```
"""
Here is a study sheet for the top 30 Medium level
LeetCode problems that are asked in software
engineering roles:

"""

# LeetCode Medium Level Solutions - Python

# 1. 3Sum - LC 15
"""
Given an integer array nums, return all the triplets
[nums[i], nums[j], nums[k]] such that i != j, i != k, and j != k,
and nums[i] + nums[j] + nums[k] == 0.
Notice that the solution set must not contain duplicate triplets.
"""
def threeSum(nums):
    nums.sort()
    res = []
    for i in range(len(nums) - 2):
        if i > 0 and nums[i] == nums[i-1]:
            continue
        l, r = i + 1, len(nums) - 1
        while l < r:
            s = nums[i] + nums[l] + nums[r]
            if s < 0:
                l += 1
            elif s > 0:
                r -= 1
            else:
                res.append([nums[i], nums[l], nums[r]])
                while l < r and nums[l] == nums[l + 1]: l += 1
                while l < r and nums[r] == nums[r - 1]: r -= 1
                l += 1; r -= 1
    return res

# 2. Container With Most Water - LC 11
"""
You are given an integer array height of length n.
There are n vertical lines drawn such that the two
endpoints of the ith line are (i, 0) and (i, height[i]).
Find two lines that together with the x-axis form a container,
such that the container contains the most water.
Return the maximum amount of water a container can store.
Notice that you may not slant the container.

"""
def maxArea(height):
    l, r = 0, len(height) - 1
    res = 0
```

```
    while l < r:
        res = max(res, min(height[l], height[r]) * (r - l))
        if height[l] < height[r]:
            l += 1
        else:
            r -= 1
    return res

# 3. Product of Array Except Self - LC 238
"""
Given an integer array nums, return an array answer such that
answer[i] is equal to the product of all the elements of nums
except nums[i].
The product of any prefix or suffix of nums is guaranteed to
fit in a 32-bit integer.
You must write an algorithm that runs in O(n) time and without
using the division operation.
"""
def productExceptSelf(nums):
    res = [1] * len(nums)
    prefix = 1
    for i in range(len(nums)):
        res[i] = prefix
        prefix *= nums[i]
    postfix = 1
    for i in reversed(range(len(nums))):
        res[i] *= postfix
        postfix *= nums[i]
    return res

# 4. Subarray Sum Equals K - LC 560
"""
Given an array of integers nums and an integer k,
return the total number of subarrays whose sum equals to k.
A subarray is a contiguous non-empty sequence of elements
within an array.
"""
def subarraySum(nums, k):
    count = 0
    curr_sum = 0
    prefix_sums = {0: 1}
    for num in nums:
        curr_sum += num
        count += prefix_sums.get(curr_sum - k, 0)
        prefix_sums[curr_sum] = prefix_sums.get(curr_sum, 0) + 1
    return count

# 5. Set Matrix Zeroes - LC 73
"""
Given an m x n integer matrix matrix, if an element is 0,
```

```
set its entire row and column to 0's.
You must do it in place.
"""
def setZeroes(matrix):
    rows, cols = len(matrix), len(matrix[0])
    row_zero = False
    for r in range(rows):
        for c in range(cols):
            if matrix[r][c] == 0:
                matrix[0][c] = 0
                if r > 0:
                    matrix[r][0] = 0
                else:
                    row_zero = True
    for r in range(1, rows):
        for c in range(1, cols):
            if matrix[0][c] == 0 or matrix[r][0] == 0:
                matrix[r][c] = 0
    if matrix[0][0] == 0:
        for r in range(rows):
            matrix[r][0] = 0
    if row_zero:
        for c in range(cols):
            matrix[0][c] = 0

# 6. Longest Substring Without Repeating Characters - LC 3
"""
Given a string s, find the length of the longest substring
without duplicate characters.
