20 HARD QUESTIONS:

Median of Two Sorted Arrays

Given two sorted arrays nums1 and nums2 of size m and n respectively, return the median of the two sorted arrays.

The overall run time complexity should be O(log (m+n)).

Example 1:

Input: nums1 = [1,3], nums2 = [2]
Output: 2.00000
Explanation: merged array = [1,2,3] and median is 2.
Example 2:

Input: nums1 = [1,2], nums2 = [3,4]
Output: 2.50000
Explanation: merged array = [1,2,3,4] and median is (2 + 3) / 2 = 2.5.

Regular Expression Matching

Given an input string s and a pattern p, implement regular expression matching with support for '.' and '*' where:

'.' Matches any single character.
'*' Matches zero or more of the preceding element.
The matching should cover the entire input string (not partial).

Example 1:

Input: s = "aa", p = "a"
Output: false
Explanation: "a" does not match the entire string "aa".
Example 2:

Input: s = "aa", p = "a*"
Output: true
Explanation: '*' means zero or more of the preceding element, 'a'. Therefore, by repeating 'a' once, it becomes "aa".
Example 3:

Input: s = "ab", p = ".*"
Output: true
Explanation: ".*" means "zero or more (*) of any character (.)".

Merge k Sorted Lists

You are given an array of k linked-lists lists, each linked-list is sorted in ascending order.

Merge all the linked-lists into one sorted linked-list and return it.

Example 1:

```
Input: lists = [[1,4,5],[1,3,4],[2,6]]
Output: [1,1,2,3,4,4,5,6]
Explanation: The linked-lists are:
[
  1->4->5,
  1->3->4,
  2->6
]
merging them into one sorted list:
1->1->2->3->4->4->5->6
```

Example 2:

```
Input: lists = []
Output: []
```

Example 3:

```
Input: lists = [[]]
Output: []
```

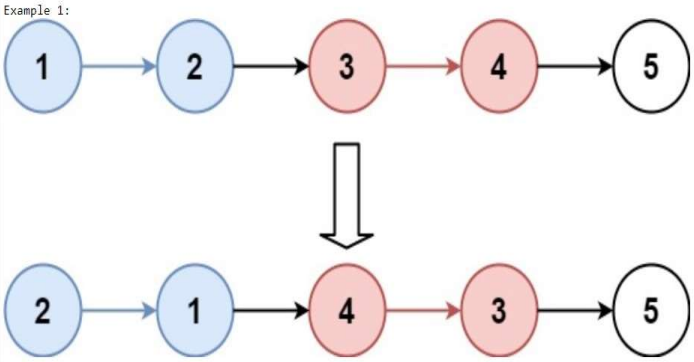
Reverse Nodes in k-Group

Given the head of a linked list, reverse the nodes of the list k at a time, and return the modified list.

k is a positive integer and is less than or equal to the length of the linked list. If the number of nodes is not a multiple of k then left-out nodes, in the end, should remain as it is.

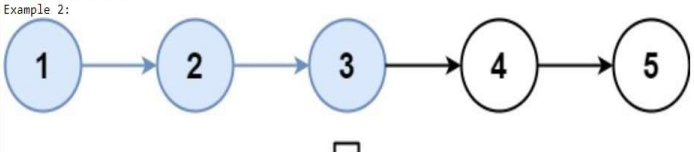
You may not alter the values in the list's nodes, only nodes themselves may be changed.

Example 1:



Input: head = [1,2,3,4,5], k = 2
Output: [2,1,4,3,5]

Example 2:



Substring with Concatenation of All Words

You are given a string s and an array of strings words. All the strings of words are of the same length.

A concatenated substring in s is a substring that contains all the strings of any permutation of words concatenated.

For example, if words = ["ab","cd","ef"], then "abcdef", "abefcd", "cdabef", "cdefab", "efabcd", and "efcdab" are all concatenated strings. "acdbef" is not a concatena
Return the starting indices of all the concatenated substrings in s. You can return the answer in any order.

