03. Grokking Patterns

Easier Problems

01. Contains Duplicate

Problem Statement

Given an integer array `nums`, return `true` if any value appears **at least twice** in the array, and return `false` if every element is distinct.

Example 1:

```
Input: nums= [1, 2, 3, 4]
Output: false
Explanation: There are no duplicates in the given array.
```

Example 2:

```
Input: nums= [1, 2, 3, 1]
Output: true
Explanation: '1' is repeating.
```

---

02. Pangram

Problem Statement

A pangram is a sentence where every letter of the English alphabet appears at least once.

Given a string `sentence` containing English letters (lower or upper-case), return `true` if `sentence` is a pangram, or `false` otherwise.

Note: The given `sentence` might contain other characters like digits or spaces, your solution should handle these too.

Example 1:

```
Input: sentence = "TheQuickBrownFoxJumpsOverTheLazyDog"
Output: true
Explanation: The sentence contains at least one occurrence of every letter of the English alphabet either in lower or upper case.
```

Example 2:

```
Input: sentence = "This is not a pangram"
Output: false
Explanation: The sentence doesn't contain at least one occurrence of every
letter of the English alphabet.
```

---

03. Reverse Vowels

Problem Statement

Given a string `s`, reverse only all the vowels in the string and return it.

The vowels are `'a'`, `'e'`, `'i'`, `'o'`, and `'u'`, and they can appear in both lower and upper cases, more than once.

Example 1:

```
Input: s= "hello"
Output: "holle"
```

Example 2:

```
Input: s= "AEIOU"
Output: "UOIEA"
```

Example 3:

```
Input: s= "DesignGUrus"
Output: "DusUgnGires"
```

---

04. Valid Palindrome

Problem Statement

A phrase is a **palindrome** if, after converting all uppercase letters into lowercase letters and removing all non-alphanumeric characters, it reads the same forward and backward. Alphanumeric characters include letters and numbers.

Given a string `s`, return `true` if it is a palindrome, or `false` otherwise.

Example 1:

```
Input: sentence = "A man, a plan, a canal, Panama!"
Output: true
Explanation: "amanaplanacanalpanama" is a palindrome.
```

Example 2:

```
Input: sentence = "Was it a car or a cat I saw?"
Output: true
```

---

05. Valid Anagram

Problem Statement

Given two strings `s` and `t`, return `true` if `t` is an anagram of `s`, and `false` otherwise.

An **Anagram** is a word or phrase formed by rearranging the letters of a different word or phrase, typically using all the original letters exactly once.

Example 1:

```
Input: s = "listen", t = "silent"
Output: true
```

Example 2:

```
Input: s = "rat", t = "car"
Output: false
```

Example 3:

```
Input: s = "hello", t = "world"
Output: false
```

---

06. Shortest Word Distance

Problem Statement

Given an array of strings `words` and two different strings that already exist in the array `word1` and `word2`, return the shortest distance between these two words in the list.

Example 1:

```
Input: words = ["the", "quick", "brown", "fox", "jumps", "over", "the",
"lazy", "dog"], word1 = "fox", word2 = "dog"
Output: 5
Explanation: The distance between "fox" and "dog" is 5 words.
```

Example 2:

```
Input: words = ["a", "c", "d", "b", "a"], word1 = "a", word2 = "b"
Output: 1
```

```
Explanation: The shortest distance between "a" and "b" is 1 word
```

---

07. Number of Good Pairs

Problem Statement

Given an array of integers `nums`, return the number of good pairs.

A pair `(i, j)` is called good if `nums[i] == nums[j]` and `i < j`.

Example 1:

```
Input: nums = [1,2,3,1,1,3]
Output: 4
Explanation: There are 4 good pairs, here are the indices: (0,3), (0,4),
(3,4), (2,5).
```

Example 2:

```
Input: nums = [1,1,1,1]
Output: 6
Explanation: Each pair in the array is a 'good pair'.
```

Example 3:

```
Input: words = nums = [1,2,3]
Output: 0
Explanation: No number is repeating.
```

---

08. Square Root

Problem Statement

Given a non-negative integer `x`, return the square root of `x` rounded down to the nearest integer. The returned integer should be non-negative as well.

You **must not use** any built-in exponent function or operator.

For example, do not use `pow(x, 0.5)` in c++ or `x ** 0.5` in python.

Example 1:

```
Input: x = 8
Output: 2
```

```
Explanation: The square root of 8 is 2.8284, and since we need to return the floor of the square root (integer), hence we returned 2.
```

Example 2:

```
Input: x = 4
Output: 2
Explanation: The square root of 4 is 2.
```

Example 3:

```
Input: x = 2
Output: 1
Explanation: The square root of 2 is 1.414, and since we need to return the floor of the square root (integer), hence we returned 1.
```

Pattern: Two Pointers

09. Pair with Target Sum

Problem Statement

Given an array of numbers sorted in ascending order and a target sum, find a **pair in the array whose sum is equal to the given target**.

Write a function to return the indices of the two numbers (i.e. the pair) such that they add up to the given target. If no such pair exists return [-1, -1].

Example 1:

```
Input: [1, 2, 3, 4, 6], target=6
Output: [1, 3]
Explanation: The numbers at index 1 and 3 add up to 6: 2+4=6
```

Example 2:

```
Input: [2, 5, 9, 11], target=11
Output: [0, 2]
Explanation: The numbers at index 0 and 2 add up to 11: 2+9=11
```

---

10. Find Non-Duplicate Number Instances

Problem Statement

Given an array of sorted numbers, move all non-duplicate number instances at the beginning of the array in-place. The relative order of the elements should be kept the same and you should **not use any**

extra space so that the solution has constant space complexity i.e., .

Move all the unique number instances at the beginning of the array and after moving return the length of the subarray that has no duplicate in it.

Example 1:

```
Input: [2, 3, 3, 3, 6, 9, 9]
Output: 4
Explanation: The first four elements after moving element will be [2, 3, 6, 9].
```

Example 2:

```
Input: [2, 2, 2, 11]
Output: 2
Explanation: The first two elements after moving elements will be [2, 11].
```

---

11. Squaring a Sorted Array

Problem Statement

Given a sorted array, create a new array containing squares of all the numbers of the input array in the sorted order.

Example 1:

```
Input: [-2, -1, 0, 2, 3]
Output: [0, 1, 4, 4, 9]
```

Example 2:

```
Input: [-3, -1, 0, 1, 2]
Output: [0, 1, 1, 4, 9]
```

---

12. Triplet Sum to Zero

Problem Statement

Given an array of unsorted numbers, find all unique triplets in it that add up to zero.

Example 1:

```
Input: [-3, 0, 1, 2, -1, 1, -2]
Output: [[-3, 1, 2], [-2, 0, 2], [-2, 1, 1], [-1, 0, 1]]
```

```
Explanation: There are four unique triplets whose sum is equal to zero.
smallest sum.'
```

Example 2:

```
Input: [-5, 2, -1, -2, 3]
Output: [[-5, 2, 3], [-2, -1, 3]]
Explanation: There are two unique triplets whose sum is equal to zero.
```

---

13. Triplet Sum Close to Target

Problem Statement

Given an array of unsorted numbers and a target number, find a **triplet in the array whose sum is as close to the target number as possible**, return the sum of the triplet. If there are more than one such triplet, return the sum of the triplet with the smallest sum.

Example 1:

```
Input: [-1, 0, 2, 3], target=3
Output: 2
Explanation: The triplet [-1, 0, 3] has the sum '2' which is closest to the
target.

There are two triplets with distance '1' from the target: [-1, 0, 3] & [-1,
2, 3]. Between these two triplets, the correct answer will be [-1, 0, 3] as
it has a sum '2' which is less than the sum of the other triplet which is
'4'. This is because of the following requirement: 'If there are more than
one such triplet, return the sum of the triplet with the smallest sum.'
```

Example 2:

```
Input: [-3, -1, 1, 2], target=1
Output: 0
Explanation: The triplet [-3, 1, 2] has the closest sum to the target.
```

Example 3:

```
Input: [1, 0, 1, 1], target=100
Output: 3
Explanation: The triplet [1, 1, 1] has the closest sum to the target.
```

Example 4:

```
Input: [0, 0, 1, 1, 2, 6], target=5
Output: 4
```

```
Explanation: There are two triplets with distance '1' from target: [1, 1, 2] & [0, 0, 6]. Between these two triplets, the correct answer will be [1, 1, 2] as it has a sum '4' which is less than the sum of the other triplet which is '6'. This is because of the following requirement: 'If there are more than one such triplet, return the sum of the triplet with the smallest sum.'
```

---

14. Triplets with Smaller Sum

Problem Statement

Given an array **arr** of unsorted numbers and a target sum, count all triplets in it such that **arr[i] + arr[j] + arr[k] < target** where **i**, **j**, and **k** are three different indices. Write a function to return the count of such triplets.

Example 1:

```
Input: [-1, 0, 2, 3], target=3
Output: 2
Explanation: There are two triplets whose sum is less than the target: [-1, 0, 3], [-1, 0, 2]
```

Example 2:

```
Input: [-1, 4, 2, 1, 3], target=5
Output: 4
Explanation: There are four triplets whose sum is less than the target: [-1, 1, 4], [-1, 1, 3], [-1, 1, 2], [-1, 2, 3]
```

---

15. Subarrays with Product Less than a Target

Problem Statement

Given an array with positive numbers and a positive target number, find all of its contiguous subarrays whose **product is less than the target number**.

Example 1:

```
Input: [2, 5, 3, 10], target=30
Output: [2], [5], [2, 5], [3], [5, 3], [10]
Explanation: There are six contiguous subarrays whose product is less than the target.
```

Example 2:

```
Input: [8, 2, 6, 5], target=50
Output: [8], [2], [8, 2], [6], [2, 6], [5], [6, 5]
Explanation: There are seven contiguous subarrays whose product is less than
the target.
```

---

16. Dutch National Flag Problem

Problem Statement

Given an array containing 0s, 1s and 2s, sort the array in-place. You should treat numbers of the array as objects, hence, we can't count 0s, 1s, and 2s to recreate the array.

The flag of the Netherlands consists of three colors: red, white and blue; and since our input array also consists of three different numbers that is why it is called **Dutch National Flag problem**.

Example 1:

```
Input: [1, 0, 2, 1, 0]
Output: [0 0 1 1 2]
```

Example 2:

```
Input: [2, 2, 0, 1, 2, 0]
Output: [0 0 1 2 2 2 ]
```

---

17. Quadruple Sum to Target

Problem Statement

Given an array of unsorted numbers and a target number, find all **unique quadruplets** in it, whose **sum is equal to the target number**.

Example 1:

```
Input: [4, 1, 2, -1, 1, -3], target=1
Output: [-3, -1, 1, 4], [-3, 1, 1, 2]
Explanation: Both the quadruplets add up to the target.
```

Example 2:

```
Input: [2, 0, -1, 1, -2, 2], target=2
Output: [-2, 0, 2, 2], [-1, 0, 1, 2]
Explanation: Both the quadruplets add up to the target.
```

---

18. Comparing Strings containing Backspaces

Problem Statement

Given two strings containing backspaces (identified by the character '#'), check if the two strings are equal.

Example 1:

```
Input: str1="xy#z", str2="xzz#"
Output: true
Explanation: After applying backspaces the strings become "xz" and "xz"
respectively.
```

Example 2:

```
Input: str1="xy#z", str2="xyz#"
Output: false
Explanation: After applying backspaces the strings become "xz" and "xy"
respectively.
```

Example 3:

```
Input: str1="xp#", str2="xyz##"
Output: true
Explanation: After applying backspaces the strings become "x" and "x"
respectively.
```

---

19. Minimum Window Sort

Problem Statement

Given an array, find the length of the smallest subarray in it which when sorted will sort the whole array.

Example 1:

```
Input: [1, 2, 5, 3, 7, 10, 9, 12]
Output: 5
Explanation: We need to sort only the subarray [5, 3, 7, 10, 9] to make the
whole array sorted
```

Example 2:

```
Input: [1, 3, 2, 0, -1, 7, 10]
Output: 5
Explanation: We need to sort only the subarray [1, 3, 2, 0, -1] to make the
whole array sorted
```

Example 3:

```
Input: [1, 2, 3]
Output: 0
Explanation: The array is already sorted
```

Example 4:

```
Input: [3, 2, 1]
Output: 3
```

Pattern: Fast & Slow Pointers

20. LinkedList Cycle

Problem Statement

Given the head of a **Singly LinkedList**, write a function to determine if the LinkedList has a **cycle** in it or not.

---

21. Middle of the LinkedList

Problem Statement

Given the head of a **Singly LinkedList**, write a method to return the **middle node** of the LinkedList.

If the total number of nodes in the LinkedList is even, return the second middle node.

