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

A sudoku solution must satisfy **all of the following rules**:

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

The '.' character indicates empty cells

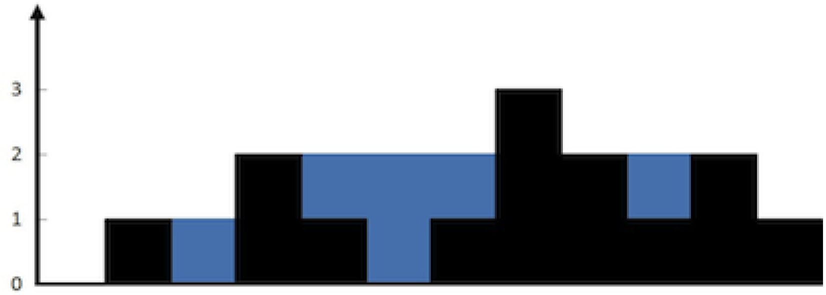
5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Input: board =
[["5","3",".",".","7",".",".",".","."],["6",".",".","1","9","5",".",".","."],[".","9","8",".",".",".",".","6","."],["8",".",".",".","6",".",".",".","3"],["4",".",".","8",".","3",".",".","1"],["7",".",".",".","2",".",".",".","6"],[".","6",".",".",".",".","2","8","."],[".",".",".","4","1","9",".",".","5"],[".",".",".",".","8",".",".","7","9"]]
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","3","9","2","4","8","5","6"],["9","6","1","5","3","7","2","8","4"],["2","8","7","4","1","9","6","3","5"],["3","4","5","2","8","6","1","7","9"]]
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
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

2) 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

3) Implement [pow(x, n)](), which calculates x raised to the power n (i.e., x^n).

Example 1:

Input: x = 2.00000, n = 10
Output: 1024.00000

Example 2:

Input: x = 2.10000, n = 3
Output: 9.26100

Example 3:

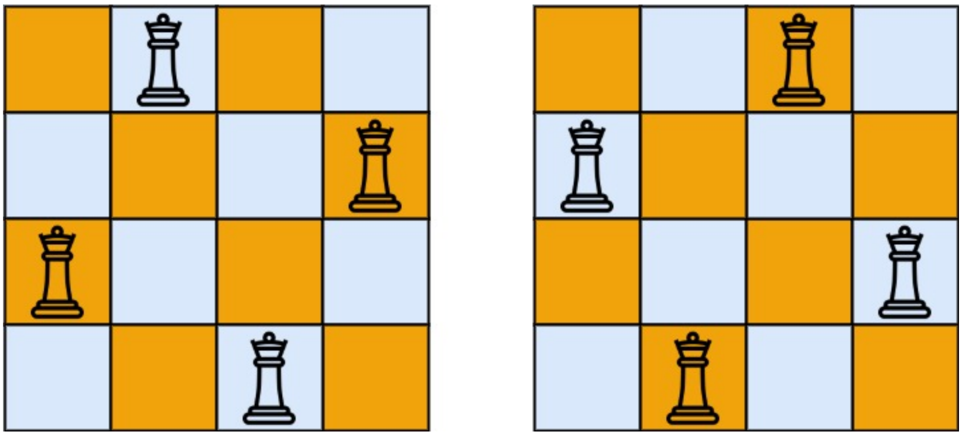
Input: x = 2.00000, n = -2
Output: 0.25000
Explanation: $2^{-2} = 1/2^2 = 1/4 = 0.25$

4) 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"]]

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

1. If the length of the string is 1, stop.
2. 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

6) Given two strings s and t, return *the number of distinct subsequences of s which equals t*.

A string's **subsequence** is a new string formed from the original string by deleting some (can be none) of the characters without disturbing the remaining characters' relative positions. (i.e., "ACE" is a subsequence of "ABCDE" while "AEC" is not).

The test cases are generated so that the answer fits on a 32-bit signed integer.

Example 1:

Input: s = "rabbbit", t = "rabbit"
Output: 3
Explanation:
As shown below, there are 3 ways you can generate "rabbit" from S.
rabbb**it**
rab**bbit**
rabb**bit**

Example 2:

Input: s = "babgbag", t = "bag"
Output: 5
Explanation:
As shown below, there are 5 ways you can generate "bag" from S.
bab**g**bag
babgba**g**
babgb**ag**
ba**b**gb**ag**
babg**bag**

7) A **transformation sequence** from word beginWord to word endWord using a dictionary wordList is a sequence of words beginWord -> s_1 -> s_2 -> ... -> s_k such that:

- Every adjacent pair of words differs by a single letter.
- Every s_i for 1 <= i <= k is in wordList. Note that beginWord does not need to be in wordList.
- s_k == endWord

Given two words, beginWord and endWord, and a dictionary wordList, return *the **number of words** in the **shortest transformation sequence** from*beginWord*to*endWord*, or* 0 *if no such sequence exists.*

Example 1: Input: beginWord = "hit", endWord = "cog", wordList = ["hot","dot","dog","lot","log","cog"]
Output: 5
Explanation: One shortest transformation sequence is "hit" -> "hot" -> "dot" -> "dog" -> cog", which is 5 words long.

Example 2:

Input: beginWord = "hit", endWord = "cog", wordList = ["hot","dot","dog","lot","log"]
Output: 0
Explanation: The endWord "cog" is not in wordList, therefore there is no valid transformation sequence.

8) You are a professional robber planning to rob houses along a street. Each house has a certain amount of money stashed, the only constraint stopping you from robbing each of them is that adjacent houses have security systems connected and **it will automatically contact the police if two adjacent houses were broken into on the same night**.

