Algo Expert Question List

EASY SERIES

Two Number Sum

Write a function that takes in a non-empty array of distinct integers and an integer representing a target sum. If any two numbers in the input array sum up to the target sum, the function should return them in an array, in sorted order. If no two numbers sum up to the target sum, the function should return an empty array. Assume that there will be at most one pair of numbers summing up to the target sum.

Sample input: [3, 5, -4, 8, 11, 1, -1, 6], 10
Sample output: [-1, 11]

EASY SERIES

Find Closest Value In BST

You are given a BST data structure consisting of BST nodes. Each BST node has an integer value stored in a property called "value" and two children nodes stored in properties called "left" and "right," respectively. A node is said to be a BST node if and only if it satisfies the BST property: its value is strictly greater than the values of every node to its left; its value is less than or equal to the values of every node to its right; and both of its children nodes are either BST nodes themselves or None (null) values. You are also given a target integer value; write a function that finds the closest value to that target value contained in the BST. Assume that there will only be one closest value.

EASY SERIES

Depth-first Search

You are given a Node class that has a name and an array of optional children Nodes. When put together, Nodes form a simple tree-like structure. Implement the depthFirstSearch method on the Node class, which takes in an empty array, traverses the tree using the Depth-first Search approach (specifically navigating the tree from left to right), stores all the of the Nodes' names in the input array, and returns it.

EASY SERIES

Linked List Construction

Write a class for a Doubly Linked List. The class should have a "head" and "tail" properties, both of which should point to either the None (null) value or to a Linked List node. Every node will have a "value" property as well as "next" and "prev" properties, both of which can point to either the None (null) value or another node. The class should support setting the head and tail of the linked list, inserting nodes before and after other nodes as well as at certain positions, removing given nodes and removing nodes with specific values, and searching for nodes with values. Only the searching method should return a value: specifically, a boolean.

Sample input:
1 -> 2 -> 3 -> 4 -> 5
Sample output (after setting 4 to head):
4 -> 1 -> 2 -> 3 -> 5
Sample output (after setting 6 to tail):
4 -> 1 -> 2 -> 3 -> 5 -> 6

EASY SERIES

Nth Fibonacci

The Fibonacci sequence is defined as follows: the first number of the sequence is 0, the second number is 1, and the nth number is the sum of the (n - 1)th and (n - 2)th numbers. Write a function that takes in an integer n and returns the nth Fibonacci number.

Sample input: 6
Sample output: 5 (0, 1, 1, 2, 3, 5)

EASY SERIES

Binary Search

Write a function that takes in a sorted array of integers as well as a target integer. The function should use the Binary Search algorithm to find if the target number is contained in the array and should return its index if it is, otherwise -1.

Sample input: [0, 1, 21, 33, 45, 45, 61, 71, 72, 73], 33
Sample output: 3

EASY SERIES

Find Three Largest Numbers

Write a function that takes in an array of integers and returns a sorted array of the three largest integers in the input array. Note that the function should return duplicate integers if necessary; for example, it should return [10, 10, 12] for an input array of [10, 5, 9, 10, 12].

Sample input: [141, 1, 17, -7, -17, -27, 18, 541, 8, 7, 7]
Sample output: [18, 141, 541]

EASY SERIES

Bubble Sort

Write a function that takes in an array of integers and returns a sorted version of that array. Use the Bubble Sort algorithm to sort the array.

Sample input: [8, 5, 2, 9, 5, 6, 3]
Sample output: [2, 3, 5, 5, 6, 8, 9]

EASY SERIES

Insertion Sort

Write a function that takes in an array of integers and returns a sorted version of that array. Use the Insertion Sort algorithm to sort the array.

Sample input: [8, 5, 2, 9, 5, 6, 3]
Sample output: [2, 3, 5, 5, 6, 8, 9]

EASY SERIES

Selection Sort

Write a function that takes in an array of integers and returns a sorted version of that array. Use the Selection Sort algorithm to sort the array.

Sample input: [8, 5, 2, 9, 5, 6, 3]
Sample output: [2, 3, 5, 5, 6, 8, 9]

EASY SERIES

Palindrome Check

Write a function that takes in a non-empty string and that returns a boolean representing whether or not the string is a palindrome. A palindrome is defined as a string that is written the same forward and backward.

Sample input: "abcdcba"
Sample output: True (it is a palindrome)

EASY SERIES

Caesar Cipher Encryptor

Given a non-empty string of lowercase letters and a non-negative integer value representing a key, write a function that returns a new string obtained by shifting every letter in the input string by k positions in the alphabet, where k is the key. Note that letters should "wrap" around the alphabet; in other words, the letter "z" shifted by 1 returns the letter "a".