Example 1:

```
Input: 1 -> 2 -> 3 -> 4 -> 5 -> null
Output: 3
```

Example 2:

```
Input: 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> null
Output: 4
```

Example 3:

```
Input: 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> null
Output: 4
```

---

22. Start of LinkedList Cycle

Problem Statement

Given the head of a **Singly LinkedList** that contains a cycle, write a function to find the **starting node of the cycle**.

---

23. Happy Number

Problem Statement

Any number will be called a happy number if, after repeatedly replacing it with a number equal to the **sum of the square of all of its digits, leads us to the number `1`. All other (not-happy) numbers will never reach `1`**. Instead, they will be stuck in a cycle of numbers that does not include `1`.

Given a positive number `n`, return `true` if it is a happy number otherwise return `false`.

Example 1:

> **Input**: 23
> **Output**: true (23 is a happy number)
> **Explanations**: Here are the steps to find out that 23 is a happy number:
>
> = 4 + 9 = 13

---

24. Palindrome LinkedList

Problem Statement

Given the head of a **Singly LinkedList**, write a method to check if the **LinkedList is a palindrome or not**.

Your algorithm should use **constant space** and the input LinkedList should be in the original form once the algorithm is finished. The algorithm should have O(N) time complexity where 'N' is the number of nodes in the LinkedList.

Example 1:

```
Input: 2 -> 4 -> 6 -> 4 -> 2 -> null
Output: true
```

Example 2:

```
Input: 2 -> 4 -> 6 -> 4 -> 2 -> 2 -> null
Output: false
```

---

25. Rearrange a LinkedList

Problem Statement

Given the head of a Singly LinkedList, write a method to modify the LinkedList such that the **nodes from the second half of the LinkedList are inserted alternately to the nodes from the first half in reverse order**. So if the LinkedList has nodes `1 -> 2 -> 3 -> 4 -> 5 -> 6 -> null`, your method should return `1 -> 6 -> 2 -> 5 -> 3 -> 4 -> null`.

Your algorithm should use only constant space the input LinkedList should be modified in-place.

Example 1:

```
Input: 2 -> 4 -> 6 -> 8 -> 10 -> 12 -> null
Output: 2 -> 12 -> 4 -> 10 -> 6 -> 8 -> null
```

---

26. Cycle in a Circular Array

Problem Statement

We are given an array containing positive and negative numbers. Suppose the array contains a number ‘M’ at a particular index. Now, if ‘M’ is positive we will move forward ‘M’ indices and if ‘M’ is negative move backwards ‘M’ indices. You should assume that the **array is circular** which means two things:

1. If, while moving forward, we reach the end of the array, we will jump to the first element to continue the movement.
2. If, while moving backward, we reach the beginning of the array, we will jump to the last element to continue the movement.

Pattern: Sliding Window

27. Maximum Sum Subarray of Size K

Problem Statement

Given an array of positive numbers and a positive number 'k,' find the **maximum sum of any contiguous subarray of size 'k'**.

Example 1:

```
Input: [2, 1, 5, 1, 3, 2], k=3
Output: 9
Explanation: Subarray with maximum sum is [5, 1, 3].
```

Example 2:

```
Input: [2, 3, 4, 1, 5], k=2
Output: 7
Explanation: Subarray with maximum sum is [3, 4].
```

---

28. Smallest Subarray With a Greater Sum

Problem Statement

Given an array of positive integers and a number ‘S,’ find the length of the **smallest** contiguous subarray whose sum is **greater than or equal to 'S'**. Return 0 if no such subarray exists.

Example 1:

```
Input: [2, 1, 5, 2, 3, 2], S=7
Output: 2
Explanation: The smallest subarray with a sum greater than or equal to '7' is [5, 2].
```

Example 2:

```
Input: [2, 1, 5, 2, 8], S=7
Output: 1
Explanation: The smallest subarray with a sum greater than or equal to '7' is [8].
```

Example 3:

```
Input: [3, 4, 1, 1, 6], S=8
Output: 3
```

---

29. Longest Substring with K Distinct Characters

Problem Statement

Given a string, find the length of the **longest substring** in it **with no more than K distinct characters**.

You can assume that K is less than or equal to the length of the given string.

Example 1:

```
Input: String="araaci", K=2
Output: 4
```

```
Explanation: The longest substring with no more than '2' distinct characters
is "araa".
```

Example 2:

```
Input: String="araaci", K=1
Output: 2
Explanation: The longest substring with no more than '1' distinct characters
is "aa".
```

Example 3:

```
Input: String="cbbebi", K=3
Output: 5
```

---

30. Fruits into Baskets

Sure, here's the problem statement and the examples in markdown format:

Problem Statement:

You are given an array of integers representing fruits, where each integer corresponds to a specific type of fruit. You are also given an integer 'k', which represents the maximum number of different types of fruits you can put into two baskets. Your goal is to find the length of the longest contiguous subarray that contains at most 'k' different types of fruits.

Examples:

Example 1:

```
Input: [1, 2, 1, 3, 4, 3, 5, 1, 2], k = 2
Output: 5
Explanation: The longest subarray with at most 2 different types of fruits
is [3, 4, 3, 5, 1].
```

Example 2:

```
Input: [3, 3, 3, 1, 2, 1, 1, 2, 3, 3, 4], k = 3
Output: 5
Explanation: The longest subarray with at most 3 different types of fruits
is [1, 2, 1, 1, 2].
```

---

31. Longest Substring with Same Letters after Repl

Problem Statement

Given a string with lowercase letters only, if you are allowed to **replace no more than ‘k’ letters** with any letter, find the **length of the longest substring having the same letters** after replacement.

Example 1:

```
Input: String="aabccbb", k=2
Output: 5
Explanation: Replace the two 'c' with 'b' to have a longest repeating substring "bbbbb".
```

Example 2:

```
Input: String="abbcb", k=1
Output: 4
Explanation: Replace the 'c' with 'b' to have a longest repeating substring "bbbb".
```

---

32. Longest Subarray with Ones after Replacement

Problem Statement

Given an array containing 0s and 1s, if you are allowed to **replace no more than ‘k’ 0s with 1s**, find the length of the **longest contiguous subarray having all 1s**.

Example 1:

```
Input: Array=[0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1], k=2
Output: 6
Explanation: Replace the '0' at index 5 and 8 to have the longest contiguous subarray of 1s having length 6.
```

Example 2:

```
Input: Array=[0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1], k=3
Output: 9
```

---

33. Permutation in a String

Problem Statement

Given a string and a pattern, find out if the **string contains any permutation of the pattern**.

Permutation is defined as the re-arranging of the characters of the string. For example, “abc” has the following six permutations:

1. abc
2. acb
3. bac
4. bca
5. cab
6. cba

If a string has ‘n’ distinct characters, it will have *n*! permutations.

Example 1:

```
Input: String="oidbcaf", Pattern="ABC"
Output: true
Explanation: The string contains "bca" which is a permutation of the given pattern.
```

---

34. String Anagrams

Problem Statement

Given a string and a pattern, find **all anagrams of the pattern in the given string**.

Every **anagram** is a **permutation** of a string. As we know, when we are not allowed to repeat characters while finding permutations of a string, we get *N*! permutations (or anagrams) of a string having N*N* characters. For example, here are the six anagrams of the string “abc”:

1. abc
2. acb
3. bac
4. bca
5. cab
6. cba

Write a function to return a list of starting indices of the anagrams of the pattern in the given string.

---

35. Smallest Window containing Substring

Problem Statement

Given a string and a pattern, find the **smallest substring** in the given string which has **all the character occurrences of the given pattern**.

Example 1:

```
Input: String="aabdec", Pattern="abc"
Output: "abdec"
Explanation: The smallest substring having all characters of the pattern is
"abdec"
```

Example 2:

```
Input: String="aabdec", Pattern="abac"
Output: "aabdec"
Explanation: The smallest substring having all characters occurrences of the
pattern is "aabdec"
```

Example 3:

```
Input: String="abdbca", Pattern="abc"
```

---

36. Words Concatenation

Problem Statement

Given a string and a list of words, find all the starting indices of substrings in the given string that are a **concatenation of all the given words** exactly once **without any overlapping** of words. It is given that all words are of the same length.

Example 1:

```
Input: String="catfoxcat", Words=["cat", "fox"]
Output: [0, 3]
Explanation: The two substring containing both the words are "catfox" &
"foxcat".
```

Example 2:

```
Input: String="catcatfoxfox", Words=["cat", "fox"]
Output: [3]
```

Pattern: Merge Intervals

37. Merge Intervals

Problem Statement

Given a list of intervals, **merge all the overlapping intervals** to produce a list that has only mutually exclusive intervals.

Example 1:

```
Intervals: [[1,4], [2,5], [7,9]]
Output: [[1,5], [7,9]]
Explanation: Since the first two intervals [1,4] and [2,5] overlap, we merged them into one [1,5].
```

Example 2:

```
Intervals: [[6,7], [2,4], [5,9]]
Output: [[2,4], [5,9]]
Explanation: Since the intervals [6,7] and [5,9] overlap, we merged them into one [5,9].
```

Example 3:

```
Intervals: [[1,4], [2,6], [3,5]]
Output: [[1,6]]
```

---

38. Insert Interval

Problem Statement

Given a list of non-overlapping intervals sorted by their start time, **insert a given interval at the correct position** and merge all necessary intervals to produce a list that has only mutually exclusive intervals.

Example 1:

```
Input: Intervals=[[1,3], [5,7], [8,12]], New Interval=[4,6]
Output: [[1,3], [4,7], [8,12]]
Explanation: After insertion, since [4,6] overlaps with [5,7], we merged them into one [4,7].
```

Example 2:

```
Input: Intervals=[[1,3], [5,7], [8,12]], New Interval=[4,10]
Output: [[1,3], [4,12]]
```

---

39. Intervals Intersection

Problem Statement

Given two lists of intervals, find the intersection of these two lists. Each list consists of disjoint intervals sorted on their start time.

Example 1:

```
Input: arr1=[[1, 3], [5, 6], [7, 9]], arr2=[[2, 3], [5, 7]]
Output: [2, 3], [5, 6], [7, 7]
Explanation: The output list contains the common intervals between the two lists.
```

Example 2:

```
Input: arr1=[[1, 3], [5, 7], [9, 12]], arr2=[[5, 10]]
Output: [5, 7], [9, 10]
Explanation: The output list contains the common intervals between the two lists.
```

---

40. Conflicting Appointments

Problem Statement

Given an array of intervals representing ‘N’ appointments, find out if a person can **attend all the appointments**.

Example 1:

```
Appointments: [[1,4], [2,5], [7,9]]
Output: false
Explanation: Since [1,4] and [2,5] overlap, a person cannot attend both of these appointments.
```

Example 2:

```
Appointments: [[6,7], [2,4], [8,12]]
Output: true
Explanation: None of the appointments overlap, therefore a person can attend all of them.
```

Example 3:

```
Appointments: [[4,5], [2,3], [3,6]]
Output: false
```

---

41. Minimum Meeting Rooms

Problem Statement

Given a list of intervals representing the start and end time of ‘N’ meetings, find the **minimum number of rooms** required to **hold all the meetings**.

Example 1:

```
Meetings: [[1,4], [2,5], [7,9]]
Output: 2
Explanation: Since [1,4] and [2,5] overlap, we need two rooms to hold these two meetings. [7,9] can occur in any of the two rooms later.
```

Example 2:

```
Meetings: [[6,7], [2,4], [8,12]]
Output: 1
Explanation: None of the meetings overlap, therefore we only need one room to hold all meetings.
```

Example 3:

```
Meetings: [[1,4], [2,3], [3,6]]
```

---

42. Maximum CPU Load

Problem Statement

We are given a list of Jobs. Each job has a Start time, an End time, and a CPU load when it is running. Our goal is to find the **maximum CPU load** at any time if all the **jobs are running on the same machine**.

Example 1:

```
Jobs: [[1,4,3], [2,5,4], [7,9,6]]
Output: 7
Explanation: Since [1,4,3] and [2,5,4] overlap, their maximum CPU load (3+4=7) will be when both the jobs are running at the same time i.e., during the time interval (2,4).
```

Example 2:

```
Jobs: [[6,7,10], [2,4,11], [8,12,15]]
Output: 15
```

---

43. Employee Free Time

Problem Statement

For ‘K’ employees, we are given a list of intervals representing each employee’s working hours. Our goal is to determine if there is a **free interval which is common to all employees**. You can assume that each list of employee working hours is sorted on the start time.

Example 1:

```
Input: Employee Working Hours=[[[1,3], [5,6]], [[2,3], [6,8]]]
Output: [3,5]
Explanation: All the employees are free between [3,5].
```

Example 2:

```
Input: Employee Working Hours=[[[1,3], [9,12]], [[2,4]], [[6,8]]]
Output: [4,6], [8,9]
```

Pattern: Cyclic Sort

44. Cyclic Sort

Problem Statement

We are given an array containing `n` objects. Each object, when created, was assigned a unique number from the range `1` to `n` based on their creation sequence. This means that the object with sequence number `3` was created just before the object with sequence number `4`.