Given an integer array nums representing the amount of money of each house, return *the maximum amount of money you can rob tonight **without alerting the police***.

Example 1:

Input: nums = [1,2,3,1]
Output: 4
Explanation: Rob house 1 (money = 1) and then rob house 3 (money = 3).
Total amount you can rob = 1 + 3 = 4.

Example 2:

Input: nums = [2,7,9,3,1]
Output: 12
Explanation: Rob house 1 (money = 2), rob house 3 (money = 9) and rob house 5 (money = 1).
Total amount you can rob = 2 + 9 + 1 = 12.

9) You are given an array **A** of size **N**. You need to first push the elements of the array into a stack and then print minimum in the stack at each pop.

Example 1: Input:
N = 5
A = {1 2 3 4 5}
Output:
1 1 1 1 1
Explanation:
After pushing elements to the stack,
the stack will be "top -> 5, 4, 3, 2, 1".
Now, start popping elements from the stack
popping 5: min in the stack is 1.popped 5
popping 4: min in the stack is 1. popped 4
popping 3: min in the stack is 1. popped 3
popping 2: min in the stack is 1. popped 2
popping 1: min in the stack is 1. popped 1

Example 2:

Input:
N = 7
A = {1 6 43 1 2 0 5}
Output:
0 0 1 1 1 1 1

10) Given a BST and a number **X**, find **Ceil of X**.
Note: Ceil(X) is a number that is either equal to X or is immediately greater than X.

Example 1:

Input:
```
     5
   /   \
  1     7
   \
    2
     \
      3
```
X = 3
Output: 3
Explanation: We find 3 in BST, so ceil
of 3 is 3.

Example 2:

Input:
```
    10
   /  \
  5    11
 / \
4   7
     \
      8
```
X = 6
Output: 7
Explanation: We find 7 in BST, so ceil
of 6 is 7.

11) You are given a string s. You can convert s to a palindrome by adding characters in front of it.

Return *the shortest palindrome you can find by performing this transformation*.

Example 1:

Input: s = "aacecaaa"
Output: "aaacecaaa"

Example 2:

Input: s = "abcd"
Output: "dcbabcd"

12) You are given an array of people, people, which are the attributes of some people in a queue (not necessarily in order). Each people[i] = [h_i, k_i] represents the i^{th} person of height h_i with **exactly** k_i other people in front who have a height greater than or equal to h_i.

Reconstruct and return *the queue that is represented by the input array* people. The returned queue should be formatted as an array queue, where queue[j] = [h_j, k_j] is the attributes of the j^{th} person in the queue (queue[0] is the person at the front of the queue).

Example 1:

Input: people = [[7,0],[4,4],[7,1],[5,0],[6,1],[5,2]]
Output: [[5,0],[7,0],[5,2],[6,1],[4,4],[7,1]]
Explanation:
Person 0 has height 5 with no other people taller or the same height in front.
Person 1 has height 7 with no other people taller or the same height in front.
Person 2 has height 5 with two persons taller or the same height in front, which is person 0 and 1.
Person 3 has height 6 with one person taller or the same height in front, which is person 1.
Person 4 has height 4 with four people taller or the same height in front, which are people 0, 1, 2, and 3.
Person 5 has height 7 with one person taller or the same height in front, which is person 1.
Hence [[5,0],[7,0],[5,2],[6,1],[4,4],[7,1]] is the reconstructed queue.

Example 2:

Input: people = [[6,0],[5,0],[4,0],[3,2],[2,2],[1,4]]
Output: [[4,0],[5,0],[2,2],[3,2],[1,4],[6,0]]

13) Given an integer array nums, return *the number of all the **arithmetic subsequences** of*nums.

A sequence of numbers is called arithmetic if it consists of **at least three elements** and if the difference between any two consecutive elements is the same.

- For example, [1, 3, 5, 7, 9], [7, 7, 7, 7], and [3, -1, -5, -9] are arithmetic sequences.
- For example, [1, 1, 2, 5, 7] is not an arithmetic sequence.

A **subsequence** of an array is a sequence that can be formed by removing some elements (possibly none) of the array.

- For example, [2,5,10] is a subsequence of [1,2,1,**2**,4,1,**5**,**10**].

The test cases are generated so that the answer fits in **32-bit** integer.

Example 1:

Input: nums = [2,4,6,8,10]
Output: 7

Explanation: All arithmetic subsequence slices are:
[2,4,6]
[4,6,8]
[6,8,10]
[2,4,6,8]
[4,6,8,10]
[2,4,6,8,10]
[2,6,10]

Example 2:

Input: nums = [7,7,7,7,7]
Output: 16
Explanation: Any subsequence of this array is arithmetic.

14) Given two string arrays word1 and word2, return true if the two arrays represent the same string, and false otherwise.

A string is represented by an array if the array elements concatenated in order forms the string.

Example 1:

Input: word1 = ["ab", "c"], word2 = ["a", "bc"]

Output: true

Explanation:

word1 represents string "ab" + "c" -> "abc"

word2 represents string "a" + "bc" -> "abc"

The strings are the same, so return true.

Example 2:

Input: word1 = ["a", "cb"], word2 = ["ab", "c"]

Output: false

Example 3:

Input: word1 = ["abc", "d", "defg"], word2 = ["abcddefg"]

Output: true

15) Given a string s containing just the characters '(', ')', '{', '}', '[' and ']', determine if the input string is valid.

An input string is valid if:

Open brackets must be closed by the same type of brackets.

Open brackets must be closed in the correct order.

Example 1:

Input: s = "()"

Output: true

Example 2:

Input: s = "()[]{}"

Output: true

Example 3:

Input: s = "(]"

Output: false