"""
def lengthOfLongestSubstring(s: str) -> int:
    char_index = {}
    left = 0
    max_len = 0
    for right, char in enumerate(s):
        if char in char_index and char_index[char] >= left:
            left = char_index[char] + 1
        char_index[char] = right
        max_len = max(max_len, right - left + 1)
    return max_len

# 7. Longest Repeating Character Replacement - LC 424
"""
You are given a string s and an integer k. You can choose any
character of the string and change it to any other uppercase
English character. You can perform this operation at most k times.
Return the length of the longest substring containing the same
letter you can get after performing the above operations.
"""
def characterReplacement(s: str, k: int) -> int:
```

```
    count = {}
    max_count = 0
    left = 0
    max_len = 0
    for right in range(len(s)):
        count[s[right]] = count.get(s[right], 0) + 1
        max_count = max(max_count, count[s[right]])
        while (right - left + 1) - max_count > k:
            count[s[left]] -= 1
            left += 1
        max_len = max(max_len, right - left + 1)
    return max_len

# 8. Minimum Window Substring - LC 76
"""
Given two strings s and t of lengths m and n respectively,
return the minimum window substring of s such that every
character in t (including duplicates) is included in the
window. If there is no such substring, return the empty string "".
The testcases will be generated such that the answer is unique.
"""
def minWindow(s: str, t: str) -> str:
    if not t or not s:
        return ""
    dict_t = {}
    for c in t:
        dict_t[c] = dict_t.get(c, 0) + 1
    required = len(dict_t)
    l, r = 0, 0
    formed = 0
    window_counts = {}
    ans = float("inf"), None, None
    while r < len(s):
        c = s[r]
        window_counts[c] = window_counts.get(c, 0) + 1
        if c in dict_t and window_counts[c] == dict_t[c]:
            formed += 1
        while l <= r and formed == required:
            c = s[l]
            if r - l + 1 < ans[0]:
                ans = (r - l + 1, l, r)
            window_counts[c] -= 1
            if c in dict_t and window_counts[c] < dict_t[c]:
                formed -= 1
            l += 1
        r += 1
    return "" if ans[0] == float("inf") else s[ans[1]:ans[2]+1]

# 9. Permutation in String - LC 567
"""
```

```
Given two strings s1 and s2, return true if s2
contains a permutation of s1, or false otherwise.
In other words, return true if one of s1's permutations
is the substring of s2.
"""
def checkInclusion(s1: str, s2: str) -> bool:
    from collections import Counter
    len1, len2 = len(s1), len(s2)
    if len1 > len2:
        return False
    count_s1 = Counter(s1)
    count_s2 = Counter()
    for i in range(len2):
        count_s2[s2[i]] += 1
        if i >= len1:
            if count_s2[s2[i - len1]] == 1:
                del count_s2[s2[i - len1]]
            else:
                count_s2[s2[i - len1]] -= 1
        if count_s1 == count_s2:
            return True
    return False

# 10. Group Anagrams - LC 49
"""
Given an array of strings strs, group the anagrams together.
You can return the answer in any order.
"""
def groupAnagrams(strs):
    from collections import defaultdict
    anagrams = defaultdict(list)
    for s in strs:
        key = tuple(sorted(s))
        anagrams[key].append(s)
    return list(anagrams.values())

# 11. Binary Tree Level Order Traversal - LC 102
"""
Given the root of a binary tree, return the level order traversal of
its nodes' values. (i.e., from left to right, level by level).
"""
def levelOrder(root):
    res = []
    if not root:
        return res
    from collections import deque
    queue = deque([root])
    while queue:
        level = []
        for _ in range(len(queue)):
```

```
            node = queue.popleft()
            level.append(node.val)
            if node.left:
                queue.append(node.left)
            if node.right:
                queue.append(node.right)
        res.append(level)
    return res

# 12. Construct Binary Tree from Inorder and Postorder Traversal - LC
106
"""
Given two integer arrays inorder and postorder where inorder is the
inorder traversal of a binary tree and postorder is the postorder
traversal of the same tree, construct and return the binary tree.