Write a function to sort the objects in-place on their creation sequence number in O(n) and without using any extra space. For simplicity, let’s assume we are passed an integer array containing only the sequence numbers, though each number is actually an object.

---

45. Find the Missing Number

Problem Statement

We are given an array containing `n` distinct numbers taken from the range `0` to `n`. Since the array has only n numbers out of the total `n+1` numbers, find the missing number.

Example 1:

```
Input: [4, 0, 3, 1]
Output: 2
```

Example 2:

```
Input: [8, 3, 5, 2, 4, 6, 0, 1]
Output: 7
```

---

46. Find all Missing Numbers

Problem Statement

We are given an unsorted array containing numbers taken from the range 1 to ‘n’. The array can have duplicates, which means some numbers will be missing. Find all those missing numbers.

Example 1:

```
Input: [2, 3, 1, 8, 2, 3, 5, 1]
Output: 4, 6, 7
Explanation: The array should have all numbers from 1 to 8, due to
duplicates 4, 6, and 7 are missing.
```

Example 2:

```
Input: [2, 4, 1, 2]
Output: 3
```

Example 3:

```
Input: [2, 3, 2, 1]
Output: 4
```

---

47. Find the Duplicate Number

Problem Statement

We are given an unsorted array containing `n+1` numbers taken from the range `1` to `n`. The array has only one duplicate but it can be repeated multiple times. **Find that duplicate number without using any extra space**. You are, however, allowed to modify the input array.

Example 1:

```
Input: [1, 4, 4, 3, 2]
Output: 4
```

Example 2:

```
Input: [2, 1, 3, 3, 5, 4]
Output: 3
```

Example 3:

```
Input: [2, 4, 1, 4, 4]
Output: 4
```

---

48. Find all Duplicate Numbers

Problem Statement

We are given an unsorted array containing `n` numbers taken from the range `1 to n`. The array has some numbers appearing twice, **find all these duplicate numbers using constant space**.

Example 1:

```
Input: [3, 4, 4, 5, 5]
Output: [4, 5]
```

Example 2:

```
Input: [5, 4, 7, 2, 3, 5, 3]
Output: [3, 5]
```

---

49. Find the Corrupt Pair

Problem Statement

We are given an unsorted array containing ‘n’ numbers taken from the range 1 to ‘n’. The array originally contained all the numbers from 1 to ‘n’, but due to a data error, one of the numbers got duplicated which also resulted in one number going missing. Find both these numbers.

Example 1:

```
Input: [3, 1, 2, 5, 2]
Output: [2, 4]
Explanation: '2' is duplicated and '4' is missing.
```

Example 2:

```
Input: [3, 1, 2, 3, 6, 4]
Output: [3, 5]
Explanation: '3' is duplicated and '5' is missing.
```

---

50. Find the Smallest Missing Positive Number

Problem Statement

Given an unsorted array containing numbers, find the **smallest missing positive number** in it.

Note: Positive numbers start from '1'.

Example 1:

```
Input: [-3, 1, 5, 4, 2]
Output: 3
Explanation: The smallest missing positive number is '3'
```

Example 2:

```
Input: [3, -2, 0, 1, 2]
Output: 4
```

Example 3:

```
Input: [3, 2, 5, 1]
Output: 4
```

Example 4:

```
Input: [33, 37, 5]
Output: 1
```

---

51. Find the First K Missing Positive Numbers

Problem Statement

Given an unsorted array containing numbers and a number ‘k’, find the first ‘k’ missing positive numbers in the array.

Example 1:

```
Input: [3, -1, 4, 5, 5], k=3
Output: [1, 2, 6]
Explanation: The smallest missing positive numbers are 1, 2 and 6.
```

Example 2:

```
Input: [2, 3, 4], k=3
Output: [1, 5, 6]
Explanation: The smallest missing positive numbers are 1, 5 and 6.
```

Example 3:

```
Input: [-2, -3, 4], k=2
Output: [1, 2]
Explanation: The smallest missing positive numbers are 1 and 2.
```

Pattern: In-place Reversal of a Linked List

52. Reverse a LinkedList

Problem Statement

Given the head of a Singly LinkedList, reverse the LinkedList. Write a function to return the new head of the reversed LinkedList.

---

53. Reverse a Sub-list

Problem Statement

Given the head of a LinkedList and two positions ‘p’ and ‘q’, reverse the LinkedList from position ‘p’ to ‘q’.

---

54. Reverse every K-element Sub-list

Problem Statement

Given the head of a LinkedList and a number ‘k’, **reverse every ‘k’ sized sub-list starting from the head**.

If, in the end, you are left with a sub-list with less than ‘k’ elements, reverse it too.

---

55. Reverse alternating K-element Sub-list

Problem Statement

Given the head of a LinkedList and a number ‘k’, **reverse every alternating ‘k’ sized sub-list** starting from the head.

If, in the end, you are left with a sub-list with less than ‘k’ elements, reverse it too.

---

56. Rotate a LinkedList

Problem Statement

Given the head of a Singly LinkedList and a number ‘k’, rotate the LinkedList to the right by ‘k’ nodes.

Pattern: Stacks

57. Balanced Parentheses

Problem Statement

Given a string `s` containing `(`, `)`, `[`, `]`, `{`, and `}` characters. Determine if a given string of parentheses is balanced.

A string of parentheses is considered balanced if every opening parenthesis has a corresponding closing parenthesis in the correct order.

Example 1:

Input: String s = "{[()]}";
Expected Output: true
Explanation: The parentheses in this string are perfectly balanced. Every opening parenthesis '{', '[', '(' has a corresponding closing parenthesis '}', ']', ')' in the correct order.

---

58. Reverse a String

Problem Statement

Given a string, write a function that uses a stack to reverse the string. The function should return the reversed string.

Examples

Example 1:

```
Input: "Hello, World!"
Output: "!dlroW ,olleH"
```

Example 2:

```
Input: "OpenAI"
Output: "IAnepO"
```

Example 3:

```
Input: "Stacks are fun!"
Output: "!nuf era skcatS"
```

---

59. Decimal to Binary Conversion

Problem Statement

Given a positive integer `n`, write a function that returns its binary equivalent as a string. The function should not use any in-built binary conversion function.

Examples

Example 1:

```
Input: 2
Output: "10"
Explanation: The binary equivalent of 2 is 10.
```

Example 2:

```
Input: 7
Output: "111"
Explanation: The binary equivalent of 7 is 111.
```

Example 3:

```
Input: 18
Output: "10010"
Explanation: The binary equivalent of 18 is 10010.
```

---

60. Next Greater Element

Problem Statement

Given an array, print the Next Greater Element (NGE) for every element.

The Next Greater Element for an element x is the first greater element on the right side of x in the array.

Elements for which no greater element exist, consider the next greater element as -1.

Examples

Example 1:

```
Input: [4, 5, 2, 25]
Output: [5, 25, 25, -1]
```

Example 1:

```
Input: [13, 7, 6, 12]
Output: [-1, 12, 12, -1]
```

Example 1:

```
Input: [1, 2, 3, 4, 5]
Output: [2, 3, 4, 5, -1]
```

---

61. Sorting a Stack

Problem Statement

Given a stack, sort it using only stack operations (push and pop).

You can use an additional temporary stack, but you may not copy the elements into any other data structure (such as an array). The values in the stack are to be sorted in descending order, with the largest elements on top.

Examples

```
1. Input: [34, 3, 31, 98, 92, 23]
   Output: [3, 23, 31, 34, 92, 98]

2. Input: [4, 3, 2, 10, 12, 1, 5, 6]
   Output: [1, 2, 3, 4, 5, 6, 10, 12]

3. Input: [20, 10, -5, -1]
   Output: [-5, -1, 10, 20]
```

---

62. Simplify Path

Problem Statement

Given an absolute file path in a Unix-style file system, simplify it by converting ".." to the previous directory and removing any "." or multiple slashes. The resulting string should represent the shortest absolute path.

Examples:

```
1.
   Input: "/a//b////c/d//././/.."
   Output: "/a/b/c"

2.
   Input: "/../"
   Output: "/"

3.
```

```
Input: "/home//foo/"
Output: "/home/foo"
```

Pattern: Monotonic Stack

63. Next Greater Element

Problem Statement

Given two integer arrays `nums1` and `nums2`, return an array `answer` such that `answer[i]` is the next greater number for every `nums1[i]` in `nums2`. The next greater element for an element `x` is the first element to the right of `x` that is greater than `x`. If there is no greater number, output `-1` for that number.

The numbers in `nums1` are all present in `nums2` and `nums2` is a permutation of `nums1`.

Examples

1.
   - Input: `nums1 = [4,2,6]`, `nums2 = [6,2,4,5,3,7]`
   - Output: `[5,4,7]`

---

64. Daily Temperatures

Problem Statement

Given an array of integers `temperatures` representing daily temperatures, your task is to calculate how many days you have to wait until a warmer temperature. If there is no future day for which this is possible, put 0 instead.

Examples

1.
   - Input: `temperatures = [70, 73, 75, 71, 69, 72, 76, 73]`
   - Output: `[1, 1, 4, 2, 1, 1, 0, 0]`
   - Explanation: The first day's temperature is 70 and the next day's temperature is 73 which is warmer. So for the first day, you only have to wait for 1 day to get a warmer temperature

---

65. Remove Nodes From Linked List

Problem Statement

Given the head node of a singly linked list, modify the list such that any node that has a node with a greater value to its right gets removed. The function should return the head of the modified list.

Examples:

1. Input: 5 -> 3 -> 7 -> 4 -> 2 -> 1
   Output: 7 -> 4 -> 2 -> 1
   Explanation: 5 and 3 are removed as they have nodes with larger values to their right.
2. Input: 1 -> 2 -> 3 -> 4 -> 5
   Output: 5
   Explanation: 1, 2, 3, and 4 are removed as they have nodes with larger values to their right.

---

66. Remove All Adjacent Duplicates In String

Problem Statement

Given a string `S`, remove all adjacent duplicate characters recursively to generate the resultant string.

Examples

1.
   - Input: `s = "abccba"`
   - Output: `""`
   - Explanation: First, we remove "cc" to get "abba". Then, we remove "bb" to get "aa". Finally, we remove "aa" to get an empty string.
2.
   - Input: `s = "foobar"`
   - Output: `"fbar"`
   - Explanation: We remove "oo" to get "fbar".
3.
   - Input: `s = "abcd"`
   - Output: `"abcd"`
   - Explanation: No adjacent duplicates so no changes.

---

67. Remove All Adjacent Duplicates in String II

Problem Statement

You are given a string `s` and an integer `k`. Your task is to remove groups of identical, consecutive characters from the string such that each group has exactly `k` characters. The removal of groups

should continue until it's no longer possible to make any more removals. The result should be the final version of the string after all possible removals have been made.

Examples

1.
   - Input: `s = "abbbaaca", k = 3`
   - Output: `"ca"`
   - Explanation: First, we remove "bbb" to get "aaaca". Then, we remove "aaa" to get "ca".

---

68. Remove K Digits

Problem Statement

Given a non-negative integer represented as a string `num` and an integer `k`, delete `k` digits from `num` to obtain the smallest possible integer. Return this minimum possible integer as a string.

Examples

1.
   - Input: `num = "1432219", k = 3`
   - Output: `"1219"`
   - Explanation: The digits removed are 4, 3, and 2 forming the new number 1219 which is the smallest.
2.
   - Input: `num = "10200", k = 1`
   - Output: `"200"`
   - Explanation: Removing the leading 1 forms the smallest number 200.

Pattern: Hash Maps

69. First Non-repeating Character

Problem Statement

Given a string, identify the position of the first character that appears only once in the string. If no such character exists, return -1.

Examples

1. **Example 1:**

- Input: "apple"
- Expected Output: 0
- Justification: The first character 'a' appears only once in the string and is the first character.

2. **Example 2:**
   - Input: "abcab"
   - Expected Output: 2
   - Justification: The first character that appears only once is 'c' and its position is 2.

---

70. Largest Unique Number

Problem Statement

Given an array of integers, identify the highest value that appears only once in the array. If no such number exists, return -1.

Examples:

1. **Example 1:**
   - Input: [5, 7, 3, 7, 5, 8]
   - Expected Output: 8
   - Justification: The number 8 is the highest value that appears only once in the array.
2. **Example 2:**
   - Input: [1, 2, 3, 2, 1, 4, 4]
   - Expected Output: 3
   - Justification: The number 3 is the highest value that appears only once in the array.
3. **Example 3:**
   - Input: [9, 9, 8, 8, 7, 7]
   - Expected Output: -1

---

71. Maximum Number of Balloons

Problem Statement

Given a string, determine the maximum number of times the word "balloon" can be formed using the characters from the string. Each character in the string can be used only once.

