1. **Palindrome Number**
   Given an integer x, return true if x is palindrome integer. An integer is a palindrome when it reads the same backward as forward.
   Example:
   Input: x = 121
   Output: true
   Solution: We can revert the number itself, and then compare the number with original number, if they are the same, then the number is a palindrome. However, if the reversed number is larger than int.MAX, we will hit integer overflow problem, to avoid the overflow issue of the reverted number, what if we only revert half of the number? After all, the reverse of the last half of the palindrome should be the same as the first half of the number, if the number is a palindrome.

2. **Remove Duplicates from Sorted Array**
   Given an integer array nums sorted in non-decreasing order, remove the duplicates such that each unique element appears only once. The relative order of the elements should be kept the same.
   Example:
   Input: [0,0,1,1,1,2,2,3,3,4]
   Output: [0,1,2,3,4]
   Solution: Using a map you can keep track of how many times a digit occurs, delete all elements that occur more than once.

3. **Add Strings**
   Given two non-negative integers, num1 and num2 represented as string, return the sum of num1 and num2 as a string.
   Example:
   Input: num1 = "456", num2 = "77"
   Output: "533"
   Solution: The easiest way to solve this problem would be to convert the two integers into an int then back to a string after adding them together. Although it's not specified, I would assume the problem should instead be solved using the school addition way of individually adding each digit one at a time.

4. **Is Subsequence**
   Given two strings s and t, return true if s is a subsequence of t, or false otherwise. A subsequence of a string is a new string that is formed from the original string by deleting some (can be none) of the characters without disturbing the relative positions of the remaining characters. (i.e., "ace" is a subsequence of "abcde" while "aec" is not).
   Example:
   Input: s = "abc", t = "ahbgdc"
   Output: true
   Solution: Go through string s and string t with two different counters. If t[i] = s[j] then shift over one letter in string s. If the end of s has been reached, then s is a subsequence of t.

5. **Valid Parentheses**
   Given a non-empty string s containing just the characters '(', ')', '{', '}', '[' and ']', determine if the input string is valid. An input string is valid if: Open brackets must be closed by the same type of brackets. Open brackets must be closed in the correct order.

Example:
Input: s = "()[]{}"
Output: true
Solution: Iterate through string s, if s is a left parenthesis then store it in a stack. Otherwise if s is a right parenthesis check the stack for a matching left parenthesis and remove it from the stack if it exists. If by the end of the string s the stack is empty, then s contains valid parentheses.

6. **Number of Days Between Two Dates**
Given two dates as strings, their format is YYYY-MM-DD as shown in the examples, count the number of days between two dates.
Example:
Input: date1 = "2020-01-15", date2 = "2019-12-31"
Output: 15
Solution: Convert years and months into days, then calculate the number of days of both dates from the same year. The difference between those two dates is the total number of days between date1 and date2.

7. **Max Consecutive Ones**
Given a binary vector nums, return the maximum number of consecutive 1's in the vector.
Example:
Input: nums = [1,1,0,1,1,1]
Output: 3
Solution: Iterate through the list and each time you see a 1, increase a variable count by 1. If you see a 0 then store count to a vector and reset count. Repeat until the end of the binary input. Once finished, return the max count from the count vector.

8. **Remove All Adjacent Duplicates In String**
Given a string s consisting of lowercase English letters. A duplicate removal consists of choosing two adjacent and equal letters and removing them. We repeatedly make duplicate removals on s until we no longer can. Return the final string after all such duplicate removals have been made. It can be proven that the answer is unique.
Example:
Input: s = "abbaca"
Output: “ca”
Solution: Loop through string, each time s[i] and s[i+1] are equal, remove them then restart loop at beginning and repeat until there is nothing left to delete.

9. **Roman to Integer**
Roman numerals are represented by seven different symbols: I, V, X, L, C, D and M.

Symbol	Value
I	1
V	5
X	10
L	50
C	100
D	500
M	1000

Roman numerals are usually written largest to smallest from left to right. For example, 2 is written as II in Roman numeral, just two ones added together. 12 is written as XII, which is simply X + II. The number 27 is written as XXVII, which is XX + V + II. However, the numeral for four is not IIII. Instead, the number four is written as IV. Because the one is before the five, we subtract it making four. The same principle applies to the number nine, which is written as IX. There are six instances where subtraction is used:
I can be placed before V (5) and X (10) to make 4 and 9.
X can be placed before L (50) and C (100) to make 40 and 90.
C can be placed before D (500) and M (1000) to make 400 and 900.
Given a valid roman numeral, convert it to an integer.
Example:
Input: s = "MCMXCIV"
Output: 1994
Solution: Starting at the right end of the string going left, if the numeral is greater than the previous, add it to sum. If the numeral is less than the previous, subtract it from the sum. The total will equal the whole numeral in integer form.

10. **Integer to Roman**
    Given an integer, convert it to a roman numeral.
    Example:
    Input: num = 1994
    Output: "MCMXCIV"
    Solution: Start with the biggest numeral and see how many times it can divide into the num, continue down the numerals until there are no more numerals that can fit in the integer. Take all the numerals that could be divided into the num and place them from left to right to get the answer.