"""
def buildTree(inorder, postorder):
    if not inorder or not postorder:
        return None
    root_val = postorder.pop()
    root = TreeNode(root_val)
    index = inorder.index(root_val)
    root.right = buildTree(inorder[index+1:], postorder)
    root.left = buildTree(inorder[:index], postorder)
    return root

# 13. Lowest Common Ancestor of a Binary Tree - LC 236
"""
Given a binary tree, find the lowest common ancestor (LCA) of two
given
nodes in the tree.
According to the definition of LCA on Wikipedia: “The lowest common
ancestor is defined between two nodes p and q as the lowest node in
T that has both p and q as descendants (where we allow a node to be
a descendant of itself).”
"""
def lowestCommonAncestor(root, p, q):
    if not root or root == p or root == q:
        return root
    left = lowestCommonAncestor(root.left, p, q)
    right = lowestCommonAncestor(root.right, p, q)
    if left and right:
        return root
    return left if left else right

# 14. Validate Binary Search Tree - LC 98
"""
Given the root of a binary tree, determine if it is a valid binary
search tree (BST).
A valid BST is defined as follows:
```

```
The left subtree of a node contains only nodes with keys less
than the node's key. The right subtree of a node contains only nodes
with keys greater than the node's key. Both the left and right
subtrees
must also be binary search trees.
"""
def isValidBST(root):
    def helper(node, low, high):
        if not node:
            return True
        if node.val <= low or node.val >= high:
            return False
        return helper(node.left, low, node.val) and helper(node.right,
node.val, high)
    return helper(root, float('-inf'), float('inf'))

# 15. Diameter of Binary Tree - LC 543
"""
Given the root of a binary tree, return the length of the diameter
of the tree.
The diameter of a binary tree is the length of the longest path
between
any two nodes in a tree. This path may or may not pass through the
root.
The length of a path between two nodes is represented by the number of
edges between them.
"""
def diameterOfBinaryTree(root):
    diameter = 0
    def depth(node):
        nonlocal diameter
        if not node:
            return 0
        left = depth(node.left)
        right = depth(node.right)
        diameter = max(diameter, left + right)
        return max(left, right) + 1
    depth(root)
    return diameter

# 16. Coin Change - LC 322
"""
You are given an integer array coins representing coins of
different denominations and an integer amount representing a total
amount of money.
Return the fewest number of coins that you need to make up that
amount. If that amount of money cannot be made up by any combination
of the coins, return -1.
You may assume that you have an infinite number of each kind of coin.
"""
```

```
def coinChange(coins, amount):
    dp = [float('inf')] * (amount + 1)
    dp[0] = 0
    for i in range(1, amount+1):
        for c in coins:
            if c <= i:
                dp[i] = min(dp[i], dp[i-c] + 1)
    return dp[amount] if dp[amount] != float('inf') else -1

# 17. House Robber - LC 198
"""
You are a professional robber planning to rob houses along a street.
Each house has a certain amount of money stashed, the only constraint
stopping you from robbing each of them is that adjacent houses have
security systems connected and it will automatically contact the
police if two adjacent houses were broken into on the same night.
Given an integer array nums representing the amount of money of each
house, return the maximum amount of money you can rob tonight without
alerting the police.
"""
def rob(nums):
    prev = curr = 0
    for num in nums:
        prev, curr = curr, max(curr, prev + num)
    return curr

# 18. Partition Equal Subset Sum - LC 416
"""
Given an integer array nums, return true if you can partition the
array
into two subsets such that the sum of the elements in both subsets is
equal or false otherwise.
"""
def canPartition(nums):
    total = sum(nums)
    if total % 2 != 0:
        return False
    target = total // 2
    dp = set([0])
    for num in nums:
        new_dp = dp.copy()
        for t in dp:
            if t + num == target:
                return True
            if t + num < target:
                new_dp.add(t + num)
        dp = new_dp
    return False

# 19. Longest Palindromic Substring - LC 5
```

```
"""
Given a string s, return the longest palindromic substring in s.
