1. **Sort List**

Given the head of a linked list, return the list after sorting it in **ascending order**.

**Example 1:**

**Input:** head = [4,2,1,3]

**Output:** [1,2,3,4]

**Example 2:**



**Input:** head = [-1,5,3,4,0]**Output:** [-1,0,3,4,5]

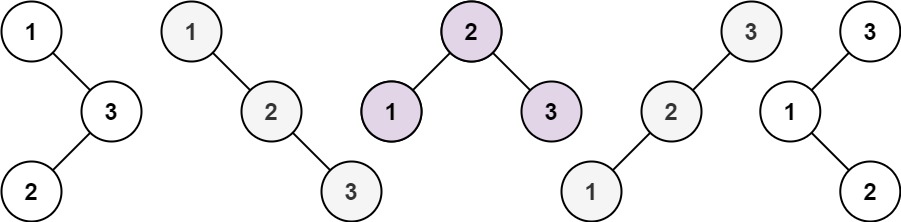
**Example 3:**

**Input:** head = []**Output:** []

**Follow up:** Can you sort the linked list in O(n logn) time and O(1) memory (i.e. constant space)?

**2. Unique Binary Search Trees**

Given an integer n, return the number of structurally unique **BST'**s (binary search trees) which has exactly n nodes of unique values from 1 to n.

**Example 1:**

**Input:** n = 3**Output:** 5

**Example 2:**

**Input:** n = 1**Output:** 1

**3. Valid Parentheses**

Given a string s containing just the characters

'(' , ')' , '{' , '}' , '[' and ']'

determine if the input string is valid.

An input string is valid if:

1. Open brackets must be closed by the same type of brackets.
2. Open brackets must be closed in the correct order.

**Examples:**

**Input:** s = "()"**Output:** true

**Input:** s = "()[]{}"**Output:** true

**Input:** s = "(]"**Output:** false

**Input:** s = "([)]"**Output:** false

**Input:** s = "{[]}"**Output:** true

**4.Convert Sorted List to Binary Search Tree**

Given the head of a singly linked list where elements are **sorted in ascending order**, convert it to a height balanced BST.

For this problem, a height-balanced binary tree is defined as a binary tree in which the depth of the two subtrees of every node never differ by more than 1.

**Example 1:**

**Input:** head = [-10,-3,0,5,9]

**Output:** [0,-3,9,-10,null,5]

**Explanation:** One possible answer is [0,-3,9,-10,null,5], which represents the shown height balanced BST.

**Example 2:**

**Input:** head = [1,3]**Output:** [3,1]

**5. Implement Queue using Stacks**

Implement a first in first out (FIFO) queue using only two stacks. The implemented queue should support all the functions of a normal queue (push, peek, pop, and empty).

Implement the MyQueue class:

* void push(int x) Pushes element x to the back of the queue.
* int pop() Removes the element from the front of the queue and returns it.
* int peek() Returns the element at the front of the queue.
* boolean empty() Returns true if the queue is empty, false otherwise.

**Notes:**

* You must use **only** standard operations of a stack, which means only push to top, peek/pop from top, size, and is empty operations are valid.
* Depending on your language, the stack may not be supported natively. You may simulate a stack using a list or deque (double-ended queue) as long as you use only a stack's standard operations.

**Example 1:**

**Input**

["MyQueue", "push", "push", "peek", "pop", "empty"]

[[], [1], [2], [], [], []]

**Output**

[null, null, null, 1, 1, false]

**Explanation**

MyQueue myQueue = new MyQueue();

myQueue.push(1); // queue is: [1]

myQueue.push(2); // queue is: [1, 2] (leftmost is front of the queue)

myQueue.peek(); // return 1

myQueue.pop(); // return 1, queue is [2]

myQueue.empty(); // return false

**Follow-up:** Can you implement the queue such that each operation is [amortized](https://en.wikipedia.org/wiki/Amortized_analysis" \t "/home/levi/Documents\\x/_blank) O(1) time complexity? In other words, performing n operations will take overall O(n) time even if one of those operations may take longer.

**6. Rotate Image**

You are given an n x n 2D matrix representing an image, rotate the image by **90** degrees (clockwise).

You have to rotate the image [in-place](https://en.wikipedia.org/wiki/In-place_algorithm" \t "/home/levi/Documents\\x/_blank), which means you have to modify the input 2D matrix directly. **DO NOT** allocate another 2D matrix and do the rotation.

**Example 1:**

**Input:**

matrix = [[1,2,3],[4,5,6],[7,8,9]]

**Output:**

[[7,4,1],[8,5,2],[9,6,3]]

**Example 2:**

**Input:**

matrix = [[5,1,9,11],[2,4,8,10],[13,3,6,7],[15,14,12,16]]

**Output:**

[[15,13,2,5],[14,3,4,1],[12,6,8,9],[16,7,10,11]]

**Example 3:**

**Input:** matrix = [[1]]**Output:** [[1]]

**Example 4:**

**Input:** matrix = [[1,2],[3,4]]**Output:** [[3,1],[4,2]]

**7.Transform String**

Given two strings A and B. Find the minimum number of steps required to transform string A into string B. The only allowed operation for the transformation is selecting a character from string A and inserting it in the beginning of string A.

**Example 1:**

**Input:**

A = "abd" ,B = "bad"

**Output:** 1

**Explanation:** The conversion can take place in

1 operation: Pick 'b' and place it at the front.

**Example 2:**

**Input:**

A = "helloworld" ,B = "owohellrld"

**Output:** 3

**Explanation:** The conversion can take

place in 3 operations:

Pick 'o' and place it at the front.

A = "ohellowrld"

Pick 'w' and place it at the front.

A = "wohellorld"

Pick 'o' and place it at the front.

A = "owohellrld"

**Your Task:**   
You dont need to read input or print anything. Complete the function **transform()** which takes two strings A and B as input parameters and returns the minimum number of steps required to transform A into B. If transformation is not possible return -1.

**Expected Time Complexity:** O(N) where N is max(length of A, length of B)   
**Expected Auxiliary Space:** O(1)

**8. Longest Consecutive Sequence**

Given an unsorted array of integers nums, return the length of the longest consecutive elements sequence.

**Example 1:**

**Input:** nums = [100,4,200,1,3,2]

**Output:** 4

**Explanation:** The longest consecutive elements sequence is [1, 2, 3, 4]. Therefore its length is 4.

**Example 2:**

**Input:** nums = [0,3,7,2,5,8,4,6,0,1]

**Output:** 9

**Expected Time Complexity:** You must write an algorithm that runs in O(n) time.