11. **Best Time to Buy and Sell Stock I**
    Given an array prices where prices[i] is the price of a given stock on the ith day. You want to maximize your profit by choosing a single day to buy one stock and choosing a different day in the future to sell that stock. Return the maximum profit you can achieve from this transaction. If you cannot achieve any profit, return 0.
    Example:
    Input: prices = [7,1,5,3,6,4]
    Output: 5
    Solution: Find the smallest element in the vector, then from the range (smallest, end of vector) find the largest vector. The difference between the smallest and largest will be the max profit.

12. **Best Time to Buy and Sell Stock II**
    Given an array prices where prices[i] is the price of a given stock on the ith day. You want to maximize your profit by choosing a single day to buy one stock and choosing a different day in the future to sell that stock. On each day, you may decide to buy or sell the stock. You may complete any number of transactions. Return the maximum profit you can achieve.
    Example:
    Input: prices = [7,1,5,3,6,4]
    Output: 7

Solution: Iterate through the vector prices checking if the following price is greater than the previous. If so then you can buy then sell and add that profit to maxprofit. Otherwise there is no profit to be made. Continue until end of vector.

13. **Best Time to Buy and Sell Stock with Transaction Fee**
    Given an array prices where prices[i] is the price of a given stock on the ith day, and an integer fee representing a transaction fee. Find the maximum profit you can achieve. You may complete as many transactions as you like, but you need to pay the transaction fee after each transaction.
    Example:
    Input: prices = [7,1,5,3,6,4], fee = 2
    Output: 3
    Solution: Iterate through vector prices looking for the minimum price. If the current price is greater than the minimum price + fee then buy and sell current price and add to profit. Otherwise keep iterating through vector prices until the end. By the end, max profit will contain the most amount of profit possible with the fee.

14. **Two Sum Equals K**
    Given an array of integers nums and an integer k, return if there exist two numbers such that they add up to k.
    Example:
    Input: nums = [3,2,4], k = 6
    Output: True
    Solution: Iterate through the vector nums with a map. Map k - the current value of nums in each iteration, if it exists then two numbers exist that add up to k.

15. **Subarray Sum Equals K**
    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.
    Example:
    Input: nums = [1,2,3], k = 3
    Output: 2
    Solution: Iterate through the vector nums, in every iteration sum up all the values from i to the end of the vector nums. If the sum = k at any point during this then increment answer by 1. Continue until the end of nums.

16. **Subarray Sums Divisible by K**
    Given an integer array nums and an integer k, return the number of non-empty subarrays that have a sum divisible by k. A subarray is a contiguous non-empty sequence of elements within an array.
    Example:
    Input: nums = [4,5,0,-2,-3,1], k = 5
    Output: 7
    Solution: Iterate through the vector mapping the cumulative sum of each element. Divide each sum by k and if the remainder exists in the sum map, then add it to ans.

17. **/Maximum Size Subarray Sum Equals k**

Given an integer array nums and an integer k, return the maximum length of a subarray that sums to k. If there is not one, return 0 instead.
Example:
Input: nums = [1,-1,5,-2,3], k = 3
Output: 4
Solution: Iterate through the vector nums mapping the cumulative sum of each element. Through each iteration, increase sum by the current element in nums. If sum - k exists in the sum map, and it is the max so far, save it to variable max. Continue until end of vector.

18. **Merge Two Sorted Linked Lists**
Given the heads of two sorted linked lists list1 and list2. Merge the two lists in a one sorted list. The list should be made by splicing together the nodes of the first two lists. Return the head of the merged linked list.
Example:
Input: list1 = [1,2,4], list2 = [1,3,4]
Output: [1,1,2,3,4,4]
Solution: Iterate through both linked lists, if the current element from list1 is less than the current element from list2, then insert the current element of list1 to the new linked list. Otherwise insert the current element from list2 to the new linked list. Continue until you reach the end of both linked lists.

19. **Remove Duplicates from Sorted List I**
Given the head of a sorted linked list, delete all duplicates such that each element appears only once. Return the linked list sorted as well.
Example:
Input: list = [1,1,2]
Output: [1,2]
Solution: Iterate through the linked list inserting all the elements into a map. A map will only insert a number once so when the map has gone through the entire linked list it will contain all the numbers without duplicates. Create a linked list from the elements in the map.

20. **Remove Duplicates from Sorted List II**
Given the head of a sorted linked list, delete all nodes that have duplicate numbers, leaving only distinct numbers from the original list. Return the linked list sorted as well.
Example:
Input: list = [1,2,3,3,4,4,5]
Output: [1,2,5]
Solution: Iterate through the linked list with a map keeping track of how many times each element shows up. Then iterate through the map and create a linked list with the values that only occurred once.

21. **Remove Duplicates from Unsorted Linked List**
Given the head of a unsorted linked list, find all the values that appear more than once in the list and delete the nodes that have any of those values. Return the linked list after the deletions.
Example:
Input: list1 = [3,2,2,1,3,2,4]
Output: [1,4]

Solution: Iterate through the linked list with a map keeping track of how many times each element shows up. Then iterate through the map and create a linked list with the values that only occurred once. (Same as previous problem).

22. **Palindrome Linked List**

Given the head of a linked list, return true if it is a palindrome.

Example:

Input: head = [1,2,2,1]

Output: true

Solution: Create a function that can reverse a linked list. Then iterate through the linked list with two pointers, one iterates through one element at a time and the other iterates twice as fast, use this to find the halfway point of the linked list. Iterate from the beginning of the linked list to the halfway point, then compare it to the reverse of the halfway point to the end of the linked list. If they are equal then the linked list is a palindrome.