Sample input: "xyz", 2
Sample output: "zab"

MEDIUM SERIES

Three Number Sum

Write a function that takes in a non-empty array of distinct integers and an integer representing a target sum. The function should find all triplets in the array that sum up to the target sum and return a two-dimensional array of all these triplets. The numbers in each triplet should be ordered in ascending order, and the triplets themselves should be ordered in ascending order with respect to the numbers they hold. If no three numbers sum up to the target sum, the function should return an empty array.

Sample input: [12, 3, 1, 2, -6, 5, -8, 6], 0
Sample output: [[-8, 2, 6], [-8, 3, 5], [-6, 1, 5]]

MEDIUM SERIES

Smallest Difference

Write a function that takes in two non-empty arrays of integers. The function should find the pair of numbers (one from the first array, one from the second array) whose absolute difference is closest to zero. The function should return an array containing these two numbers, with the number from the first array in the first position. Assume that there will only be one pair of numbers with the smallest difference.

Sample input: [-1, 5, 10, 20, 28, 3], [26, 134, 135, 15, 17]
Sample output: [28, 26]

MEDIUM SERIES

BST Construction

Write a Binary Search Tree (BST) class. The class should have a "value" property set to be an integer, as well as "left" and "right" properties, both of which should point to either the None (null) value or to another BST. A node is said to be a BST node if and only if it satisfies the BST property: its value is strictly greater than the values of every node to its left; its value is less than or equal to the values of every node to its right; and both of its children nodes are either BST nodes themselves or None (null) values. The BST class should support insertion, searching, and removal of values. The removal method should only remove the first instance of the target value.

Sample input:

Sample output (after inserting 12):

Sample output (after removing 10):

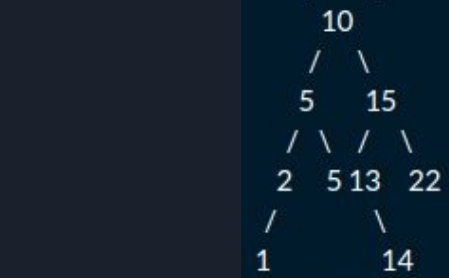
Sample output (after searching for 15): True

MEDIUM SERIES

Validate BST

You are given a Binary Tree data structure consisting of Binary Tree nodes. Each Binary Tree node has an integer value stored in a property called "value" and two children nodes stored in properties called "left" and "right," respectively. Children nodes can either be Binary Tree nodes themselves or the None (null) value. Write a function that returns a boolean representing whether or not the Binary Tree is a valid BST. A node is said to be a BST node if and only if it satisfies the BST property: its value is strictly greater than the values of every node to its left; its value is less than or equal to the values of every node to its right; and both of its children nodes are either BST nodes themselves or None (null) values.

Sample input:



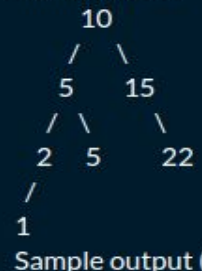
Sample output: True

MEDIUM SERIES

BST Traversal

You are given a BST data structure consisting of BST nodes. Each BST node has an integer value stored in a property called "value" and two children nodes stored in properties called "left" and "right," respectively. A node is said to be a BST node if and only if it satisfies the BST property: its value is strictly greater than the values of every node to its left; its value is less than or equal to the values of every node to its right; and both of its children nodes are either BST nodes themselves or None (null) values. Write three functions that take in an empty array, traverse the BST, add its nodes' values to the input array, and return that array. The three functions should traverse the BST using the in-order traversal, pre-order traversal, and post-order traversal techniques, respectively.

Sample input:



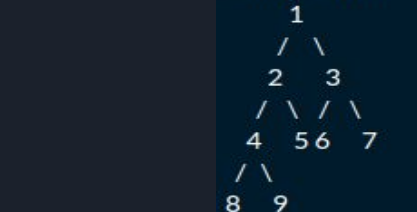
Sample output (inOrderTraverse): [1, 2, 5, 5, 10, 15, 22]
Sample output (preOrderTraverse): [10, 5, 2, 1, 5, 15, 22]
Sample output (postOrderTraverse): [1, 2, 5, 5, 22, 15, 10]

MEDIUM SERIES

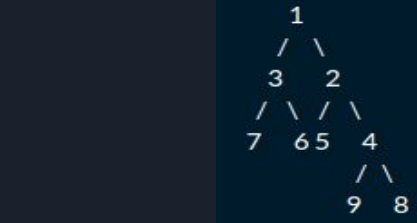
Invert Binary Tree

Write a function that takes in a Binary Tree and inverts it. In other words, the function should swap every left node in the tree for its corresponding (mirrored) right node. Each Binary Tree node has a value stored in a property called "value" and two children nodes stored in properties called "left" and "right," respectively. Children nodes can either be Binary Tree nodes themselves or the None (null) value.