"""
def longestPalindrome(s):
    start = end = 0
    for i in range(len(s)):
        len1 = expandAroundCenter(s, i, i)
        len2 = expandAroundCenter(s, i, i+1)
        length = max(len1, len2)
        if length > end - start:
            start = i - (length -1)//2
            end = i + length//2
    return s[start:end+1]

def expandAroundCenter(s, left, right):
    while left >=0 and right < len(s) and s[left] == s[right]:
        left -=1
        right +=1
    return right - left -1

# 20. Unique Paths - LC 62
"""
There is a robot on an m x n grid. The robot is initially located at
the
top-left corner (i.e., grid[0][0]). The robot tries to move to the
bottom-right corner (i.e., grid[m - 1][n - 1]). The robot can only
move
either down or right at any point in time.
Given the two integers m and n, return the number of possible unique
paths
that the robot can take to reach the bottom-right corner.
The test cases are generated so that the answer will be less than or
equal to 2 * 109.
"""
def uniquePaths(m, n):
    dp = [[1]*n for _ in range(m)]
    for i in range(1,m):
        for j in range(1,n):
            dp[i][j] = dp[i-1][j] + dp[i][j-1]
    return dp[m-1][n-1]

# 21. Letter Combinations of a Phone Number - LC 17
"""
Given a string containing digits from 2-9 inclusive, return all
possible
letter combinations that the number could represent. Return the answer
in any order.
A mapping of digits to letters (just like on the telephone buttons) is
given below. Note that 1 does not map to any letters.
"""
```

```
def letterCombinations(digits):
    if not digits:
        return []
    phone =
{"2":"abc","3":"def","4":"ghi","5":"jkl","6":"mno","7":"pqrs","8":"tuv
","9":"wxyz"}
    res = []
    def backtrack(index, path):
        if index == len(digits):
            res.append(path)
            return
        for c in phone[digits[index]]:
            backtrack(index+1, path + c)
    backtrack(0, "")
    return res

# 22. Word Search - LC 79
"""
Given an m x n grid of characters board and a string word,
return true if word exists in the grid.
The word can be constructed from letters of sequentially
adjacent cells, where adjacent cells are horizontally or
vertically neighboring. The same letter cell may not be used
more than once.
"""
def exist(board, word):
    rows, cols = len(board), len(board[0])
    def dfs(r, c, idx):
        if idx == len(word):
            return True
        if r < 0 or c < 0 or r >= rows or c >= cols or board[r][c] !=
word[idx]:
            return False
        temp = board[r][c]
        board[r][c] = "#"
        res = dfs(r+1,c,idx+1) or dfs(r-1,c,idx+1) or dfs(r,c+1,idx+1)
or dfs(r,c-1,idx+1)
        board[r][c] = temp
        return res
    for i in range(rows):
        for j in range(cols):
            if dfs(i,j,0):
                return True
    return False

# 23. Combination Sum - LC 39
"""
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.
The same number may be chosen from candidates an unlimited number of
times. Two combinations are unique if the frequency of at least one
of the chosen numbers is different.
The test cases are generated such that the number of unique
combinations
that sum up to target is less than 150 combinations for the given
input.
"""
def combinationSum(candidates, target):
    res = []
    def backtrack(remain, comb, start):
        if remain == 0:
            res.append(list(comb))
            return
        elif remain < 0:
            return
        for i in range(start, len(candidates)):
            comb.append(candidates[i])
            backtrack(remain - candidates[i], comb, i)
            comb.pop()
    backtrack(target, [], 0)
    return res

# 24. Permutations - LC 46
"""
Given an array nums of distinct integers, return all the possible
permutations. You can return the answer in any order.
"""
def permute(nums):
    res = []
    def backtrack(start=0):
        if start == len(nums):
            res.append(nums[:])
            return
        for i in range(start, len(nums)):
            nums[start], nums[i] = nums[i], nums[start]
            backtrack(start+1)
            nums[start], nums[i] = nums[i], nums[start]
    backtrack()
    return res

# 25. Subsets II - LC 90
"""
Given an integer array nums that may contain duplicates, return all
possible subsets (the power set).
The solution set must not contain duplicate subsets. Return the
solution in any order.