Example 1:

Input: s = "barfoothefoobarman", words = ["foo","bar"]
Output: [0,9]
Explanation: Since words.length == 2 and words[i].length == 3, the concatenated substring has to be of length 6.
The substring starting at 0 is "barfoo". It is the concatenation of ["bar","foo"] which is a permutation of words.
The substring starting at 9 is "foobar". It is the concatenation of ["foo","bar"] which is a permutation of words.
The output order does not matter. Returning [9,0] is fine too.
Example 2:

Input: s = "wordgoodgoodgoodbestword", words = ["word","good","best","word"]
Output: []
Explanation: Since words.length == 4 and words[i].length == 4, the concatenated substring has to be of length 16.
There is no substring of length 16 is s that is equal to the concatenation of any permutation of words.
We return an empty array.
Example 3:

Input: s = "barfoofoobarthefoobarman", words = ["bar","foo","the"]
Output: [6,9,12]
Explanation: Since words.length == 3 and words[i].length == 3, the concatenated substring has to be of length 9.
The substring starting at 6 is "foobarthe". It is the concatenation of ["foo","bar","the"] which is a permutation of words.
The substring starting at 9 is "barthefoo". It is the concatenation of ["bar","the","foo"] which is a permutation of words.
The substring starting at 12 is "thefoobar". It is the concatenation of ["the","foo","bar"] which is a permutation of words.

Longest Valid Parentheses

Given a string containing just the characters '(' and ')', return the length of the longest valid (well-formed) parentheses substring

.

Example 1:

Input: s = "(()"
Output: 2
Explanation: The longest valid parentheses substring is "()".
Example 2:

Input: s = ")()())"
Output: 4
Explanation: The longest valid parentheses substring is "()()".
Example 3:

Input: s = ""
Output: 0

Sudoku Solver

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

A sudoku solution must satisfy all of the following rules:

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.

Example 1:

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Input: board = [["5","3",".",".","7",".",".",".","."],["6",".",".","1","9","5",".",".","."],[".","9","8",".",".",".",".","6","."],["8",".",".",".","6",".",".",".","3"],["4",".",".","8",".","3",".",".","1"],["7
Output: [["5","3","4","6","7","8","9","1","2"],["6","7","2","1","9","5","3","4","8"],["1","9","8","3","4","2","5","6","7"],["8","5","9","7","6","1","4","2","3"],["4","2","6","8","5","3","7","9","1"],["7","1","
Explanation: The input board is shown above and the only valid solution is shown below:

5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7

First Missing Positive

Given an unsorted integer array nums, return the smallest missing positive integer.

You must implement an algorithm that runs in O(n) time and uses constant extra space.

Example 1:

```
Input: nums = [1,2,0]
Output: 3
Explanation: The numbers in the range [1,2] are all in the array.
```

Example 2:

```
Input: nums = [3,4,-1,1]
Output: 2
Explanation: 1 is in the array but 2 is missing.
```

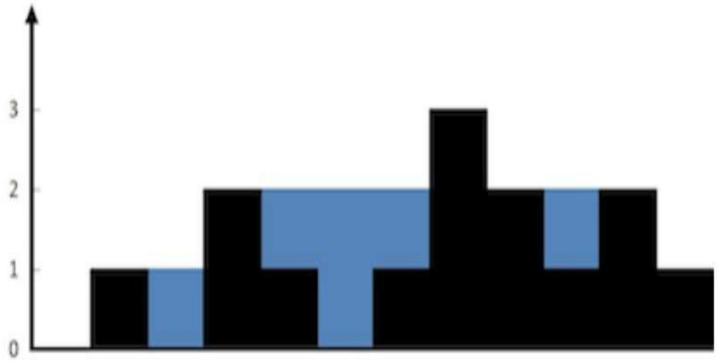
Example 3:

```
Input: nums = [7,8,9,11,12]
Output: 1
Explanation: The smallest positive integer 1 is missing.
```

Trapping Rain Water

Given n non-negative integers representing an elevation map where the width of each bar is 1, compute how much water it can trap after raining.

