Lab test one

1.REMOVE ALL OCCERENCES: **Given an integer array nums and an integer val, remove all occurrences of val in nums in-place. The relative order of the elements may be changed.**

Program:

```
def remove_element(nums, val):
    i = 0
    for j in range(len(nums)):
        if nums[j] != val:
            nums[i] = nums[j]
            i += 1
    return i
nums = [3, 2, 2, 3]
val = 3
new_length = remove_element(nums, val)
print("New length:", new_length)
print("Modified array:", nums[:new_length])
```

time complexity of $O(n)$$O(n)$

2. **Determine if a 9 x 9 Sudoku board is valid. Only the filled cells need to be validated according to the following rules:**

1. **Each row must contain the digits 1-9 without repetition.**
2. **Each column must contain the digits 1-9 without repetition.**
3. **Each of the nine 3 x 3 sub-boxes of the grid must contain the digits 1-9 without repetition.**

Program:

```
def is_valid_sudoku(board):
    def has_duplicates(values):
        seen = set()
        for value in values:
            if value != '.':
                if value in seen:
                    return True
                seen.add(value)
```

```
        return False
    for row in board:
        if has_duplicates(row):
            return False
    for col in zip(*board):
        if has_duplicates(col):
            return False
    for box_row in range(0, 9, 3):
        for box_col in range(0, 9, 3):
            box = [board[r][c] for r in range(box_row, box_row + 3) for c in range(box_col, box_col + 3)]
            if has_duplicates(box):
                return False
    return True
sudoku_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"]
]
print(is_valid_sudoku(sudoku_board))
```

time complexity of this solution is $O(1)$

3. . Sudoku Solver

Write a program to solve a Sudoku puzzle by filling the empty cells.

A sudoku solution must satisfy all of the following rules:

1. **Each of the digits 1-9 must occur exactly once in each row.**
2. **Each of the digits 1-9 must occur exactly once in each column.**

3. **Each of the digits 1-9 must occur exactly once in each of the 9 3x3 sub-boxes of the grid.**

Program:

```
def solve_sudoku(board):
    def is_valid(board, row, col, num):
        for x in range(9):
            if board[row][x] == num:
                return False
        for x in range(9):
            if board[x][col] == num:
                return False
        start_row, start_col = 3 * (row // 3), 3 * (col // 3)
        for i in range(3):
            for j in range(3):
                if board[i + start_row][j + start_col] == 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)
sudoku_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(sudoku_board)

for row in sudoku_board:
    print(row)
```

The worst-case time complexity of this approach is $O(981)$$O(9_{81}$

4. Count and Say

The count-and-say sequence is a sequence of digit strings defined by the recursive formula:

- **countAndSay(1) = "1"**
- **countAndSay(n) is the way you would "say" the digit string from countAndSay(n-1), which is then converted into a different digit string.**

Program:

```
def count_and_say(n):
    if n == 1:
        return "1"
    previous_seq = count_and_say(n - 1)
    result = []
    count = 1
    for i in range(1, len(previous_seq)):
        if previous_seq[i] == previous_seq[i - 1]:
            count += 1
        else:
            result.append(str(count))
```

```
            result.append(previous_seq[i - 1])
            count = 1
    result.append(str(count))
    result.append(previous_seq[-1])

    return ''.join(result)
for i in range(1, 6):
    print(f"countAndSay({i}) = {count_and_say(i)}")
```

the time complexity for generating each term is $O(L(n-1))$$O(L(n-1))$

5. . Combination Sum

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.

Program:

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

the overall time complexity is $O(N\log N+2N)$$O(N\log N+2^N)$

6. Combination Sum II

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.

Note: The solution set must not contain duplicate combinations

Program:

```
def combinationSum2(candidates, target):
    def backtrack(remaining, start, path):
        if remaining == 0:
            result.append(list(path))
            return
        elif remaining < 0:
            return
        for i in range(start, len(candidates)):
            if i > start and candidates[i] == candidates[i - 1]:
                continue  # Skip duplicates
            path.append(candidates[i])
            backtrack(remaining - candidates[i], i + 1, path)
            path.pop()
    result = []
    candidates.sort()
    backtrack(target, 0, [])
    return result
candidates = [10, 1, 2, 7, 6, 1, 5]
target = 8
print(combinationSum2(candidates, target))
```

7. Permutations II

Given a collection of numbers, nums, that might contain duplicates, return *all possible unique permutations in any order.*

Program:

```
def permuteUnique(nums):
```

```
    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
nums = [1, 1, 2]
print(permuteUnique(nums))
```