"""
def subsetsWithDup(nums):
```

```
    res = []
    nums.sort()
    def backtrack(start, path):
        res.append(path[:])
        for i in range(start, len(nums)):
            if i > start and nums[i] == nums[i-1]:
                continue
            path.append(nums[i])
            backtrack(i+1, path)
            path.pop()
    backtrack(0, [])
    return res

# 26. Add Two Numbers - LC 2
"""
You are given two non-empty linked lists representing two non-negative
integers. The digits are stored in reverse order, and each of their
nodes contains a single digit. Add the two numbers and return the sum
as a linked list.
You may assume the two numbers do not contain any leading zero, except
the
number 0 itself.
"""
class ListNode:
    def __init__(self, val=0, next=None):
        self.val = val
        self.next = next

def addTwoNumbers(l1, l2):
    dummy = curr = ListNode()
    carry = 0
    while l1 or l2 or carry:
        val1 = (l1.val if l1 else 0)
        val2 = (l2.val if l2 else 0)
        carry, out = divmod(val1 + val2 + carry, 10)
        curr.next = ListNode(out)
        curr = curr.next
        l1 = l1.next if l1 else None
        l2 = l2.next if l2 else None
    return dummy.next

# 27. Reorder List - LC 143
"""
You are given the head of a singly linked-list.
The list can be represented as:
L0 → L1 → … → Ln - 1 → Ln
Reorder the list to be on the following form:
L0 → Ln → L1 → Ln - 1 → L2 → Ln - 2 → …
You may not modify the values in the list's nodes.
Only nodes themselves may be changed.
```

```
"""
def reorderList(head):
    if not head:
        return
    slow, fast = head, head.next
    while fast and fast.next:
        slow = slow.next
        fast = fast.next.next
    second = slow.next
    slow.next = None
    prev = None
    while second:
        nxt = second.next
        second.next = prev
        prev = second
        second = nxt
    first, second = head, prev
    while second:
        tmp1, tmp2 = first.next, second.next
        first.next = second
        second.next = tmp1
        first, second = tmp1, tmp2

# 28. Remove Nth Node From End of List - LC 19
"""
Given the head of a linked list, remove the nth node from the
end of the list and return its head.
"""
def removeNthFromEnd(head, n):
    dummy = ListNode(0, head)
    slow = fast = dummy
    for _ in range(n):
        fast = fast.next
    while fast.next:
        slow = slow.next
        fast = fast.next
    slow.next = slow.next.next
    return dummy.next

# 29. Merge k Sorted Lists - LC 23
"""
You are given an array of k linked-lists lists, each linked-list
is sorted in ascending order.
Merge all the linked-lists into one sorted linked-list and return it.
"""
import heapq
def mergeKLists(lists):
    heap = []
    for i, node in enumerate(lists):
        if node:
```

```
            heapq.heappush(heap, (node.val, i, node))
    dummy = curr = ListNode()
    while heap:
        val, i, node = heapq.heappop(heap)
        curr.next = node
        curr = curr.next
        if node.next:
            heapq.heappush(heap, (node.next.val, i, node.next))
    return dummy.next

# 30. Linked List Cycle II - LC 142
"""
Given the head of a linked list, return the node where the cycle
begins.
If there is no cycle, return null.
There is a cycle in a linked list if there is some node in the list
that
can be reached again by continuously following the next pointer.
Internally,
pos is used to denote the index of the node that tail's next pointer
is
connected to (0-indexed). It is -1 if there is no cycle. Note that pos
is
not passed as a parameter.
Do not modify the linked list.
"""
def detectCycle(head):
    slow = fast = head
    while fast and fast.next:
        slow = slow.next
        fast = fast.next.next
        if slow == fast:
            slow2 = head
            while slow2 != slow:
                slow2 = slow2.next
                slow = slow.next
            return slow
    return None

#----------HARD LEVEL
PROBLEMS----------------------------------------
# Hard Level LeetCode Problem Solutions - Python


# 2. Median of Two Sorted Arrays - LC 4
"""
Given two sorted arrays nums1 and nums2 of size m and n
respectively, return the median of the two sorted arrays.
The overall run time complexity should be O(log (m+n)).