Example 1:



Input: height = [0,1,0,2,1,0,1,3,2,1,2,1]
Output: 6
Explanation: The above elevation map (black section) is represented by array [0,1,0,2,1,0,1,3,2,1,2,1]. In this case, 6 units of rain water (blue section) are being trapped.
Example 2:

Input: height = [4,2,0,3,2,5]
Output: 9

Wildcard Matching

Given an input string (s) and a pattern (p), implement wildcard pattern matching with support for '?' and '*' where:

'?' Matches any single character.
'*' Matches any sequence of characters (including the empty sequence).
The matching should cover the entire input string (not partial).

Example 1:

Input: s = "aa", p = "a"
Output: false
Explanation: "a" does not match the entire string "aa".
Example 2:

Input: s = "aa", p = "*"
Output: true
Explanation: '*' matches any sequence.
Example 3:

Input: s = "cb", p = "?a"
Output: false
Explanation: '?' matches 'c', but the second letter is 'a', which does not match 'b'.

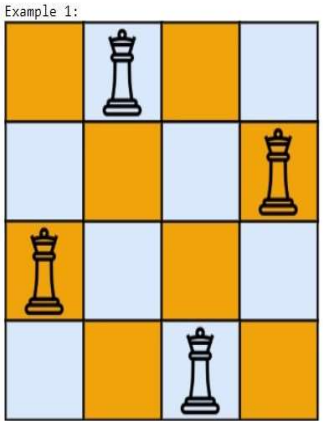
N-Queens

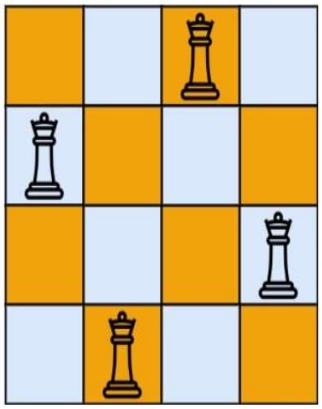
The n-queens puzzle is the problem of placing n queens on an n x n chessboard such that no two queens attack each other.

Given an integer n, return all distinct solutions to the n-queens puzzle. You may return the answer in any order.

Each solution contains a distinct board configuration of the n-queens' placement, where 'Q' and '.' both indicate a queen and an empty space, respectively.

Example 1:





Input: n = 4
Output: [[".Q..","...Q","Q...","..Q."],["..Q.","Q...","...Q",".Q.."]]
Explanation: There exist two distinct solutions to the 4-queens puzzle as shown above
Example 2:

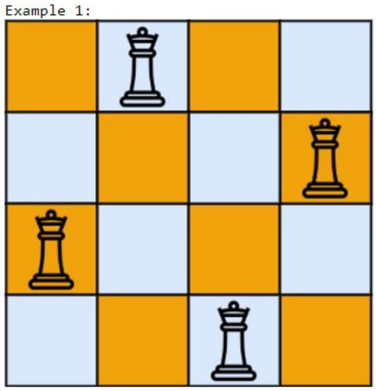
Input: n = 1
Output: [["Q"]]

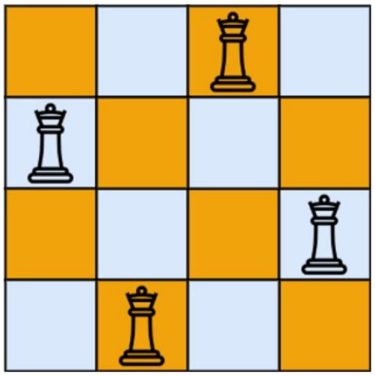
N-Queens II

The n-queens puzzle is the problem of placing n queens on an n x n chessboard such that no two queens attack each other.

Given an integer n, return the number of distinct solutions to the n-queens puzzle.

Example 1:





Input: n = 4
Output: 2
Explanation: There are two distinct solutions to the 4-queens puzzle as shown.
Example 2:

Input: n = 1
Output: 1

Permutation Sequence

The set [1, 2, 3, ..., n] contains a total of n! unique permutations.