Sample input:



Sample output:



MEDIUM SERIES

Maximum Subset Sum With No Adjacent Elements

Write a function that takes in an array of positive integers and returns an integer representing the maximum sum of non-adjacent elements in the array. If a sum cannot be generated, the function should return 0.

Sample input: [75, 105, 120, 75, 90, 135]
Sample output: 330 (75, 120, 135)

MEDIUM SERIES

Number Of Ways To Make Change

Given an array of positive integers representing coin denominations and a single non-negative integer representing a target amount of money, implement a function that returns the number of ways to make change for that target amount using the given coin denominations. Note that an unlimited amount of coins is at your disposal.

Sample input: 6, [1, 5]
Sample output: 2 (1x1 + 1x5 and 6x1)

MEDIUM SERIES

Min Number Of Coins For Change

Given an array of positive integers representing coin denominations and a single non-negative integer representing a target amount of money, implement a function that returns the smallest number of coins needed to make change for that target amount using the given coin denominations. Note that an unlimited amount of coins is at your disposal. If it is impossible to make change for the target amount, return -1.

Sample input: 7, [1, 5, 10]
Sample output: 3 (2x1 + 1x5)

MEDIUM SERIES

Levenshtein Distance

Write a function that takes in two strings and returns the minimum number of edit operations that need to be performed on the first string to obtain the second string. There are three edit operations: insertion of a character, deletion of a character, and substitution of a character for another.

Sample input: "abc", "yabd"
Sample output: 2 (insert "y"; substitute "c" for "d")

MEDIUM SERIES

Kadane's Algorithm

Write a function that takes in a non-empty array of integers and returns the maximum sum that can be obtained by summing up all the numbers in a non-empty subarray of the input array. A subarray must only contain adjacent numbers.

Sample input: [3, 5, -9, 1, 3, -2, 3, 4, 7, 2, -9, 6, 3, 1, -5, 4]
Sample output: 19 ([1, 3, -2, 3, 4, 7, 2, -9, 6, 3, 1])

MEDIUM SERIES

Single Cycle Check

You are given an array of integers. Each integer represents a jump of its value in the array. For instance, the integer 2 represents a jump of 2 indices forward in the array; the integer -3 represents a jump of 3 indices backward in the array. If a jump spills past the array's bounds, it wraps over to the other side. For instance, a jump of -1 at index 0 brings us to the last index in the array. Similarly, a jump of 1 at the last index in the array brings us to index 0. Write a function that returns a boolean representing whether the jumps in the array form a single cycle. A single cycle occurs if, starting at any index in the array and following the jumps, every element is visited exactly once before landing back on the starting index.

Sample input: [2, 3, 1, -4, -4, 2]
Sample output: True

MEDIUM SERIES

Breadth-first Search

You are given a Node class that has a name and an array of optional children Nodes. When put together, Nodes form a simple tree-like structure. Implement the breadthFirstSearch method on the Node class, which takes in an empty array, traverses the tree using the Breadth-first Search approach (specifically navigating the tree from left to right), stores all of the Nodes' names in the input array, and returns it.

Sample input:

Sample output: ["A", "B", "C", "D", "E", "F", "G", "H", "I", "J", "K"]

MEDIUM SERIES

River Sizes

You are given a two-dimensional array (matrix) of potentially unequal height and width containing only 0s and 1s. Each 0 represents land, and each 1 represents part of a river. A river consists of any number of 1s that are either horizontally or vertically adjacent (but not diagonally adjacent). The number of adjacent 1s forming a river determine its size. Write a function that returns an array of the sizes of all rivers represented in the input matrix. Note that these sizes do not need to be in any particular order.

Sample input:

```
[
  [1, 0, 0, 1, 0],
  [1, 0, 1, 0, 0],
  [0, 0, 1, 0, 1],
  [1, 0, 1, 0, 1],
  [1, 0, 1, 1, 0],
]
```

Sample output: [1, 2, 2, 2, 5]

MEDIUM SERIES

Youngest Common Ancestor

You're given three inputs, all of which are instances of a class that have an "ancestor" property pointing to their youngest ancestor. The first input is the top ancestor in an ancestral tree (i.e., the only instance that has no ancestor), and the other two inputs are descendants in the ancestral tree. Write a function that returns the youngest common ancestor to the two descendants.