Examples:

1. **Example 1:**

- Input: "balloonballoon"
- Expected Output: 2
- Justification: The word "balloon" can be formed twice from the given string.

2. **Example 2:**
   - Input: "bbaall"
   - Expected Output: 0
   - Justification: The word "balloon" cannot be formed from the given string as we are missing the character 'o' twice.

---

72. Longest Palindrome

Problem Statement:

Given a string, determine the length of the longest palindrome that can be constructed using the characters from the string. You don't need to return the palindrome itself, just its maximum possible length.

Examples:

1.
   - **Input**: "applepie"
   - **Expected Output**: 5
   - **Justification**: The longest palindrome that can be constructed from the string is "pepep", which has a length of 5. There are are other palindromes too but they all will be of length 5.

---

73. Ransom Note

Problem Statement

Given two strings, one representing a ransom note and the other representing the available letters from a magazine, determine if it's possible to construct the ransom note using only the letters from the magazine. Each letter from the magazine can be used only once.

Examples:

1. **Example 1:**
   - Input: Ransom Note = "hello", Magazine = "hellworld"
   - Expected Output: true
   - Justification: The word "hello" can be constructed from the letters in "hellworld".

Pattern: Tree Breadth First Search

74. Binary Tree Level Order Traversal

Problem Statement

Given a binary tree, populate an array to represent its level-by-level traversal. You should populate the values of all **nodes of each level from left to right** in separate sub-arrays.

Examples:

Example 1:

```
Input:
    3
   / \
  9  20
    /  \
   15   7

Output:
[
  [3],
  [9, 20],
  [15, 7]
]
```

Example 2:

```
Input:
       1
      / \
     2   3
    / \
   4   5

Output:
[
  [1],
  [2, 3],
  [4, 5]
]
```

---

75. Reverse Level Order Traversal

Problem Statement

Given a binary tree, populate an array to represent its level-by-level traversal in reverse order, i.e., the **lowest level comes first**. You should populate the values of all nodes in each level from left to right in separate sub-arrays.

Examples:

Example 1:

```
Input:
    3
   / \
  9  20
    /  \
   15   7

Output:
[
  [15, 7],
  [9, 20],
  [3]
]
```

Example 2:

```
Input:
       1
      / \
     2   3
    / \
   4   5

Output:
[
  [4, 5],
  [2, 3],
  [1]
]
```

---

76. Zigzag Traversal

Problem Statement:

Given a binary tree, populate an array to represent its zigzag level order traversal. You should populate the values of all nodes of the first level from left to right, then right to left for the next level, and keep alternating in the same manner for the following levels.

Examples:

Example 1:

```
Input:
    3
   / \
  9  20
    /  \
   15   7

Output:
[
  [3],
  [20, 9],
  [15, 7]
]
```

Example 2:

```
Input:
       1
      / \
     2   3
    / \
   4   5

Output:
[
  [1],
  [3, 2],
  [4, 5]
]
```

---

77. Level Averages in a Binary Tree

Problem Statement:

Given a binary tree, populate an array to represent the averages of all of its levels. Each element of the resulting array should represent the average value of all nodes at that level.

Examples:

Example 1:

```
Input:
    3
   / \
  9  20
    /  \
   15   7

Output:
[3, 14.5, 11]
```

Example 2:

```
Input:
       1
      / \
     2   3
    / \
   4   5

Output:
[1, 2.5, 4]
```

---

78. Minimum Depth of a Binary Tree

Problem Statement

Find the minimum depth of a binary tree. The minimum depth is the number of nodes along the **shortest path from the root node to the nearest leaf node**.

Examples:

Example 1:

```
Input:
    3
   / \
  9  20
    /  \
   15   7
```

```
Output:
2
```

Example 2:

```
Input:
        1
       / \
      2   3
     / \
    4   5

Output:
2
```

---

79. Level Order Successor

Problem Statement

Given a binary tree and a node, find the level order successor of the given node in the tree. The level order successor is the node that appears right after the given node in the level order traversal.

Examples:

Example 1:

```
Input:
     3
    / \
   9  20
     /  \
    15   7

Node: 9

Output:
20
```

Example 2:

```
Input:
        1
       / \
      2   3
     / \
```

```
   4    5

Node: 2

Output:
3
```

---

80. Connect Level Order Siblings

Problem Statement

Given a binary tree, connect each node with its level order successor. The last node of each level should point to a `null` node.

Examples:

Example 1:

```
Input:
     3
    / \
   9  20
     /  \
    15   7

Output:
     3 -> 20 -> null
    /     /  \
   9 -> 15 -> 7 -> null
```

Example 2:

```
Input:
        1
       / \
      2   3
     / \
    4   5

Output:
        1 -> 3 -> null
       /     / \
      2 -> 5 -> null
     / \
    4 -> null
```

81. Connect All Level Order Siblings

Problem Statement

Given a binary tree, connect each node with its level order successor. The last node of each level should point to the first node of the next level.

Examples:

Example 1:

```
Input:
    3
   / \
  9  20
    /  \
   15   7

Output:
    3 -> 9 -> 20 -> 15 -> 7 -> null
```

Example 2:

```
Input:
       1
      / \
     2   3
    / \
   4   5

Output:
       1 -> 2 -> 3 -> 4 -> 5 -> null
```

82. Right View of a Binary Tree

Problem Statement

Given a binary tree, return an array containing nodes in its right view. The right view of a binary tree is the set of **nodes visible when the tree is seen from the right side**.

Pattern: Tree Depth First Search

83. Binary Tree Path Sum

Problem Statement

Given a binary tree and a number ‘S’, find if the tree has a path from root-to-leaf such that the sum of all the node values of that path equals ‘S’.

---

84. All Paths for a Sum

Problem Statement

Given a binary tree and a number ‘S’, find all paths from root-to-leaf such that the sum of all the node values of each path equals ‘S’.

---

85. Sum of Path Numbers

Problem Statement

Given a binary tree where each node can only have a digit (0-9) value, each root-to-leaf path will represent a number. Find the total sum of all the numbers represented by all paths.

---

86. Path With Given Sequence

Problem Statement

Given a binary tree and a number sequence, find if the sequence is present as a root-to-leaf path in the given tree.

---

87. Count Paths for a Sum

Problem Statement

Given a binary tree and a number ‘S’, find all paths in the tree such that the sum of all the node values of each path equals ‘S’. Please note that the paths can start or end at any node but all paths must follow direction from parent to child (top to bottom).

---

88. Tree Diameter

Problem Statement

Given a binary tree, find the length of its diameter. The diameter of a tree is the number of nodes on the **longest path between any two leaf nodes**. The diameter of a tree may or may not pass through the root.

Note: You can always assume that there are at least two leaf nodes in the given tree.

Examples:

Example 1:

```
Input:
    1
   / \
  2   3
 / \
4   5

Output:
3
```

Example 2:

```
Input:
     1
    / \
   2   3
  / \
 4   5
/     \
6       7

Output:
5
```

---

89. Path with Maximum Sum

Problem Statement

Find the path with the maximum sum in a given binary tree. Write a function that returns the maximum sum.

A path can be defined as a **sequence of nodes between any two nodes** and doesn't necessarily pass through the root. The path must contain at least one node.

Pattern: Graphs

90. Find if Path Exists in Graph

Problem Statement

Given an undirected graph, represented as a list of edges. Each edge is illustrated as a pair of integers `[u, v]`, signifying that there's a mutual connection between node `u` and node `v`.

Given this graph, a starting node `start`, and a destination node `end`, your task is to ascertain if a path exists between the starting node and the destination node.

Examples

1. **Example 1:**
    - Input: `4, [[0,1],[1,2],[2,3]], 0, 3`
    - Expected Output: `true`
    - Justification: There's a path from node 0 -> 1 -> 2 -> 3.

---

91. Number of Provinces

Problem Statement

Imagine a country with several cities. Some cities have direct roads connecting them, while others might be connected through a sequence of intermediary cities. Using a matrix representation, if `matrix[i][j]` holds the value `1`, it indicates that city `i` is directly linked to city `j` and vice versa. If it holds `0`, then there's no direct link between them.

Determine the number of separate city clusters (or provinces).

---

92. Minimum Number of Vertices to Reach All Nodes

Problem Statement

Given a directed acyclic graph with `n` nodes labeled from `0` to `n-1`, determine the smallest number of initial nodes such that you can access all the nodes by traversing edges. Return these nodes.

Examples

1. **Input:**
    - `n = 6`
    - `edges = [[0,1],[0,2],[2,5],[3,4],[4,2]]`

    Expected Output: `[0,3]`

    Justification: Starting from nodes 0 and 3, you can reach all other nodes in the graph. Starting from node 0, you can reach nodes 1, 2, and 5. Starting from node 3, you can reach nodes 4 and 2 (and by extension 5).

Pattern: Island (Matrix Traversal)

93. Number of Islands

Problem Statement

Given a 2D array (i.e., a matrix) containing only 1s (land) and 0s (water), count the number of islands in it.

An **island** is a connected set of 1s (land) and is surrounded by either an edge or 0s (water). Each cell is considered connected to other cells horizontally or vertically (not diagonally).

Examples:

Example 1:

```
Input:
[
 [1, 1, 0, 0, 0],
 [1, 1, 0, 0, 0],
 [0, 0, 1, 0, 0],
 [0, 0, 0, 1, 1]
]

Output:
3
```

Example 2:

```
Input:
[
 [1, 1, 0, 0, 0],
 [0, 1, 0, 0, 1],
 [1, 0, 0, 1, 1],
 [0, 0, 0, 0, 0]
]

Output:
4
```

---

94. Biggest Island

Problem Statement

Given a 2D array (i.e., a matrix) containing only 1s (land) and 0s (water), find the biggest island in it. Write a function to return the area of the biggest island.

An **island** is a connected set of 1s (land) and is surrounded by either an edge or 0s (water). Each cell is considered connected to other cells horizontally or vertically (not diagonally).

Examples:

Example 1:

```
Input:
[
 [1, 1, 0, 0, 0],
 [1, 1, 0, 0, 0],
 [0, 0, 1, 0, 0],
 [0, 0, 0, 1, 1]
]

Output:
4
```

Example 2:

```
Input:
[
 [1, 1, 0, 0, 0],
 [0, 1, 0, 0, 1],
 [1, 0, 0, 1, 1],
 [0, 0, 0, 0, 0]
]

Output:
5
```

---

95. Flood Fill

Problem Statement

Any image can be represented by a 2D integer array (i.e., a matrix) where each cell represents the pixel value of the image.

Flood fill algorithm takes a starting cell (i.e., a pixel) and a color. The given color is applied to all horizontally and vertically connected cells with the same color as that of the starting cell. Recursively, the algorithm fills cells with the new color until it encounters a cell with a different color than the starting cell.

Given a matrix, a starting cell, and a color, flood fill the matrix.

Examples:

Example 1:

```
Input:
image = [
 [1, 1, 1],
 [1, 1, 0],
 [1, 0, 1]
]
sr = 1
sc = 1
newColor = 2

Output:
[
 [2, 2, 2],
 [2, 2, 0],
 [2, 0, 1]
]
```

Example 2:

```
Input:
image = [
 [1, 1, 1, 1],
 [1, 1, 0, 0],
 [1, 0, 1, 1],
 [0, 1, 1, 0]
]
sr = 2
sc = 3
newColor = 2

Output:
[
 [1, 1, 1, 1],
 [1, 1, 0, 0],
 [1, 0, 2, 2],
 [0, 1, 1, 0]
]
```

96. Number of Closed Islands

Problem Statement

You are given a 2D matrix containing only 1s (land) and 0s (water).

An **island** is a connected set of 1s (land) and is surrounded by either an edge or 0s (water). Each cell is considered connected to other cells horizontally or vertically (not diagonally).

A **closed island** is an island that is totally surrounded by 0s (i.e., water). This means all horizontally and vertically connected cells of a closed island are water. This also means that, by definition, a closed island can't touch an edge (as then the edge cells are not connected to any water cell).

---

97. Problem Challenge 1

Problem Statement

You are given a 2D matrix containing only 1s (land) and 0s (water).

An **island** is a connected set of 1s (land) and is surrounded by either an edge or 0s (water). Each cell is considered connected to other cells horizontally or vertically (not diagonally).

There are **no lakes** on the island, so the water inside the island is not connected to the water around it. A cell is a square with a side length of 1..

The given matrix has only one island, write a function to find the perimeter of that island.

---

98. Problem Challenge 2

Problem Statement

You are given a 2D matrix containing only 1s (land) and 0s (water).

An **island** is a connected set of 1s (land) and is surrounded by either an edge or 0s (water). Each cell is considered connected to other cells horizontally or vertically (not diagonally).

Two islands are considered the same if and only if they can be translated (not rotated or reflected) to equal each other.