By listing and labeling all of the permutations in order, we get the following sequence for n = 3:

"123"
"132"
"213"
"231"
"312"
"321"
Given n and k, return the kth permutation sequence.

Example 1:

Input: n = 3, k = 3
Output: "213"
Example 2:

Input: n = 4, k = 9
Output: "2314"
Example 3:

Input: n = 3, k = 1
Output: "123"

Valid Number

A valid number can be split up into these components (in order):

A decimal number or an integer.
(Optional) An 'e' or 'E', followed by an integer.
A decimal number can be split up into these components (in order):

(Optional) A sign character (either '+' or '-').
One of the following formats:
One or more digits, followed by a dot '.'.
One or more digits, followed by a dot '.', followed by one or more digits.
A dot '.', followed by one or more digits.
An integer can be split up into these components (in order):

(Optional) A sign character (either '+' or '-').
One or more digits.
For example, all the following are valid numbers: ["2", "0089", "-0.1", "+3.14", "4.", "-.9", "2e10", "-90E3", "3e+7", "+6e-1", "53.5e93", "-123.456e789"], while the following are not valid numbers: ["abc", "1a

Given a string s, return true if s is a valid number.

Example 1:

Input: s = "0"
Output: true
Example 2:

Input: s = "e"
Output: false
Example 3:

Input: s = "."
Output: false

Text Justification

Given an array of strings words and a width maxWidth, format the text such that each line has exactly maxWidth characters and is fully (left and right) justified.

You should pack your words in a greedy approach; that is, pack as many words as you can in each line. Pad extra spaces ' ' when necessary so that each line has exactly maxWidth characters.

Extra spaces between words should be distributed as evenly as possible. If the number of spaces on a line does not divide evenly between words, the empty slots on the left will be assigned more spaces than the

For the last line of text, it should be left-justified, and no extra space is inserted between words.

Note:

A word is defined as a character sequence consisting of non-space characters only.
Each word's length is guaranteed to be greater than 0 and not exceed maxWidth.
The input array words contains at least one word.

Example 1:

```
Input: words = ["This", "is", "an", "example", "of", "text", "justification."], maxWidth = 16
Output:
[
   "This    is    an",
   "example  of text",
   "justification.  "
]
```

Example 2:

```
Input: words = ["What","must","be","acknowledgment","shall","be"], maxWidth = 16
Output:
[
  "What   must   be",
  "acknowledgment  ",
  "shall be        "
]
Explanation: Note that the last line is "shall be    " instead of "shall     be", because the last line must be left-justified instead of fully-justified.
Note that the second line is also left-justified because it contains only one word.
```

Example 3:

```
Input: words = ["Science","is","what","we","understand","well","enough","to","explain","to","a","computer.","Art","is","everything","else","we","do"], maxWidth = 20
Output:
[
  "Science  is  what we",
  "understand      well",
  "enough to explain to"
```

Edit Distance

Given two strings word1 and word2, return the minimum number of operations required to convert word1 to word2.

You have the following three operations permitted on a word:

Insert a character
Delete a character
Replace a character

Example 1:

```
Input: word1 = "horse", word2 = "ros"
Output: 3
Explanation:
horse -> rorse (replace 'h' with 'r')
rorse -> rose (remove 'r')
rose -> ros (remove 'e')
```

Example 2:

```
Input: word1 = "intention", word2 = "execution"
Output: 5
Explanation:
intention -> inention (remove 't')
inention -> enention (replace 'i' with 'e')
enention -> exention (replace 'n' with 'x')
exention -> exection (replace 'n' with 'c')
exection -> execution (insert 'u')
```

Minimum Window Substring

Given two strings s and t of lengths m and n respectively, return the minimum window
substring
 of s such that every character in t (including duplicates) is included in the window. If there is no such substring, return the empty string "".

The testcases will be generated such that the answer is unique.