Sample input: Node A, Node E, Node I (from the ancestral tree below)

Sample output: Node B

MEDIUM SERIES

Youngest Common Ancestor

You're given three inputs, all of which are instances of a class that have an "ancestor" property pointing to their youngest ancestor. The first input is the top ancestor in an ancestral tree (i.e., the only instance that has no ancestor), and the other two inputs are descendants in the ancestral tree. Write a function that returns the youngest common ancestor to the two descendants.

Sample input: Node A, Node E, Node I (from the ancestral tree below)

Sample output: Node B

MEDIUM SERIES

Min Heap Construction

Implement a Min Heap class. The class should support insertion, removal (of the minimum / root value), peeking (of the minimum / root value), as well as sifting a value up and down the heap and building the heap from an input array. The heap should be represented in the form of an array.

Sample input: [48, 12, 24, 7, 8, -5, 24, 391, 24, 56, 2, 6, 8, 41]
-> buildHeap(array)
-> insert(76)
-> remove()
-> remove()
-> insert(87)
Sample output:
-> [-5, 2, 6, 7, 8, 8, 24, 391, 24, 56, 12, 24, 48, 41]
-> [-5, 2, 6, 7, 8, 8, 24, 391, 24, 56, 12, 24, 48, 41, 76]
-> [2, 7, 6, 24, 8, 8, 24, 391, 76, 56, 12, 24, 48, 41]
-> [6, 7, 8, 24, 8, 24, 24, 391, 76, 56, 12, 41, 48]
-> [6, 7, 8, 24, 8, 24, 24, 391, 76, 56, 12, 41, 48, 87]

MEDIUM SERIES

Remove Kth Node From End

Write a function that takes in the head of a Singly Linked List and an integer k (assume that the list has at least k nodes). The function should remove the kth node from the end of the list. Note that every node in the Singly Linked List has a "value" property storing its value as well as a "next" property pointing to the next node in the list or None (null) if it is the tail of the list.

Sample input: 0 -> 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> 8 -> 9, 4
Sample output: 0 -> 1 -> 2 -> 3 -> 4 -> 5 -> 7 -> 8 -> 9

MEDIUM SERIES

Permutations

Write a function that takes in an array of unique integers and returns an array of all permutations of those integers. If the input array is empty, your function should return an empty array.

Sample input: [1, 2, 3]
Sample output: [[1, 2, 3], [1, 3, 2], [2, 1, 3], [2, 3, 1], [3, 1, 2], [3, 2, 1]]

MEDIUM SERIES

Powerset

Write a function that takes in an array of unique integers and returns its powerset. The powerset P(X) of a set X is the set of all subsets of X. For example, the powerset of [1,2] is [[], [1], [2], [1,2]]. Note that the sets in the powerset do not need to be in any particular order.

Sample input: [1, 2, 3]
Sample output: [[], [1], [2], [3], [1, 2], [1, 3], [2, 3], [1, 2, 3]]

MEDIUM SERIES

Search In Sorted Matrix

You are given a two-dimensional array (matrix) of distinct integers where each row is sorted and each column is also sorted. The matrix does not necessarily have the same height and width. You are also given a target number, and you must write a function that returns an array of the row and column indices of the target number if it is contained in the matrix and [-1, -1] if it is not contained in the matrix.

Sample input:

```
[
    [1, 4, 7, 12, 15, 1000],
    [2, 5, 19, 31, 32, 1001],
    [3, 8, 24, 33, 35, 1002],
    [40, 41, 42, 44, 45, 1003],
    [99, 100, 103, 106, 128, 1004],
], 44
```

Sample output: [3, 3]

MEDIUM SERIES

Write a Min Max Stack class. The class should support pushing and popping values on and off the stack, peeking at values at the top of the stack, and getting both the minimum and the maximum values in the stack. All class methods, when considered independently, should run in constant time and with constant space.

Sample input:

```
-> push(5)
-> getMin()
-> getMax()
-> peek()
-> push(7)
-> getMin()
-> getMax()
-> peek()
-> push(2)
-> getMin()
-> getMax()
-> peek()
-> pop()
-> pop()
-> getMin()
-> getMax()
-> peek()
```

Sample output:

```
-> (push 5)
-> 5 (min)
-> 5 (max)
-> 5 (peek)
-> (push 7)
-> 5 (min)
-> 7 (max)
-> 7 (peek)
-> (push 2)
-> 2 (min)
-> 7 (max)
-> 2 (peek)
-> 2 (pop)
-> 7 (pop)
-> 5 (min)
-> 5 (max)
-> 5 (peek)
```

MEDIUM SERIES

Balanced Brackets

Write a function that takes in a string made up of brackets ("(", "[", "{", ")", "]", and "}") and other optional characters. The function should return a boolean representing whether or not the string is balanced in regards to brackets. A string is said to be balanced if it has as many opening brackets of a given type as it has closing brackets of that type and if no bracket is unmatched. Note that a closing bracket cannot match a corresponding opening bracket that comes after it. Similarly, brackets cannot overlap each other as in "[(])".