Write a function to find the number of distinct islands in the given matrix.

---

99. Problem Challenge 3

Problem Statement

You are given a 2D matrix containing different characters, you need to find if there exists any cycle consisting of the same character in the matrix.

A **cycle** is a path in the matrix that starts and ends at the same cell and has **four or more cells**. From a given cell, you can move to one of the cells adjacent to it - in one of the four directions (up, down, left, or right), if it has the **same character value** of the current cell.

Write a function to find if the matrix has a cycle.

Pattern: Two Heaps

100. Find the Median of a Number Stream

Problem Statement

Design a class to calculate the median of a number stream. The class should have the following two methods:

1. `insertNum(int num)`: stores the number in the class
2. `findMedian()`: returns the median of all numbers inserted in the class

If the count of numbers inserted in the class is even, the median will be the average of the middle two numbers.

Example 1:

```
1. insertNum(3)
2. insertNum(1)
3. findMedian() -> output: 2
4. insertNum(5)
5. findMedian() -> output: 3
6. insertNum(4)
7. findMedian() -> output: 3.5
```

---

101. Sliding Window Median

Problem Statement

Given an array of numbers and a number ‘k’, find the median of all the ‘k’ sized sub-arrays (or windows) of the array.

Example 1:

Input: nums=[1, 2, -1, 3, 5], k = 2
Output: [1.5, 0.5, 1.0, 4.0]

Explanation: Let's consider all windows of size ‘2’:

[**1, 2**, -1, 3, 5] -> median is 1.5
[1, **2, -1**, 3, 5] -> median is 0.5
[1, 2, **-1, 3**, 5] -> median is 1.0
[1, 2, -1, **3, 5**] -> median is 4.0

---

102. Maximize Capital

Problem Statement

Given a set of investment projects with their respective profits, we need to find the **most profitable projects**. We are given an initial capital and are allowed to invest only in a fixed number of projects. Our goal is to choose projects that give us the maximum profit. Write a function that returns the maximum total capital after selecting the most profitable projects.

We can start an investment project only when we have the required capital

---

103. Next Interval

Problem Statement

Given an array of intervals, find the next interval of each interval. In a list of intervals, for an interval ‘i’ its next interval ‘j’ will have the smallest ‘start’ greater than or equal to the ‘end’ of ‘i’.

Write a function to return an array containing indices of the next interval of each input interval. If there is no next interval of a given interval, return -1. It is given that none of the intervals have the same start point.

Example 1:

```
Input: Intervals [[2,3], [3,4], [5,6]]
Output: [1, 2, -1]
```

Pattern: Subsets

104. Subsets

Problem Statement

Given a set with distinct elements, find all of its distinct subsets.

Example 1:

```
Input: [1, 3]
Output: [], [1], [3], [1,3]
```

Example 2:

```
Input: [1, 5, 3]
Output: [], [1], [5], [3], [1,5], [1,3], [5,3], [1,5,3]
```

---

105. Subsets With Duplicates

Problem Statement

Given a set of numbers that might contain duplicates, find all of its distinct subsets.

Example 1:

```
Input: [1, 3, 3]
Output: [], [1], [3], [1,3], [3,3], [1,3,3]
```

Example 2:

```
Input: [1, 5, 3, 3]
Output: [], [1], [5], [3], [1,5], [1,3], [5,3], [1,5,3], [3,3], [1,3,3],
[3,3,5], [1,5,3,3]
```

---

106. Permutations

Problem Statement

Given a set of distinct numbers, find all of its permutations.

Permutation is defined as the re-arranging of the elements of the set. For example, {1, 2, 3} has the following six permutations:

{1, 2, 3} {1, 3, 2} {2, 1, 3} {2, 3, 1} {3, 1, 2} {3, 2, 1}

If a set has distinct elements it will have permutations.

Example 1:

```
Input: [1,3,5]
Output: [1,3,5], [1,5,3], [3,1,5], [3,5,1], [5,1,3], [5,3,1]
```

---

107. String Permutations by changing case

Problem Statement

Given a string, find all of its permutations preserving the character sequence but changing case.

Example 1:

```
Input: "ad52"
Output: "ad52", "Ad52", "aD52", "AD52"
```

Example 2:

```
Input: "ab7c"
Output: "ab7c", "Ab7c", "aB7c", "AB7c", "ab7C", "Ab7C", "aB7C", "AB7C"
```

---

108. Balanced Parentheses

Problem Statement

For a given number ‘N’, write a function to generate all combination of ‘N’ pairs of balanced parentheses.

Example 1:

```
Input: N=2
Output: (()), ()()
```

Example 2:

```
Input: N=3
Output: ((())), (()()), (())(), ()(()), ()()()
```

---

109. Unique Generalized Abbreviations

Problem Statement

Given a word, write a function to generate all of its unique generalized abbreviations.

A generalized abbreviation of a word can be generated by replacing each substring of the word with the count of characters in the substring. Take the example of “ab” which has four substrings: “”, “a”, “b”, and “ab”. After replacing these substrings in the actual word by the count of characters, we get all the generalized abbreviations: “ab”, “1b”, “a1”, and “2”.

Note: All contiguous characters should be considered one substring

---

110. Evaluate Expression

Problem Statement

Given an expression containing digits and operations (+, -, *), find all possible ways in which the expression can be evaluated by grouping the numbers and operators using parentheses.

Example 1:

```
Input: "1+2*3"
Output: 7, 9
Explanation:
  1+(2*3) => 7
  (1+2)*3 => 9
```

Example 2:

```
Input: "2*3-4-5"
Output: 8, -12, 7, -7, -3
Explanation:
  2*(3-(4-5)) => 8
  2*(3-4-5) => -12
  2*3-(4-5) => 7
  2*(3-4)-5 => -7
  (2*3)-4-5 => -3
```

---

111. Structurally Unique Binary Search Trees

Problem Statement

Given a number ‘n’, write a function to return all structurally unique Binary Search Trees (BST) that can store values 1 to ‘n’?

Example 1:

```
Input: 2
Output: List containing root nodes of all structurally unique BSTs.
Explanation: Here are the 2 structurally unique BSTs storing all numbers
from 1 to 2:
```

Example 2:

```
Input: 3
Output: List containing root nodes of all structurally unique BSTs.
Explanation: Here are the 5 structurally unique BSTs storing all numbers
from 1 to 3:
```

---

112. Count of Structurally Unique Binary Search Trees

Problem Statement

Given a number ‘n’, write a function to return the count of structurally unique Binary Search Trees (BST) that can store values 1 to ‘n’.

Example 1:

```
Input: 2
Output: 2
Explanation: As we saw in the previous problem, there are 2 unique BSTs storing numbers from 1-2.
```

Example 2:

```
Input: 3
Output: 5
Explanation: There will be 5 unique BSTs that can store numbers from 1 to 3.
```

Pattern: Modified Binary Search

113. Order-agnostic Binary Search

Problem Statement

Given a sorted array of numbers, find if a given number ‘key’ is present in the array. Though we know that the array is sorted, we don’t know if it’s sorted in ascending or descending order. You should assume that the array can have duplicates.

Write a function to return the index of the ‘key’ if it is present in the array, otherwise return -1.

Example 1:

```
Input: [4, 6, 10], key = 10
Output: 2
```

Example 2:

```
Input: [1, 2, 3, 4, 5, 6, 7], key = 5
Output: 4
```

Example 3:

```
Input: [10, 6, 4], key = 10
Output: 0
```

---

114. Ceiling of a Number

Problem Statement

Given an array of numbers sorted in an ascending order, find the ceiling of a given number ‘key’. The ceiling of the ‘key’ will be the smallest element in the given array greater than or equal to the ‘key’.

Write a function to return the index of the ceiling of the ‘key’. If there isn’t any ceiling return -1.

Example 1:

```
Input: [4, 6, 10], key = 6
Output: 1
Explanation: The smallest number greater than or equal to '6' is '6' having
index '1'.
```

Example 2:

```
Input: [1, 3, 8, 10, 15], key = 12
Output: 4
```

---

115. Next Letter

Problem Statement

Given an array of lowercase letters sorted in ascending order, find the **smallest letter** in the given array **greater than a given ‘key’**.

Assume the given array is a **circular list**, which means that the last letter is assumed to be connected with the first letter. This also means that the smallest letter in the given array is greater than the last letter of the array and is also the first letter of the array.

Write a function to return the next letter of the given ‘key’.

Example 1:

```
Input: ['a', 'c', 'f', 'h'], key = 'f'
Output: 'h'
```

---

116. Number Range

Problem Statement

Given an array of numbers sorted in ascending order, find the range of a given number ‘key’. The range of the ‘key’ will be the first and last position of the ‘key’ in the array.

Write a function to return the range of the ‘key’. If the ‘key’ is not present return [-1, -1].

Example 1:

```
Input: [4, 6, 6, 6, 9], key = 6
Output: [1, 3]
```

Example 2:

```
Input: [1, 3, 8, 10, 15], key = 10
Output: [3, 3]
```

Example 3:

```
Input: [1, 3, 8, 10, 15], key = 12
Output: [-1, -1]
```

---

117. Search in a Sorted Infinite Array

Problem Statement

Given an infinite sorted array (or an array with unknown size), find if a given number ‘key’ is present in the array. Write a function to return the index of the ‘key’ if it is present in the array, otherwise return -1.

Since it is not possible to define an array with infinite (unknown) size, you will be provided with an interface `ArrayReader` to read elements of the array. `ArrayReader.get(index)` will return the number at index; if the array’s size is smaller than the index, it will return `Integer.MAX_VALUE`.

---

118. Minimum Difference Element

Problem Statement

Given an array of numbers sorted in ascending order, find the element in the array that has the minimum difference with the given ‘key’.

Example 1:

```
Input: [4, 6, 10], key = 7
Output: 6
Explanation: The difference between the key '7' and '6' is minimum than any
other number in the array
```

Example 2:

```
Input: [4, 6, 10], key = 4
Output: 4
```

Example 3:

```
Input: [1, 3, 8, 10, 15], key = 12
Output: 10
```

Example 4:

```
Input: [4, 6, 10], key = 17
Output: 10
```

---

119. Bitonic Array Maximum

Problem Statement

Find the maximum value in a given Bitonic array. An array is considered bitonic if it is monotonically increasing and then monotonically decreasing. Monotonically increasing or decreasing means that for any index i in the array `arr[i] != arr[i+1]`.

Example 1:

```
Input: [1, 3, 8, 12, 4, 2]
Output: 12
Explanation: The maximum number in the input bitonic array is '12'.
```

Example 2:

```
Input: [3, 8, 3, 1]
Output: 8
```

Example 3:

```
Input: [1, 3, 8, 12]
Output: 12
```

Example 4:

```
Input: [10, 9, 8]
Output: 10
```

---

120. Search Bitonic Array

Problem Statement

Given a Bitonic array, find if a given ‘key’ is present in it. An array is considered bitonic if it is monotonically increasing and then monotonically decreasing. Monotonically increasing or decreasing means that for any index i in the array `arr[i] != arr[i+1]`.

Write a function to return the index of the ‘key’. If the 'key' appears more than once, return the smaller index. If the ‘key’ is not present, return -1.

Example 1:

```
Input: [1, 3, 8, 4, 3], key=4
Output: 3
```

Example 2:

```
Input: [3, 8, 3, 1], key=8
Output: 1
```

---

121. Search in Rotated Array

Problem Statement

Given an array of numbers which is sorted in ascending order and also rotated by some arbitrary number, find if a given ‘key’ is present in it.

Write a function to return the index of the ‘key’ in the rotated array. If the ‘key’ is not present, return -1. You can assume that the given array does not have any duplicates.

Example 1:

```
Input: [10, 15, 1, 3, 8], key = 15
Output: 1
Explanation: '15' is present in the array at index '1'.
```

Example 2:

```
Input: [4, 5, 7, 9, 10, -1, 2], key = 10
Output: 4
Explanation: '10' is present in the array at index '4'.
```

---

122. Rotation Count

Problem Statement

Given an array of numbers which is sorted in ascending order and is rotated ‘k’ times around a pivot, find ‘k’.

You can assume that the array does not have any duplicates.

Example 1:

```
Input: [10, 15, 1, 3, 8]
Output: 2
Explanation: The array has been rotated 2 times.
```

Example 2:

```
Input: [4, 5, 7, 9, 10, -1, 2]
Output: 5
Explanation: The array has been rotated 5 times.
```

Example 3:

```
Input: [1, 3, 8, 10]
Output: 0
Explanation: The array has not been rotated.
```

Pattern: Bitwise XOR

123. Single Number

Problem Statement

In a non-empty array of integers, every number appears twice except for one, find that single number.

Example 1:

```
Input: 1, 4, 2, 1, 3, 2, 3
Output: 4
```

Example 2:

```
Input: 7, 9, 7
Output: 9
```

---

124. Two Single Numbers

Problem Statement

In a non-empty array of numbers, every number appears exactly twice except two numbers that appear only once. Find the two numbers that appear only once.