Example 1:

Input: s = "ADOBECODEBANC", t = "ABC"
Output: "BANC"
Explanation: The minimum window substring "BANC" includes 'A', 'B', and 'C' from string t.
Example 2:

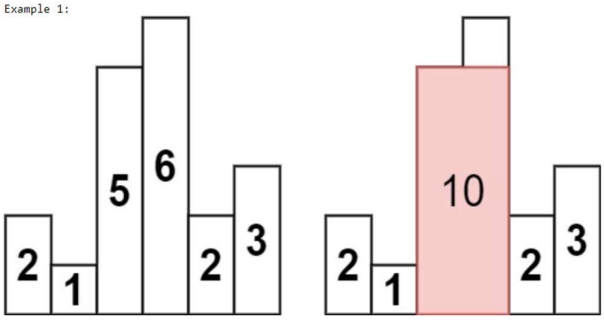
Input: s = "a", t = "a"
Output: "a"
Explanation: The entire string s is the minimum window.
Example 3:

Input: s = "a", t = "aa"
Output: ""
Explanation: Both 'a's from t must be included in the window.
Since the largest window of s only has one 'a', return empty string.

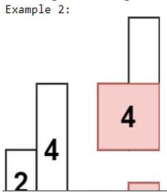
Largest Rectangle in Histogram

Given an array of integers heights representing the histogram's bar height where the width of each bar is 1, return the area of the largest rectangle in the histogram.

Example 1:



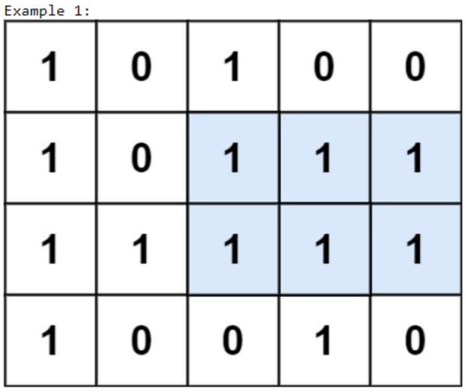
Input: heights = [2,1,5,6,2,3]
Output: 10
Explanation: The above is a histogram where width of each bar is 1.
The largest rectangle is shown in the red area, which has an area = 10 units.
Example 2:



Maximal Rectangle

Given a rows x cols binary matrix filled with 0's and 1's, find the largest rectangle containing only 1's and return its area.

Example 1:




```
Input: matrix = [["1","0","1","0","0"],["1","0","1","1","1"],["1","1","1","1","1"],["1","0","0","1","0"]]
Output: 6
Explanation: The maximal rectangle is shown in the above picture.
Example 2:

Input: matrix = [["0"]]
Output: 0
Example 3:

Input: matrix = [["1"]]
Output: 1
```

Scramble String

```
We can scramble a string s to get a string t using the following algorithm:

If the length of the string is 1, stop.
If the length of the string is > 1, do the following:
Split the string into two non-empty substrings at a random index, i.e., if the string is s, divide it to x and y where s = x + y.
Randomly decide to swap the two substrings or to keep them in the same order. i.e., after this step, s may become s = x + y or s = y + x.
Apply step 1 recursively on each of the two substrings x and y.
Given two strings s1 and s2 of the same length, return true if s2 is a scrambled string of s1, otherwise, return false.



Example 1:

Input: s1 = "great", s2 = "rgeat"
Output: true
Explanation: One possible scenario applied on s1 is:
"great" --> "gr/eat" // divide at random index.
"gr/eat" --> "gr/eat" // random decision is not to swap the two substrings and keep them in order.
"gr/eat" --> "g/r / e/at" // apply the same algorithm recursively on both substrings. divide at random index each of them.
"g/r / e/at" --> "r/g / e/at" // random decision was to swap the first substring and to keep the second substring in the same order.
"r/g / e/at" --> "r/g / e/ a/t" // again apply the algorithm recursively, divide "at" to "a/t".
"r/g / e/ a/t" --> "r/g / e/ a/t" // random decision is to keep both substrings in the same order.
The algorithm stops now, and the result string is "rgeat" which is s2.
As one possible scenario led s1 to be scrambled to s2, we return true.
Example 2:

Input: s1 = "abcde", s2 = "caebd"
Output: false
Example 3:

Input: s1 = "a", s2 = "a"
Output: true
```