Sample input: "([])(){}(())()()"
Sample output: True (it is balanced)

MEDIUM SERIES

Longest Palindromic Substring

Write a function that, given a string, returns its longest palindromic substring. A palindrome is defined as a string that is written the same forward and backward. Assume that there will only be one longest palindromic substring.

Sample input: "abaxyzzyxf"
Sample output: "xyzzyx"

MEDIUM SERIES

Suffix Trie Construction

Write a class for a suffix-trie-like data structure. The class should have a "root" property set to be the root node of the trie. The class should support creation from a string and the searching of strings. The creation method (called populateSuffixTrieFrom()) will be called when the class is instantiated and should populate the "root" property of the class. Note that every string added to the trie should end with the special "endSymbol" character: "*".

Sample input (for creation): "babc"
Sample output (for creation):

```
{
  "c": {"*": True},
  "b": {
    "c": {"*": True},
    "a": {"b": {"c": {"*": True}}},
  },
  "a": {"b": {"c": {"*": True}}},
}
```

Sample input (for searching in the suffix trie above): "abc"
Sample output (for searching in the suffix trie above): True

HARD SERIES

Four Number Sum

Write a function that takes in a non-empty array of distinct integers and an integer representing a target sum. The function should find all quadruplets in the array that sum up to the target sum and return a two-dimensional array of all these quadruplets in no particular order. If no four numbers sum up to the target sum, the function should return an empty array.

Sample input: [7, 6, 4, -1, 1, 2], 16
Sample output: [[7, 6, 4, -1], [7, 6, 1, 2]]

HARD SERIES

Subarray Sort

Write a function that takes in an array of integers of length at least 2. The function should return an array of the starting and ending indices of the smallest subarray in the input array that needs to be sorted in place in order for the entire input array to be sorted. If the input array is already sorted, the function should return [-1, -1].

Sample input: [1, 2, 4, 7, 10, 11, 7, 12, 6, 7, 16, 18, 19]
Sample output: [3, 9]

HARD SERIES

Largest Range

Write a function that takes in an array of integers and returns an array of length 2 representing the largest range of numbers contained in that array. The first number in the output array should be the first number in the range while the second number should be the last number in the range. A range of numbers is defined as a set of numbers that come right after each other in the set of real integers. For instance, the output array [2, 6] represents the range {2, 3, 4, 5, 6}, which is a range of length 5. Note that numbers do not need to be ordered or adjacent in the array in order to form a range. Assume that there will only be one largest range.

Sample input: [1, 11, 3, 0, 15, 5, 2, 4, 10, 7, 12, 6]
Sample output: [0, 7]

HARD SERIES

Min Rewards

Imagine that you're a teacher who's just graded the final exam in a class. You have a list of student scores on the final exam in a particular order (not necessarily sorted), and you want to reward your students. You decide to do so fairly by giving them arbitrary rewards following two rules: first, all students must receive at least one reward; second, any given student must receive strictly more rewards than an adjacent student (a student immediately to the left or to the right) with a lower score and must receive strictly fewer rewards than an adjacent student with a higher score. Assume that all students have different scores; in other words, the scores are all unique. Write a function that takes in a list of scores and returns the minimum number of rewards that you must give out to students, all the while satisfying the two rules.

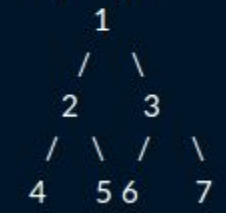
Sample input: [8, 4, 2, 1, 3, 6, 7, 9, 5]
Sample output: 25 ([4, 3, 2, 1, 2, 3, 4, 5, 1])

HARD SERIES

Max Path Sum In Binary Tree

Write a function that takes in a Binary Tree and returns its max path sum. A path is a collection of connected nodes where no node is connected to more than two other nodes; a path sum is the sum of the values of the nodes in a particular path. Each Binary Tree node has a value stored in a property called "value" and two children nodes stored in properties called "left" and "right," respectively. Children nodes can either be Binary Tree nodes themselves or the None (null) value.

Sample input:



Sample output: 18

HARD SERIES

Max Sum Increasing Subsequence

Given an non-empty array of integers, write a function that returns an array of length 2. The first element in the output array should be an integer value representing the greatest sum that can be generated from a strictly-increasing subsequence in the array. The second element should be an array of the numbers in that subsequence. A subsequence is defined as a set of numbers that are not necessarily adjacent but that are in the same order as they appear in the array. Assume that there will only be one increasing subsequence with the greatest sum.