Example 1:

```
Input: [1, 4, 2, 1, 3, 5, 6, 2, 3, 5]
Output: [4, 6]
```

Example 2:

```
Input: [2, 1, 3, 2]
Output: [1, 3]
```

---

125. Complement of Base 10 Number

Problem Statement

Every non-negative integer N has a binary representation, for example, 8 can be represented as “1000” in binary and 7 as “0111” in binary.

The complement of a binary representation is the number in binary that we get when we change every 1 to a 0 and every 0 to a 1. For example, the binary complement of “1010” is “0101”.

For a given positive number N in base-10, return the complement of its binary representation as a base-10 integer.

---

126. Flip and Invert an Image

Problem Statement

Given a square binary matrix representing an image, we want to flip the image horizontally, then invert it.

To flip an image horizontally means that each row of the image is reversed. For example, flipping [0, 1, 1] horizontally results in [1, 1, 0].

To invert an image means that each 0 is replaced by 1, and each 1 is replaced by 0. For example, inverting [1, 1, 0] results in [0, 0, 1].

Example 1:

```
Input: [
  [1,0,1],
  [1,1,1],
  [0,1,1]
]
Output: [
  [0,1,0],
  [0,0,0],
  [0,0,1]
]
```

Explanation: First reverse each row: [[1,0,1],[1,1,1],[1,1,0]]

Pattern: Top 'K' Elements

127. Top *K* Numbers

Problem Statement

Given an unsorted array of numbers, find the ‘K’ largest numbers in it.

Example 1:

```
Input: [3, 1, 5, 12, 2, 11], K = 3
Output: [5, 12, 11]
```

Example 2:

```
Input: [5, 12, 11, -1, 12], K = 3
Output: [12, 11, 12]
```

---

128. Kth Smallest Number

Problem Statement

Given an unsorted array of numbers, find Kth smallest number in it.

Please note that it is the Kth smallest number in the sorted order, not the Kth distinct element.

Note: For a detailed discussion about different approaches to solve this problem, take a look at Kth Smallest Number.

Example 1:

```
Input: [1, 5, 12, 2, 11, 5], K = 3
Output: 5
Explanation: The 3rd smallest number is '5', as the first two smaller
numbers are [1, 2].
```

---

129. *K* Closest Points to the Origin

Problem Statement

Given an array of points in a 2D plane, find ‘K’ closest points to the origin.

Example 1:

```
Input: points = [[1,2],[1,3]], K = 1
Output: [[1,2]]
Explanation: The Euclidean distance between (1, 2) and the origin is
sqrt(5).
The Euclidean distance between (1, 3) and the origin is sqrt(10).
Since sqrt(5) < sqrt(10), therefore (1, 2) is closer to the origin.
```

Example 2:

```
Input: point = [[1, 3], [3, 4], [2, -1]], K = 2
Output: [[1, 3], [2, -1]]
```

---

130. Connect Ropes

Problem Statement

Given 'N' ropes with different lengths, we need to connect these ropes into one big rope with minimum cost. The cost of connecting two ropes is equal to the sum of their lengths.

Example 1:

```
Input: [1, 3, 11, 5]
Output: 33
Explanation: First connect 1+3(=4), then 4+5(=9), and then 9+11(=20). So the
total cost is 33 (4+9+20)
```

Example 2:

```
Input: [3, 4, 5, 6]
Output: 36
Explanation: First connect 3+4(=7), then 5+6(=11), 7+11(=18). Total cost is
36 (7+11+18)
```

Example 3:

```
Input: [1, 3, 11, 5, 2]
Output: 42
```

---

131. Top *K* Frequent Numbers

Problem Statement

Given an unsorted array of numbers, find the top 'K' frequently occurring numbers in it.

Example 1:

```
Input: [1, 3, 5, 12, 11, 12, 11], K = 2
Output: [12, 11]
Explanation: Both '11' and '12' apeared twice.
```

Example 2:

```
Input: [5, 12, 11, 3, 11], K = 2
Output: [11, 5] or [11, 12] or [11, 3]
Explanation: Only '11' appeared twice, all other numbers appeared once.
```

---

132. Frequency Sort

Problem Statement

Given a string, sort it based on the decreasing frequency of its characters.

Example 1:

```
Input: "Programming"
Output: "rrggmmPiano"
Explanation: 'r', 'g', and 'm' appeared twice, so they need to appear before any other character.
```

Example 2:

```
Input: "abcbab"
Output: "bbbaac"
Explanation: 'b' appeared three times, 'a' appeared twice, and 'c' appeared only once.
```

---

133. Kth Largest Number in a Stream

Problem Statement

Design a class to efficiently find the Kth largest element in a stream of numbers.

The class should have the following two things:

1. The constructor of the class should accept an integer array containing initial numbers from the stream and an integer ‘K’.
2. The class should expose a function `add(int num)` which will store the given number and return the Kth largest number.

Example 1:

```
Input: [3, 1, 5, 12, 2, 11], K = 4
1. Calling add(6) should return '5'.
2. Calling add(13) should return '6'.
2. Calling add(4) should still return '6'.
```

---

134. *K* Closest Numbers

Problem Statement

Given a sorted number array and two integers ‘K’ and ‘X’, find ‘K’ closest numbers to ‘X’ in the array. Return the numbers in the sorted order. ‘X’ is not necessarily present in the array.

Example 1:

```
Input: [5, 6, 7, 8, 9], K = 3, X = 7
Output: [6, 7, 8]
```

Example 2:

```
Input: [2, 4, 5, 6, 9], K = 3, X = 6
Output: [4, 5, 6]
```

Example 3:

```
Input: [2, 4, 5, 6, 9], K = 3, X = 10
Output: [5, 6, 9]
```

---

135. Maximum Distinct Elements

Problem Statement

Given an array of numbers and a number ‘K’, we need to remove ‘K’ numbers from the array such that we are left with maximum distinct numbers.

Example 1:

```
Input: [7, 3, 5, 8, 5, 3, 3], and K=2
Output: 3
Explanation: We can remove two occurrences of 3 to be left with 3 distinct
numbers [7, 3, 8], we have
to skip 5 because it is not distinct and occurred twice.
Another solution could be to remove one instance of '5' and '3' each to be
left with three distinct numbers [7, 5, 8], in this case, we have to skip 3
because it occurred twice.
```

---

136. Sum of Elements

Problem Statement

Given an array, find the sum of all numbers between the K1'th and K2'th smallest elements of that array.

Example 1:

```
Input: [1, 3, 12, 5, 15, 11], and K1=3, K2=6
Output: 23
Explanation: The 3rd smallest number is 5 and 6th smallest number 15. The
sum of numbers coming
between 5 and 15 is 23 (11+12).
```

Example 2:

```
Input: [3, 5, 8, 7], and K1=1, K2=4
Output: 12
Explanation: The sum of the numbers between the 1st smallest number (3) and
the 4th smallest
number (8) is 12 (5+7).
```

---

137. Rearrange String

Problem Statement

Given a string, find if its letters can be rearranged in such a way that no two same characters come next to each other.

Example 1:

```
Input: "aappp"
Output: "papap"
Explanation: In "papap", none of the repeating characters come next to each
other.
```

Example 2:

```
Input: "Programming"
Output: "rgmrgmPiano" or "gmringmrPoa" or "gmrPagimnor", etc.
Explanation: None of the repeating characters come next to each other.
```

Example 3:

```
Input: "aapa"
Output: ""
```

```
Explanation: In all arrangements of "aapa", atleast two 'a' will come
together
```

---

138. Rearrange String K Distance Apart

Problem Statement

Given a string and a number ‘K’, find if the string can be rearranged such that the same characters are at least ‘K’ distance apart from each other.

Example 1:

```
Input: "mmpp", K=2
Output: "mpmp" or "pmpm"
Explanation: All same characters are 2 distance apart.
```

Example 2:

```
Input: "Programming", K=3
Output: "rgmPrgmiano" or "gmringmrPoa" or "gmrPagimnor" and a few more
Explanation: All same characters are 3 distance apart.
```

Example 3:

```
Input: "aab", K=2
Output: "aba"
Explanation: All same characters are 2 distance apart.
```

---

139. Scheduling Tasks

Problem Statement

You are given a list of tasks that need to be run, in any order, on a server. Each task will take one CPU interval to execute but once a task has finished, it has a cooling period during which it can’t be run again. If the cooling period for all tasks is ‘K’ intervals, find the minimum number of CPU intervals that the server needs to finish all tasks.

If at any time the server can’t execute any task then it must stay idle.

Example 1:

```
Input: [a, a, a, b, c, c], K=2
Output: 7
Explanation: a -> c -> b -> a -> c -> idle -> a
```

---

140. Frequency Stack

Problem Statement

Design a class that simulates a Stack data structure, implementing the following two operations:

1. `push(int num)`: Pushes the number 'num' on the stack.
2. `pop()`: Returns the most frequent number in the stack. If there is a tie, return the number which was pushed later.

Example:

```
After following push operations: push(1), push(2), push(3), push(2),
push(1), push(2), push(5)

1. pop() should return 2, as it is the most frequent number
2. Next pop() should return 1
3. Next pop() should return 2
```

Pattern: K-way Merge

141. Merge K Sorted Lists

Problem Statement

Given an array of 'K' sorted LinkedLists, merge them into one sorted list.

Example 1:

```
Input: L1=[2, 6, 8], L2=[3, 6, 7], L3=[1, 3, 4]
Output: [1, 2, 3, 3, 4, 6, 6, 7, 8]
```

Example 2:

```
Input: L1=[5, 8, 9], L2=[1, 7]
Output: [1, 5, 7, 8, 9]
```

---

142. Kth Smallest Number in M Sorted Lists

Problem Statement

Given 'M' sorted arrays, find the K'th smallest number among all the arrays.

Example 1:

```
Input: L1=[2, 6, 8], L2=[3, 6, 7], L3=[1, 3, 4], K=5
Output: 4
Explanation: The 5th smallest number among all the arrays is 4, this can be verified from
the merged list of all the arrays: [1, 2, 3, 3, 4, 6, 6, 7, 8]
```

Example 2:

```
Input: L1=[5, 8, 9], L2=[1, 7], K=3
Output: 7
Explanation: The 3rd smallest number among all the arrays is 7.
```

---

143. Kth Smallest Number in a Sorted Matrix

Problem Statement

Given an N * N matrix where each row and column is sorted in ascending order, find the Kth smallest element in the matrix.

Example 1:

```
Input: Matrix=[
    [2, 6, 8],
    [3, 7, 10],
    [5, 8, 11]
  ],
  K=5
Output: 7
Explanation: The 5th smallest number in the matrix is 7.
```

---

144. Smallest Number Range

Problem Statement

Given ‘M’ sorted arrays, find the smallest range that includes at least one number from each of the ‘M’ lists.

Example 1:

```
Input: L1=[1, 5, 8], L2=[4, 12], L3=[7, 8, 10]
Output: [4, 7]
Explanation: The range [4, 7] includes 5 from L1, 4 from L2 and 7 from L3.
```

Example 2:

```
Input: L1=[1, 9], L2=[4, 12], L3=[7, 10, 16]
Output: [9, 12]
Explanation: The range [9, 12] includes 9 from L1, 12 from L2 and 10 from L3
```

---

145. K Pairs with Largest Sums

Problem Statement

Given two sorted arrays in descending order, find ‘K’ pairs with the largest sum where each pair consists of numbers from both the arrays.

Example 1:

```
Input: L1=[9, 8, 2], L2=[6, 3, 1], K=3
Output: [9, 3], [9, 6], [8, 6]
Explanation: These 3 pairs have the largest sum. No other pair has a sum larger than any of these.
```

Example 2:

```
Input: L1=[5, 2, 1], L2=[2, -1], K=3
Output: [5, 2], [5, -1], [2, 2]
```

Pattern: Greedy Algorithms

146. Valid Palindrome II

Problem Statement

Given string `s`, determine whether it's possible to make a given string palindrome by removing at most one character.

A palindrome is a word or phrase that reads the same backward as forward.

Examples

1. **Example 1:**
   - Input: `"racecar"`
   - Expected Output: `true`
   - Justification: The string is already a palindrome, so no removals are needed.
2. **Example 2:**
   - Input: `"abccdba"`
   - Expected Output: `true`

- Justification: Removing the character 'd' forms the palindrome "abccba".

3. **Example 3:**
   - Input: `"abcdef"`
   - Expected Output:

---

147. Maximum Length of Pair Chain

Problem Statement

Given a collection of pairs where each pair contains two elements [a, b], find the maximum length of a chain you can form using pairs.

A pair [a, b] can follow another pair [c, d] in the chain if b < c.

You can select pairs in any order and don't need to use all the given pairs.

