candidates, target):
    candidates.sort()  # Sort the candidates to handle duplicates easily
    result = []

    def backtrack(remaining, combination, start):
        if remaining == 0:
            result.append(list(combination))  # Found a valid combination
            return
        elif remaining < 0:
            return  # Exceeded the target, no need to proceed

        for i in range(start, len(candidates)):
            if i > start and candidates[i] == candidates[i - 1]:
                continue  # Skip duplicates
            candidate = candidates[i]
            combination.append(candidate)
            backtrack(remaining - candidate, combination, i + 1)  # Move to the next index
            combination.pop()  # Backtrack

    backtrack(target, [], 0)
    return result
```

```
# Example usage
candidates = [10, 1, 2, 7, 6, 1, 5]
target = 8
print(combinationSum2(candidates, target))
# Output: [[1, 1, 6], [1, 2, 5], [1, 7], [2, 6]]
```

6. Permutations II

```
def permuteUnique(nums):
    def backtrack(path, used):
        if len(path) == len(nums):
            result.append(path[:])
            return

        for i in range(len(nums)):
            if used[i] or (i > 0 and nums[i] == nums[i - 1] and not used[i - 1]):
                continue

            used[i] = True
            path.append(nums[i])
            backtrack(path, used)
            path.pop()
            used[i] = False

    nums.sort()  # Sort the numbers to handle duplicates
    result = []
    used = [False] * len(nums)
    backtrack([], used)
    return result

# Example usage
nums = [1, 1, 2]
print(permuteUnique(nums))
# Output: [[1, 1, 2], [1, 2, 1], [2, 1, 1]]
```

7. Maximum Subarray

```
def maxSubArray(nums):
```

```
    max_current = max_global = nums[0]

    for num in nums[1:]:
        max_current = max(num, max_current + num)
        if max_current > max_global:
            max_global = max_current

    return max_global

# Example usage
nums = [-2, 1, -3, 4, -1, 2, 1, -5, 4]
print(maxSubArray(nums))  # Output: 6
```

8. Permutation Sequence

```
def getPermutation(n, k):
    # Initialize the list of numbers and the result string
    numbers = list(range(1, n + 1))
    result = ""

    # Convert k to zero-indexed
    k -= 1

    # Iterate for each position
    for i in range(n, 0, -1):
        # Determine the factorial of (i-1)
        fact = math.factorial(i - 1)

        # Determine the index of the current digit
        index = k // fact

        # Append the digit at the index to the result
        result += str(numbers[index])

        # Remove the digit from the list
        numbers.pop(index)
```

```
        # Update k
        k %= fact

    return result

# Example usage
n = 3
k = 3
print(getPermutation(n, k))  # Output: "213"
```