Sample input: [10, 70, 20, 30, 50, 11, 30]
Sample output: [110, [10, 20, 30, 50]]

HARD SERIES

Longest Common Subsequence

Implement a function that returns the longest subsequence common to two given strings. A subsequence is defined as a group of characters that appear sequentially, with no importance given to their actual position in a string. In other words, characters do not need to appear consecutively in order to form a subsequence. Assume that there will only be one longest common subsequence.

Sample input: "ZXVVYZW", "XKYKZPW"
Sample output: ["X", "Y", "Z", "W"]

HARD SERIES

Min Number Of Jumps

You are given a non-empty array of integers. Each element represents the maximum number of steps you can take forward. For example, if the element at index 1 is 3, you can go from index 1 to index 2, 3, or 4. Write a function that returns the minimum number of jumps needed to reach the final index. Note that jumping from index i to index i + x always constitutes 1 jump, no matter how large x is.

Sample input: [3, 4, 2, 1, 2, 3, 7, 1, 1, 1, 3]
Sample output: 4 (3 --> 4 or 2 --> 2 or 3 --> 7 --> 3)

HARD SERIES

Water Area

You are given an array of integers. Each non-zero integer represents the height of a pillar of width 1. Imagine water being poured over all of the pillars and return the surface area of the water trapped between the pillars viewed from the front. Note that spilled water should be ignored. Refer to the first minute of the video explanation below for a visual example.

Sample input: [0, 8, 0, 0, 5, 0, 0, 10, 0, 0, 1, 1, 0, 3]
Sample output: 48

HARD SERIES

Knapsack Problem

You are given an array of arrays. Each subarray in this array holds two integer values and represents an item; the first integer is the item's value, and the second integer is the item's weight. You are also given an integer representing the maximum capacity of a knapsack that you have. Your goal is to fit items in your knapsack, all the while maximizing their combined value. Note that the sum of the weights of the items that you pick cannot exceed the knapsack's capacity. Write a function that returns the maximized combined value of the items that you should pick, as well as an array of the indices of each item picked. Assume that there will only be one combination of items that maximizes the total value in the knapsack.

Sample input: [[1, 2], [4, 3], [5, 6], [6, 7]], 10
Sample output: [10, [1, 3]]

HARD SERIES

Disk Stacking

You are given a non-empty array of arrays. Each subarray holds three integers and represents a disk. These integers denote each disk's width, depth, and height, respectively. Your goal is to stack up the disks and to maximize the total height of the stack. A disk must have a strictly smaller width, depth, and height than any other disk below it. Write a function that returns an array of the disks in the final stack, starting with the top disk and ending with the bottom disk. Note that you cannot rotate disks; in other words, the integers in each subarray must represent [width, depth, height] at all times. Assume that there will only be one stack with the greatest total height.

Sample input: [[2, 1, 2], [3, 2, 3], [2, 2, 8], [2, 3, 4], [1, 2, 1], [4, 4, 5]]
Sample output: [[2, 1, 2], [3, 2, 3], [4, 4, 5]]

HARD SERIES

Topological Sort

You are given a list of arbitrary jobs that need to be completed; these jobs are represented by integers. You are also given a list of dependencies. A dependency is represented as a pair of jobs where the first job is prerequisite of the second one. In other words, the second job depends on the first one; it can only be completed once the first job is completed. Write a function that takes in a list of jobs and a list of dependencies and returns a list containing a valid order in which the given jobs can be completed. If no such order exists, the function should return an empty list.

Sample input: [1, 2, 3, 4], [[1, 2], [1, 3], [3, 2], [4, 2], [4, 3]]
Sample output: [1, 4, 3, 2] or [4, 1, 3, 2]

HARD SERIES

Boggle Board

You are given a two-dimensional array (matrix) of potentially unequal height and width containing letters; this matrix represents a boggle board. You are also given a list of words. Write a function that returns an array of all the words contained in the boggle board. A word is constructed in the boggle board by connecting adjacent (horizontally, vertically, or diagonally) letters, without using any single letter at a given position more than once; while words can of course have repeated letters, those repeated letters must come from different positions in the boggle board in order for the word to be contained in the board. Note that two or more words are allowed to overlap and use the same letters in the boggle board.