Examples

1. **Example 1**:
   - **Input**: `[[1,2], [3,4], [2,3]]`
   - **Expected Output**: `2`
   - **Justification**: The longest chain is `[1,2] -> [3,4]`. The chain `[1,2] -> [2,3]` is invalid because 2 is not smaller than 2.

---

148. Minimum Add to Make Parentheses Valid

Problem Statement

Given a string str containing '(' and ')' characters, find the minimum number of parentheses that need to be added to a string of parentheses to make it valid.

A valid string of parentheses is one where each opening parenthesis '(' has a corresponding closing parenthesis ')' and vice versa. The goal is to determine the least amount of additions needed to achieve this balance.

Examples

1. **Example 1:**
   - **Input:** "(()"
   - **Expected Output:** 1
   - **Justification:** The string has two opening parentheses and one closing parenthesis

---

149. Remove Duplicate Letters

Problem Statement

Given a string `s`, remove all duplicate letters from the input string while maintaining the original order of the letters.

Additionally, the returned string should be the smallest in lexicographical order among all possible results.

Examples:

1.
   - **Input:** "bbaac"
   - **Expected Output:** "bac"
   - **Justification:** Removing the extra 'b' and one 'a' from the original string gives 'bac', which is the smallest lexicographical string without duplicate letters.
2.
   - **Input:** "zabccdef"
   - **Expected Output:** "zabcdef"

---

150. Largest Palindromic Number

Problem Statement

Given a string `s` containing 0 to 9 digits, create the largest possible palindromic number using the string characters.

A palindromic number reads the same backward as forward.

If it's not possible to form such a number using all digits of the given string, you can skip some of them.

Examples

1.
   - **Input:** "323211444"
   - **Expected Output:** "432141234"
   - **Justification:** This is the largest palindromic number that can be formed from the given digits.
2.
   - **Input:** "998877"
   - **Expected Output:** "987789"

151. Removing Minimum and Maximum From Array

Problem Statement

Determine the `minimum` number of `deletions` required to remove the smallest and the largest elements from an array of integers.

In each `deletion`, you are allowed to remove either the `first (leftmost)` or the `last (rightmost)` element of the array.

Examples

1. **Example 1:**
    - Input: `[3, 2, 5, 1, 4]`
    - Expected Output: `3`
    - Justification: The smallest element is `1` and the largest is `5`. Removing `4`, `1`, and then `5` (or `5`, `4`, and then `1`) in three moves is the most efficient strategy.

Pattern: 0/1 Knapsack (Dynamic Programming)

152. Knapsack

Problem Statement

Given the weights and profits of ‘N’ items, we are asked to put these items in a knapsack with a capacity ‘C.’ The goal is to get the maximum profit out of the knapsack items. Each item can only be selected once, as we don’t have multiple quantities of any item.

Let’s take Merry’s example, who wants to carry some fruits in the knapsack to get maximum profit. Here are the weights and profits of the fruits:

Items: { Apple, Orange, Banana, Melon }

Weights: { 2, 3, 1, 4 }

Profits: { 4, 5, 3, 7 }

Knapsack capacity: 5

---

153. Equal Subset Sum Partition

Problem Statement

Given a set of positive numbers, find if we can partition it into two subsets such that the sum of elements in both subsets is equal.

Example 1:

```
Input: {1, 2, 3, 4}
Output: True
Explanation: The given set can be partitioned into two subsets with equal sum: {1, 4} & {2, 3}
```

Example 2:

```
Input: {1, 1, 3, 4, 7}
Output: True
Explanation: The given set can be partitioned into two subsets with equal sum: {1, 3, 4} & {1, 7}
```

Example 3:

```
Input: {2, 3, 4, 6}
Output: False
Explanation: The given set cannot be partitioned into two subsets with equal sum.
```

---

154. Subset Sum

Problem Statement

Given a set of positive numbers, determine if a subset exists whose sum is equal to a given number ‘S’.

Example 1:

```
Input: {1, 2, 3, 7}, S=6
Output: True
The given set has a subset whose sum is '6': {1, 2, 3}
```

Example 2:

```
Input: {1, 2, 7, 1, 5}, S=10
Output: True
The given set has a subset whose sum is '10': {1, 2, 7}
```

Example 3:

```
Input: {1, 3, 4, 8}, S=6
Output: False
```

```
The given set does not have any subset whose sum is equal to '6'.
```

---

155. Minimum Subset Sum Difference

Problem Statement

Given a set of positive numbers, partition the set into two subsets with minimum difference between their subset sums.

Example 1:

```
Input: {1, 2, 3, 9}
Output: 3
Explanation: We can partition the given set into two subsets where minimum absolute difference
between the sum of numbers is '3'. Following are the two subsets: {1, 2, 3} & {9}.
```

Example 2:

```
Input: {1, 2, 7, 1, 5}
Output: 0
Explanation: We can partition the given set into two subsets where minimum absolute difference
between the sum of number is '0'
```

---

156. Count of Subset Sum

Problem Statement

Given a set of positive numbers, find the total number of subsets whose sum is equal to a given number ‘S’.

Example 1:

```
Input: {1, 1, 2, 3}, S=4
Output: 3
The given set has '3' subsets whose sum is '4': {1, 1, 2}, {1, 3}, {1, 3}
Note that we have two similar sets {1, 3}, because we have two '1' in our input.
```

Example 2:

```
Input: {1, 2, 7, 1, 5}, S=9
Output: 3
```

```
The given set has '3' subsets whose sum is '9': {2, 7}, {1, 7, 1}, {1, 2, 1, 5}
```

---

157. Target Sum

Problem Statement

You are given a set of positive numbers and a target sum ‘S’. Each number should be assigned either a ‘+’ or ‘-’ sign. We need to find the total ways to assign symbols to make the sum of the numbers equal to the target ‘S’.

Example 1:

```
Input: {1, 1, 2, 3}, S=1
Output: 3
Explanation: The given set has '3' ways to make a sum of '1': {+1-1-2+3} & {-1+1-2+3} & {+1+1+2-3}
```

Example 2:

```
Input: {1, 2, 7, 1}, S=9
Output: 2
Explanation: The given set has '2' ways to make a sum of '9': {+1+2+7-1} & {-1+2+7+1}
```

Pattern: Backtracking

158. Combination Sum

Problem Statement

Given an array of distinct positive 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.

Example 1:

```
Input: candidates = [2, 3, 6, 7], target = 7
Output: [[2, 2, 3], [7]]
Explanation: The elements in these two combinations sum up to 7.
```

Example 2:

```
Input: candidates = [2, 4, 6, 8], target = 10
Output: [[2,2,2,2,2], [2,2,2,4], [2,2,6], [2,4,4], [2,8], [4,6]]
Explanation: The elements in these six combinations sum up to 10.
```

---

159. Word Search

Problem Statement

Given an m x n grid of characters board and a string word, return true if the 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.

---

160. Sudoku Solver

Problem Statement

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

A sudoku solution must satisfy all of the following rules:

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

The '.' character indicates empty cells.

---

161. Factor Combinations

Problem Statement

Numbers can be regarded as the product of their factors.

For example, 8 = 2 x 2 x 2 = 2 x 4.

Given an integer n, return all possible combinations of its factors. You may return the answer in **any order**.

Example 1:

```
Input: n = 8
Output: [[2, 2, 2], [2, 4]]
```

Example 2:

```
Input: n = 20
Output: [[2, 2, 5], [2, 10], [4, 5]]
```

---

162. Split a String Into the Max Number of Unique

Problem Statement

Given a string `s`, return the maximum number of unique substrings that the given string can be split into.

You can split string `s` into any list of **non-empty substrings**, where the concatenation of the substrings forms the original string. However, you must split the substrings such that all of them are **unique**.

A substring is a contiguous sequence of characters within a string.

Pattern: Trie

163. Implement Trie (Prefix Tree)

Problem Statement

Design and implement a Trie (also known as a Prefix Tree). A trie is a tree-like data structure that stores a dynamic set of strings, and is particularly useful for searching for words with a given prefix.

Implement the `Solution` class:

- `Solution()` Initializes the object.
- `void insert(word)` Inserts `word` into the trie, making it available for future searches.
- `bool search(word)` Checks if the word exists in the trie.
- `bool startsWith(word)` Checks if any word in the trie starts with the given prefix.

---

164. Index Pairs of a String

Problem Statement

Given a string `text` and a list of strings `words`, identify all `[i, j]` index pairs such that the substring `text[i...j]` is in words.

These index pairs should be returned in ascending order, first by the start index, then by the end index. Find every occurrence of each word within the `text`, ensuring that overlapping occurrences are also identified.

Examples

1.
    - **Input:** text = `"bluebirdskyscraper"`, words = `["blue", "bird", "sky"]`
    - **Expected Output:** `[[0, 3], [4, 7], [8, 10]]`

---

165. Design Add and Search Words Data Structure

Problem Statement

Design a data structure that supports the addition of new words and the ability to check if a string matches any previously added word.

Implement the `Solution` class:

- `Solution()` Initializes the object.
- `void addWord(word)` Inserts `word` into the data structure, making it available for future searches.
- `bool search(word)` Checks if there is any word in the data structure that matches `word`. The method returns `true` if such a match exists, otherwise returns `false`.

---

166. Extra Characters in a String

Problem Statement

Given a string `s` and an array of words `words`. Break string `s` into multiple non-overlapping substrings such that each substring should be part of the `words.` There are some characters left which are not part of any substring.

Return the minimum number of remaining characters in `s`, which are not part of any substring after string break-up.

Examples

1. **Example 1:**
    - **Input:** `s = "amazingracecar"`, `words = ["race", "car"]`
    - **Expected Output:** `7`

---

167. Search Suggestions System

Problem Statement

Given a list of distinct strings `products` and a string `searchWord`.

Determine a set of `product suggestions` after each character of the search word is typed. Every time a character is typed, return a `list` containing up to three product names from the `products` list that have the same prefix as the typed string.

If there are more than 3 matching products, return 3 lexicographically smallest products. These product names should be returned in lexicographical (alphabetical) order.

Pattern: Topological Sort (Graph)

168. Tasks Scheduling

Problem Statement

There are ‘N’ tasks, labeled from ‘0’ to ‘N-1’. Each task can have some prerequisite tasks which need to be completed before it can be scheduled. Given the number of tasks and a list of prerequisite pairs, find out if it is possible to schedule all the tasks.

Example 1:

```
Input: Tasks=3, Prerequisites=[0, 1], [1, 2]
Output: true
Explanation: To execute task '1', task '0' needs to finish first. Similarly,
task '1' needs
to finish before '2' can be scheduled. One possible scheduling of tasks is:
[0, 1, 2]
```

---

169. Tasks Scheduling Order

Problem Statement

There are ‘N’ tasks, labeled from ‘0’ to ‘N-1’. Each task can have some prerequisite tasks which need to be completed before it can be scheduled. Given the number of tasks and a list of prerequisite pairs, write a method to find the ordering of tasks we should pick to finish all tasks.

Example 1:

```
Input: Tasks=3, Prerequisites=[0, 1], [1, 2]
Output: [0, 1, 2]
Explanation: To execute task '1', task '0' needs to finish first. Similarly,
task '1' needs to finish before '2' can be scheduled. A possible scheduling
of tasks is: [0, 1, 2]
```

---

170. All Tasks Scheduling Orders

Problem Statement

There are ‘N’ tasks, labeled from ‘0’ to ‘N-1’. Each task can have some prerequisite tasks which need to be completed before it can be scheduled. Given the number of tasks and a list of prerequisite pairs, write a method to print all possible ordering of tasks meeting all prerequisites.

Example 1:

```
Input: Tasks=3, Prerequisites=[0, 1], [1, 2]
Output: [0, 1, 2]
Explanation: There is only possible ordering of the tasks.
```

---

171. Alien Dictionary

Problem Statement

There is a dictionary containing words from an alien language for which we don’t know the ordering of the letters. Write a method to find the correct order of the letters in the alien language. It is given that the input is a valid dictionary and there exists an ordering among its letters.

Example 1:

```
Input: Words: ["ba", "bc", "ac", "cab"]
Output: bac
```

---

172. Reconstructing a Sequence

Problem Statement

Given a sequence `originalSeq` and an array of sequences, write a method to find if `originalSeq` can be **uniquely** reconstructed from the array of sequences.

Unique reconstruction means that we need to find if `originalSeq` is the **only sequence** such that all sequences in the array are subsequences of it.

Example 1:

```
Input: originalSeq: [1, 2, 3, 4], seqs: [[1, 2], [2, 3], [3, 4]]
Output: true
```

---

173. Minimum Height Trees

Problem Statement

We are given an undirected graph that has the characteristics of a k-ary tree. In such a graph, we can choose any node as the root to make a k-ary tree. The root (or the tree) with the minimum height will be called Minimum Height Tree (MHT). There can be multiple MHTs for a graph. In this problem, we need to find all those roots which give us MHTs. Write a method to find all MHTs of the given graph and return a list of their roots.