"""
```

```
def findMedianSortedArrays(nums1, nums2):
    if len(nums1) > len(nums2):
        nums1, nums2 = nums2, nums1
    x, y = len(nums1), len(nums2)
    low, high = 0, x
    while low <= high:
        partitionX = (low + high) // 2
        partitionY = (x + y + 1) // 2 - partitionX

        maxX = float('-inf') if partitionX == 0 else nums1[partitionX
- 1]
        maxY = float('-inf') if partitionY == 0 else nums2[partitionY
- 1]

        minX = float('inf') if partitionX == x else nums1[partitionX]
        minY = float('inf') if partitionY == y else nums2[partitionY]

        if maxX <= minY and maxY <= minX:
            if (x + y) % 2 == 0:
                return (max(maxX, maxY) + min(minX, minY)) / 2
            else:
                return max(maxX, maxY)
        elif maxX > minY:
            high = partitionX - 1
        else:
            low = partitionX + 1

# 3. Longest Valid Parentheses - LC 32
"""
Given a string containing just the characters '(' and ')', return the
length
of the longest valid (well-formed) parentheses substring.
"""
def longestValidParentheses(s):
    max_len = 0
    stack = [-1]
    for i, c in enumerate(s):
        if c == '(':
            stack.append(i)
        else:
            stack.pop()
            if not stack:
                stack.append(i)
            else:
                max_len = max(max_len, i - stack[-1])
    return max_len

# 4. Trapping Rain Water - LC 42
"""
Given n non-negative integers representing an elevation map where the
```

```
width of each bar is 1, compute how much water it can trap after
raining.
"""
def trap(height):
    left, right = 0, len(height) -1
    left_max = right_max = 0
    res = 0
    while left < right:
        if height[left] < height[right]:
            if height[left] >= left_max:
                left_max = height[left]
            else:
                res += left_max - height[left]
            left +=1
        else:
            if height[right] >= right_max:
                right_max = height[right]
            else:
                res += right_max - height[right]
            right -= 1
    return res

# 5. Serialize and Deserialize Binary Tree - LC 297
"""
Serialization is the process of converting a data structure or
object into a sequence of bits so that it can be stored in a
file or memory buffer, or transmitted across a network connection
link to be reconstructed later in the same or another computer
environment.
Design an algorithm to serialize and deserialize a binary tree.
There is no restriction on how your serialization/deserialization
algorithm
should work. You just need to ensure that a binary tree can be
serialized
to a string and this string can be deserialized to the original tree
structure.
Clarification: The input/output format is the same as how LeetCode
serializes a binary tree. You do not necessarily need to follow this
format,
so please be creative and come up with different approaches yourself.
"""
class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right

def serialize(root):
    """Encodes a tree to a single string."""
    vals = []
```

```
    def preorder(node):
        if node:
            vals.append(str(node.val))
            preorder(node.left)
            preorder(node.right)
        else:
            vals.append('#')
    preorder(root)
    return ' '.join(vals)

def deserialize(data):
    """Decodes your encoded data to tree."""
    vals = iter(data.split())
    def helper():
        val = next(vals)
        if val == '#':
            return None
        node = TreeNode(int(val))
        node.left = helper()
        node.right = helper()
        return node
    return helper()

# 6. Regular Expression Matching - LC 10
"""
Given an input string s and a pattern p, implement regular
expression matching with support for '.' and '*' where:
'.' Matches any single character
'*' Matches zero or more of the preceding element.
The matching should cover the entire input string (not partial).
"""
def isMatch(s, p):
    m, n = len(s), len(p)
    dp = [[False]*(n+1) for _ in range(m+1)]
    dp[0][0] = True

    for j in range(2, n+1):
        if p[j-1] == '*':
            dp[0][j] = dp[0][j-2]

    for i in range(1, m+1):
        for j in range(1, n+1):
            if p[j-1] == '.' or p[j-1] == s[i-1]:
                dp[i][j] = dp[i-1][j-1]
            elif p[j-1] == '*':
                dp[i][j] = dp[i][j-2]
                if p[j-2] == '.' or p[j-2] == s[i-1]:
                    dp[i][j] = dp[i][j] or dp[i-1][j]
    return dp[m][n]
```

```
# 7. Sliding Window Maximum - LC 239
"""
You are given an array of integers nums, there is a sliding window
of size k which is moving from the very left of the array to the
very right. You can only see the k numbers in the window. Each time
the sliding window moves right by one position.