Sample input:

```
[
  ["t", "h", "i", "s", "i", "s", "a"],
  ["s", "i", "m", "p", "l", "e", "x"],
  ["b", "x", "x", "x", "x", "e", "b"],
  ["x", "o", "g", "g", "l", "x", "o"],
  ["x", "x", "x", "D", "T", "r", "a"],
  ["R", "E", "P", "E", "A", "d", "x"],
  ["x", "x", "x", "x", "x", "x", "x"],
  ["N", "O", "T", "R", "E", "-", "P"],
  ["x", "x", "D", "E", "T", "A", "E"],
],
["this", "is", "not", "a", "simple", "boggle", "board", "test", "REPEATED", "NOTRE-PEATED"]
```

Sample output: ["this", "is", "a", "simple", "boggle", "board", "NOTRE-PEATED"]

HARD SERIES

Continuous Median

Write a class that can support the following two functionalities: 1) the continuous insertion of numbers and 2) the instant (O(1) time) retrieval of the median of the numbers that have been inserted thus far. The getMedian() method, which handles the retrieval of the median, has already been written for you. You simply have to write the insert() method.

Sample input: insert(5), insert(10), getMedian(), insert(100), getMedian()
Sample output: -, -, 7.5, -, 10

HARD SERIES

Find Loop

Write a function that takes in the head of a Singly Linked List that contains a loop (in other words, the list's tail node points to some node in the list instead of the None (null) value). The function should return the node (the actual node - not just its value) from which the loop originates in constant space. Note that every node in the Singly Linked List has a "value" property storing its value as well as a "next" property pointing to the next node in the list.

Sample input:

```
n0 -> n1 -> n2 -> n3 -> n4 -> n5 -> n6
                ^          v
                n9 <- n8 <- n7
```

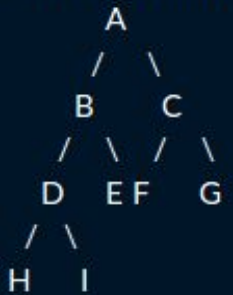
Sample output: n4

HARD SERIES

Lowest Common Manager

You're given three inputs, all of which are instances of a class that have a "directReports" property pointing to their direct reports. The first input is the top manager in an organizational chart (i.e., the only instance that is not anybody else's direct report), and the other two inputs are reports in the organizational chart. Write a function that returns the lowest common manager to the two reports.

Sample input: Node A, Node E, Node I (from the ancestral tree below)



Sample output: Node B

HARD SERIES

Shifted Binary Search

Write a function that takes in a sorted array of integers as well as a target integer. The caveat is that the numbers in the array have been shifted by some amount; in other words, they have been moved to the left or the right by one or more positions. For example, [1, 2, 3, 4] might become [3, 4, 1, 2]. The function should use a variation of the Binary Search algorithm to find if the target number is contained in the array and should return its index if it is, otherwise -1.

Sample input: [45, 61, 71, 72, 73, 0, 1, 21, 33, 45], 33
Sample output: 8

HARD SERIES

Search For Range

Write a function that takes in a sorted array of integers as well as a target integer. The function should use a variation of the Binary Search algorithm to find a range of indices in between which the target number is contained in the array and should return this range in the form of an array. The first number in the output array should represent the first index at which the target number is located while the second number should represent the last index at which the target number is located. The function should return [-1, -1] if the number is not contained in the array.

Sample input: [0, 1, 21, 33, 45, 45, 45, 45, 45, 45, 61, 71, 73], 45
Sample output: [4, 9]

HARD SERIES

Quick Select

Write a function that takes in an array of distinct integers as well as an integer k and returns the kth smallest number in that array in linear time, on average. The array could be sorted, but isn't necessarily so.

Sample input: [8, 5, 2, 9, 7, 6, 3], 3
Sample output: 5

HARD SERIES

Quick Sort

Write a function that takes in an array of integers and returns a sorted version of that array. Use the Quick Sort algorithm to sort the array.

Sample input: [8, 5, 2, 9, 5, 6, 3]
Sample output: [2, 3, 5, 5, 6, 8, 9]

HARD SERIES

Heap Sort

Write a function that takes in an array of integers and returns a sorted version of that array. Use the Heap Sort algorithm to sort the array.

Sample input: [8, 5, 2, 9, 5, 6, 3]
Sample output: [2, 3, 5, 5, 6, 8, 9]

HARD SERIES

Longest Substring Without Duplication

Write a function that takes in a string and that returns its longest substring without duplicate characters. Assume that there will only be one longest substring without duplication.

Sample input: "clementisacap"
Sample output: "mentisac"

HARD SERIES

Underscorify Substring

Write a function that takes in two strings: a main string and a potential substring of the main string. The function should return a version of the main string with every instance of the substring in it wrapped between underscores. If two instances of the substring in the main string overlap each other or sit side by side, the underscores relevant to these two substrings should only appear on the far left of the left substring and on the far right of the right substring. If the main string does not contain the other string at all, return the main string intact.