Example 1:

```
Input: vertices: 5, Edges: [[0, 1], [1, 2], [1, 3], [2, 4]]
Output:[1, 2]
```

Pattern: Union Find

174. Redundant Connection

Problem Statement

Given an undirected graph containing `1` to `n` nodes. The graph is represented by a 2D array of `edges`, where **edges[i] = [a_i, b_i]**, represents an edge between a_i, and b_i.

Identify one edge that, if removed, will turn the graph into a tree.

A tree is a graph that is connected and has no cycles.

Assume that the graph is always reducible to a tree by removing just one edge.

If there are multiple answers, return the edge that occurs last in the input.

---

175. Number of Provinces

Problem Statement

There are `n` cities. Some of them are connected in a network. If City `A` is directly connected to City `B`, and City `B` is directly connected to City `C`, city `A` is directly connected to City `C`.

If a group of cities are connected directly or indirectly, they form a **province**.

You are given a square matrix of size `n x n`, where each cell's value indicates whether a direct connection between cities exists (1 for connected and 0 for not connected).

Determine the total number of provinces.

---

176. Is Graph Bipartite_

Problem Statement

Given a 2D array `graph[][]`, representing the undirected graph, where `graph[u]` is an array of nodes that are connected with node `u`.

Determine whether a given undirected graph is a bipartite graph.

The graph is a bipartite graph, if we can split the set of nodes into two distinct subsets such that no two nodes within the same subset are adjacent (i.e., no edge exists between any two nodes within a single subset).

Examples

1. **Example 1:**
    - **Input:** `[[1,3], [0,2], [1,3], [0,2]]`
- **Expected Output:** `true`

---

177. Path With Minimum Effort

Problem Statement

Imagine you're on a grid of size n x n, where each point has an elevation. Your task is to find a path from the top-left corner to the bottom-right corner that minimizes the effort required to travel between consecutive points, where effort is defined as the absolute difference in elevations between two points. Your goal is to determine the smallest maximum effort required for any path from the first point to the last.

Example Generation

Example 1:

- Input: `[[1,2,3],[3,8,4],[5,3,5]]`
- Expected Output: `1`

Pattern: Ordered Set

178. Merge Similar Items

Problem Statement

Given two 2D arrays of item-value pairs, named `items1` and `items2`.

- **item[i] = [value$_i$, weight$_i$]**, where each value$_i$ in these arrays is unique within its own array and is paired with a weight$_i$.

Combine these arrays such that if an item appears in both, its values are summed up. The final merged array should be sorted based on the $value_i$.

Examples

1. **Example 1:**
   - **Input:** `items1 = [[1,2],[4,3]]`, `items2 = [[2,1],[4,3],[3,4]]`
   - **Expected Output:** `[[1,2],[2,1],[3,4],[4,6]]`

---

179. 132 Pattern

Problem Statement

Given an array `nums`, containing `N` integers.

A `132 pattern` consists of three numbers, say x, y, and z, where `x < z` and `z < y`. This is often referred to as a '132' pattern because if we represent x, y, and z as 1, 3, and 2, respectively, it mimics the positional pattern in '132'.

Return `true` if such a pattern exists within a given sequence of numbers. Otherwise, return `false`.

Examples

1. **Example 1:**
   - Input: nums = [3, 5, 0, 3, 4]
   - Expected Output: True

---

180. My Calendar I

Problem Statement

Given a 2D array `nums` of size `N x 2`.

- **nums[i] = [$start_i$, end_i]**, where $start_i$ is the starting time of the event and end_i is the ending time of the event.

For each `nums[i]`, determine if a requested booking time conflicts with any existing bookings.

Return a boolean array of size `N`, representing whether the booking can be done in the given time interval.

Examples

1. **Example 1:**

- Input: nums = [[10, 20], [15, 25], [20, 30]]
- Expected Output: `[true, false, true]`

---

181. Longest Continuous Subarray

Problem Statement

Find the length of the longest contiguous subarray within an array of integers, where the absolute difference between any two elements in this subarray does not exceed a specified limit. The challenge lies in ensuring the subarray is continuous and the range (difference between the maximum and minimum element in the subarray) fits within the given threshold.

Examples

1. **Example 1:**
   - **Input:** Array = [10, 1, 2, 4, 7], Limit = 5
   - **Expected Output:** 3

Pattern: Multi-threaded

182. Same Tree

Problem Statement

Given the roots of two binary trees 'p' and 'q', write a function to check if they are the same or not.

Two binary trees are considered the same if they met following two conditions:

1. Both tree are structurally identical.
2. Each corresponding node on both the trees have the same value.

Example 1:

Given the following two binary trees:

Output: true

Explanation: Both trees are structurally identical and have same values.

Example 2:

Given the following two binary trees:

Output: false

Explanation: Trees are structurally different.

---

183. Invert Binary Tree

Problem Statement:

Given the root of a binary tree, invert it.

Examples:

Example 1:

```
Input:
     4
   /   \
  2     7
 / \   / \
1   3 6   9

Output:
     4
   /   \
  7     2
 / \   / \
9   6 3   1
```

Example 2:

```
Input:
     2
   /   \
  3     1
 / \   / \
4   5 6   7

Output:
     2
   /   \
  1     3
 / \   / \
7   6 5   4
```

---

184. Binary Search Tree Iterator

Problem Statement

Implement an iterator for the in-order traversal of a binary search tree (BST). That is, given a BST, we need to implement two functions:

bool hasNext(): Returns true if at least one element is left in the in-order traversal of the BST. **int next()**: Return the next element in the in-order traversal of the BST.

Miscellaneous

185. Kth Smallest Number

Problem Statement

Given an unsorted array of numbers, find Kth smallest number in it.

Please note that it is the Kth smallest number in the sorted order, not the Kth distinct element.

Example 1:

```
Input: [1, 5, 12, 2, 11, 5], K = 3
Output: 5
Explanation: The 3rd smallest number is '5', as the first two smaller
numbers are [1, 2].
```

Example 2:

```
Input: [1, 5, 12, 2, 11, 5], K = 4
Output: 5
Explanation: The 4th smallest number is '5', as the first three smaller
numbers are
[1, 2, 5].
```

Example 3:

```
Input: [5, 12, 11, -1, 12], K = 3
Output: 11
```

---

186. Coding Patterns_ A Cheat Sheet

Coding Patterns: A Cheat Sheet

Here is a brief description of all the coding patterns discussed in this course:

1. Pattern: Two Pointers

Description: This method uses two pointers to traverse an array or a list from different ends or directions.

Usage: It's particularly useful for ordered data structures, where we can make intelligent decisions based on the position of the pointers.

Problems: 'Pair with Target Sum', 'Remove Duplicates', 'Squaring a Sorted Array'.

2. Pattern: Island (Matrix Traversal)

Description: It involves traversing a matrix to find 'islands' or contiguous groups of elements.

Usage: It's generally used in grid-based problems, especially when we need to group connected elements together.

Problems: 'Number of Islands', 'Max Area of Island', 'Flood Fill'.

3. Pattern: Fast & Slow Pointers

Description: In this method, two pointers move at different speeds in a data structure.

Usage: It is commonly used to detect cycles in a structure, find middle elements, or to solve other specific problems related to linked lists.

Problems: 'LinkedList Cycle', 'Middle of the LinkedList', 'Palindrome LinkedList'.

4. Pattern: Sliding Window

Description: This pattern involves creating a 'window' into the data structure and then moving that window around to gather specific information.

Usage: Mostly used in array or list-based problems where you need to find a contiguous subset that fulfills certain conditions.

Problems: 'Maximum Sum Subarray of Size K', 'Smallest Subarray with a given sum', 'Longest Substring with K Distinct Characters'.

5. Pattern: Merge Intervals

Description: This pattern involves merging overlapping intervals.

Usage: Often used in problems involving time intervals, ranges, or sequences.

Problems: 'Merge Intervals', 'Insert Interval', 'Intervals Intersection'.

6. Pattern: Cyclic Sort

Description: This pattern involves sorting an array containing numbers in a given range.

Usage: It's useful in situations where the data involves a finite range of natural numbers.

Problems: 'Cyclic Sort', 'Find the Missing Number', 'Find all Duplicates'.

7. Pattern: In-place Reversal of a Linked List

Description: This pattern involves reversing elements of a linked list in-place.

Usage: It's generally used when reversing a sequence without using extra space.

Problems: 'Reverse a LinkedList', 'Reverse a Sub-list', 'Reverse Every K-element Sub-list'.

8. Pattern: Tree Breadth First Search

Description: This pattern involves level-by-level traversal of a tree.

Usage: It's used when we need to traverse a tree or graph in a level-by-level (breadth-first) manner.

Problems: 'Level Order Traversal', 'Reverse Level Order Traversal', 'Zigzag Traversal'.

9. Pattern: Tree Depth First Search

Description: This pattern involves traversing a tree or graph depth-wise before visiting siblings or neighbors.

Usage: It's used when you need to search deeper into a tree/graph first before going across.

Problems: 'Binary Tree Path Sum', 'All Paths for a Sum', 'Count Paths for a Sum'.

10. Pattern: Two Heaps

Description: This pattern involves using two heaps to divide a set of numbers into two parts.

Usage: It's useful when you need to find median numbers in a sequence, or other similar problems.

Problems: 'Find the Median of a Number Stream', 'Sliding Window Median', 'Maximize Capital'.

11. Pattern: Subsets

Description: This pattern involves generating all subsets of a set.

Usage: It's helpful for solving problems that require exploring all subsets of a given set.

Problems: 'Subsets', 'Subsets With Duplicates', 'Permutations'.

12. Pattern: Modified Binary Search

Description: This is a tweaked version of the binary search algorithm.

Usage: It's used when a simple binary search isn't sufficient, like finding a number in a bitonic array.

Problems: 'Order-agnostic Binary Search', 'Ceiling of a Number', 'Next Letter'.

13. Pattern: Top 'K' Elements

Description: This pattern is used to find the top 'k' elements among a certain category.

Usage: It's commonly used in problems involving sorting, searching, and in heap data structures.

Problems: 'Top K Frequent Numbers', 'Kth Largest Number in a Stream', 'Top K Frequent Elements'.

14. Pattern: Bitwise XOR

Description: This pattern involves the use of Bitwise XOR to solve various array-based problems.

Usage: It's used when we need to manipulate and compare bits directly.

Problems: 'Single Number', 'Two Single Numbers', 'Complement of Base 10 Number'.

15. Pattern: Backtracking

Description: This pattern involves exploring all possible solutions and then backtracking to correct the course whenever you're on the wrong path.

Usage: It's typically used for solving complex combinatorial problems, puzzles, and games.

Problems: 'Sudoku Solver', 'N-Queens', 'Generate Parentheses'.

16. Pattern: 0/1 Knapsack (Dynamic Programming)

Description: This pattern deals with problems where items have different values and weights, and we need to determine the maximum value we can carry.

Usage: It's typically used in optimization problems, especially those involving physical constraints.

Problems: '0/1 Knapsack', 'Equal Subset Sum Partition', 'Subset Sum'.

17. Pattern: Topological Sort (Graph)

Description: This pattern involves sorting nodes in a directed graph in a specific order where the preceding node comes before the following node.

Usage: It's used for scheduling problems and in scenarios where order needs to be imposed on how you process nodes.

Problems: 'Task Scheduling Order', 'All Tasks Scheduling Orders', 'Alien Dictionary'.

18. Pattern: K-way Merge

Description: This pattern involves merging 'k' sorted lists.

Usage: It's typically used in problems involving lists, where merging is required.

Problems: 'Merge K Sorted Lists', 'Kth Smallest Number in M Sorted Lists', 'Smallest Number Range'.

19. Pattern: Monotonic Stack

Description: This pattern involves using a stack to maintain a monotonic (either entirely non-increasing or non-decreasing) order of elements.

Usage: It's often used for solving problems where you need to find the next greater or smaller elements.

Problems: 'Next Greater Element', 'Next Smaller Element', 'Largest Rectangle in Histogram'.

20. Pattern: Multi-threaded

Description: This pattern involves designing algorithms that can execute multiple threads in parallel.

Usage: It's used in situations where a task can be divided into independent sub-tasks that can execute concurrently.

Problems: 'Invert Binary Tree', 'Binary Search Tree Iterator', 'Same Tree'.

21. Pattern: Union Find

Description: Union Find, also known as Disjoint Set Union (DSU), is a data structure that keeps track of a partition of a set into disjoint subsets.

Usage: This pattern is particularly useful for problems where we need to find whether 2 elements belong to the same group or need to solve connectivity-related problems in a graph or tree.

Problems: 'Graph Redundant Connection', 'Number of Provinces', 'Is Graph Bipartite'.