Return the max sliding window.
"""
from collections import deque
def maxSlidingWindow(nums, k):
    if not nums:
        return []
    q = deque()
    res = []
    for i, num in enumerate(nums):
        while q and nums[q[-1]] < num:
            q.pop()
        q.append(i)
        if q[0] == i - k:
            q.popleft()
        if i >= k - 1:
            res.append(nums[q[0]])
    return res

# 8. N-Queens - LC 51
"""
The n-queens puzzle is the problem of placing n queens on an
n x n chessboard such that no two queens attack each other.
Given an integer n, return all distinct solutions to the n-queens
puzzle. You may return the answer in any order.
Each solution contains a distinct board configuration of the n-queens'
placement, where 'Q' and '.' both indicate a queen and an empty space,
respectively.
"""
def solveNQueens(n):
    res = []
    board = [["."]*n for _ in range(n)]
    cols = set()
    diag1 = set()
    diag2 = set()

    def backtrack(r=0):
        if r == n:
            res.append(["".join(row) for row in board])
            return
        for c in range(n):
            if c in cols or (r+c) in diag1 or (r-c) in diag2:
                continue
            board[r][c] = "Q"
            cols.add(c)
```

```
            diag1.add(r+c)
            diag2.add(r-c)
            backtrack(r+1)
            board[r][c] = "."
            cols.remove(c)
            diag1.remove(r+c)
            diag2.remove(r-c)
    backtrack()
    return res

# 9. Word Ladder II - LC 126
"""
A transformation sequence from word beginWord to word endWord using a
dictionary wordList is a sequence of words
beginWord -> s1 -> s2 -> ... -> sk such that:
Every adjacent pair of words differs by a single letter.
Every si for 1 <= i <= k is in wordList. Note that beginWord does not
need
to be in wordList.
sk == endWord
Given two words, beginWord and endWord, and a dictionary wordList,
return
all the shortest transformation sequences from beginWord to endWord,
or an empty
list if no such sequence exists. Each sequence should be returned as a
list of the words [beginWord, s1, s2, ..., sk].
"""
from collections import defaultdict, deque

def findLadders(beginWord, endWord, wordList):
    wordSet = set(wordList)
    if endWord not in wordSet:
        return []
    layer = {}
    layer[beginWord] = [[beginWord]]

    while layer:
        new_layer = defaultdict(list)
        for word in layer:
            if word == endWord:
                return layer[word]
            for i in range(len(word)):
                for c in 'abcdefghijklmnopqrstuvwxyz':
                    new_word = word[:i] + c + word[i+1:]
                    if new_word in wordSet:
                        new_layer[new_word] += [j + [new_word] for j
in layer[word]]
        wordSet -= set(new_layer.keys())
        layer = new_layer
    return []
```

```
# 10. Edit Distance - LC 72
"""
Given two strings word1 and word2, return the minimum number of
operations required to convert word1 to word2.
You have the following three operations permitted on a word:
Insert a character
Delete a character
Replace a character
"""
def minDistance(word1, word2):
    m, n = len(word1), len(word2)
    dp = [[0]*(n+1) for _ in range(m+1)]

    for i in range(m+1):
        dp[i][0] = i
    for j in range(n+1):
        dp[0][j] = j

    for i in range(1, m+1):
        for j in range(1, n+1):
            if word1[i-1] == word2[j-1]:
                dp[i][j] = dp[i-1][j-1]
            else:
                dp[i][j] = 1 + min(dp[i-1][j],    # delete
                                   dp[i][j-1],    # insert
                                   dp[i-1][j-1])  # replace
    return dp[m][n]

#------MISCELLANEOUS-----------------------------------------

"""
DEPTH FIRST SEARCH
"""
def breadth_first_search(adj_list, start_node):
    parents = [None for v in adj_list]
    parents[start_node] = start_node
    levels = [[start_node]]
    while len(levels[-1])>0:
        levels.append([])
        for u in levels[-2]:
            for v in adj_list[u]:
                if parents[v] is None:
                    parents[v] = u
                    levels[-1].append(v)
    return parents, levels

"""BREADTH FIRST SEARCH"""
def breadth_first_search(adj_list, start_node):
    parents = [None for v in adj_list]
```

```
    parents[start_node] = start_node
    levels = [[start_node]]
    while len(levels[-1])>0:
        levels.append([])
        for u in levels[-2]:
            for v in adj_list[u]:
                if parents[v] is None:
                    parents[v] = u
                    levels[-1].append(v)
    return parents, levels

"""Input/Output"""
def ask_and_square():
    try:
        number = float(input("Please enter a number: "))
        squared = number ** 2
        print(f"The square of {number} is {squared}.")
    except ValueError:
        print("That was not a valid number. Please try again.")

# LeetCode 387: First Unique Character in String
"""
Given a string s, find the first non-repeating character in it
and return its index. If it does not exist, return -1.
"""
def first_uniq_char(s):
    seen = dict()
    for ch in s:
        if ch in seen:
            seen[ch] += 1
        else:
            seen[ch] = 1

    for idx, ch in enumerate(s):
        if seen[ch] == 1:
            return idx
    return -1

# Flatten Nested Array
def flatten_array(array):
    new_array = []
    for entry in array:
        if isinstance(entry,int):
            new_array += [entry]
        else:
            new_array += flatten_array(entry)
    return new_array

# LeetCode 300: Longest Increasing Subsequence
"""
```

```
Given an integer array nums, return the length of the
longest strictly increasing subsequence.
"""
def longest_increasing_subsequence(nums):
    if not nums:
        return 0

    n = len(nums)
    dp = [1]*n

    for i in range(n):
        for j in range(i):
            if nums[j] < nums[i]:
                dp[i] = max(dp[i], dp[j]+1)

    return dp[n-1]

# Read and flatten file
def read_and_flatten_file(filename):
    result = []
    with open(filename, 'r') as f:
        for line in f:
            # Strip whitespace/newline and split by commas
            parts = line.strip().split(',')
            for part in parts:
                if part:  # skip empty strings
                    result.append(int(part))
    return result

# LeetCode 242: Is Valid Anagram
def is_anagram(s, t):
    if len(s)!=len(t):
        return False
    s_dic = dict()
    t_dic = dict()
    for i in range(len(s)):
        if s[i] in s_dic:
            s_dic[s[i]] += 1
        else:
            s_dic[s[i]] = 1
        if t[i] in t_dic:
            t_dic[t[i]] += 1
        else:
            t_dic[t[i]] = 1
    for key in s_dic:
        if key not in t_dic or t_dic[key] != s_dic[key]:
            return False
    return True

# Fibonacci with cache
```

```
def fibonacci(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    prev_two = [0,1]
    for i in range(2, n+1):
        prev_two[0], prev_two[1] = prev_two[1],
prev_two[0]+prev_two[1]
    return prev_two[-1]

# LeetCode 347: Top K Frequent Elements
"""
Given an integer array nums and an integer k, return the k most
frequent
elements. You may return the answer in any order.
"""
def top_k_frequent(nums, k):
    num_count = dict()
    for num in nums:
        if num in num_count:
            num_count[num] += 1
        else:
            num_count[num] = 1
    sort_dic = sorted([(dic_key,num_count[dic_key]) for dic_key in
num_count.keys()], key = lambda x: x[1], reverse=True)
    return [el[0] for el in sort_dic[0:k]]

"""Binary Search"""
def binary_search(sorted_list, target):
    n = len(sorted_list)
    left, right = 0, n
    while left < right:
        mid = (left+right)//2
        if target == sorted_list[mid]:
            return mid
        elif target < sorted_list[mid]:
            right = mid
        else:
            left = mid+1
    return -1
```