Sample input: "testthis is a testtest to see if testestest it works", "test"
Sample output: "_test_this is a _testtest_ to see if _testestest_ it works"

HARD SERIES

Pattern Matcher

You are given two non-empty strings. The first one is a pattern consisting of only "x"s and / or "y"s; the other one is a normal string of alphanumeric characters. Write a function that checks whether or not the normal string matches the pattern. A string S0 is said to match a pattern if replacing all "x"s in the pattern with some string S1 and replacing all "y"s in the pattern with some string S2 yields the same string S0. If the input string does not match the input pattern, return an empty array; otherwise, return an array holding the representations of "x" and "y" in the normal string, in that order. If the pattern does not contain any "x"s or "y"s, the respective letter should be represented by an empty string in the final array that you return. Assume that there will never be more than one pair of strings S1 and S2 that appropriately represent "x" and "y" in the input string.

Sample input: "xxyxxy", "gogopowerrangergogopowerranger"
Sample output: ["go", "powerranger"]

HARD SERIES

Multi String Search

Write a function that takes in a "big" string and an array of "small" strings, all of which are smaller in length than the big string. The function should return an array of booleans, where each boolean represents whether or not the small string at that index in the array of small strings is contained in the big string. Note that you cannot use language-built-in string-matching methods.

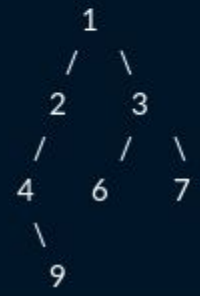
Sample input: "this is a big string", ["this", "yo", "is", "a", "bigger", "string", "kappa"]
Sample output: [True, False, True, True, False, True, False]

VERY HARD SERIES

Iterative In-order Traversal

Write a function that takes in a Binary Tree and traverses it using the in-order traversal technique but without using recursion. As the tree is being traversed, a callback function passed in as an argument to the main function should be called on each node (i.e. callback(currentNode)). Each Binary Tree node has a value stored in a property called "value," a parent node in a property called "parent," and two children nodes stored in properties called "left" and "right," respectively. Children nodes can either be Binary Tree nodes themselves or the None (null) value.

Sample input:



Sample output:

callback(4)
callback(9)
callback(2)
callback(1)
callback(6)
callback(3)
callback(7)

VERY HARD SERIES

Max Profit With K Transactions

You are given an array of integers representing the prices of a single stock on various days (each index in the array represents a different day). You are also given an integer k, which represents the number of transactions you are allowed to make. One transaction consists of buying the stock on a given day and selling it on another, later day. Write a function that returns the maximum profit that you can make buying and selling the stock, given k transactions. Note that you can only hold 1 share of the stock at a time; in other words, you cannot buy more than 1 share of the stock on any given day, and you cannot buy a share of the stock if you are still holding another share.

Sample input: [5, 11, 3, 50, 60, 90], 2
Sample output: 93 (Buy: 5, Sell: 11; Buy: 3, Sell: 90)

VERY HARD SERIES

Palindrome Partitioning Min Cuts

Given a non-empty string, write a function that returns the minimum number of cuts needed to perform on the string such that each remaining substring is a palindrome. A palindrome is defined as a string that is written the same forward as backward. Note that single-character strings are palindromes.

Sample input: "noonabbad"
Sample output: 2 ("noon | abba | d")

VERY HARD SERIES

Knuth–Morris–Pratt Algorithm

Write a function that takes in two strings and checks if the first string contains the second one using the Knuth–Morris–Pratt algorithm. The function should return a boolean.

Sample input: "aefoaefcdaefcdaed", "aefcdaed"
Sample output: True

VERY HARD SERIES

LRU Cache

Implement a class for a Least Recently Used (LRU) Cache. The cache should support inserting key / value pairs (the insertKeyValuePair() method), retrieving a key's value (the getValueFromKey() method), and retrieving the most recently "active" key (the getMostRecentKey() method); each of these methods should run in constant time. When a key / value pair is inserted or a key's value is retrieved, the key in question should become the most recent key. Also, the LRUCache class should store a maxSize property set to the size of the cache, which is passed in as an argument during instantiation. This size represents the maximum number of key / value pairs that the cache can hold at once. If a key / value pair is added to the cache when it has reached maximum capacity, the least recently used ("active") key / value pair should be evicted from the cache and no longer retrievable; the newly added key / value pair should effectively replace it. Inserting a key / value pair with an already existing key should simply replace the key's value in the cache with the new value and should not evict a key / value pair if the cache is full. Attempting to retrieve a value from a key that is not in the cache should return the None (null) value.

Sample input: (for an LRU Cache of size 3)
insertKeyValuePair("a", 1)
insertKeyValuePair("b", 2)
insertKeyValuePair("c", 3)
getMostRecentKey()
getValueFromKey("a")
getMostRecentKey()
insertKeyValuePair("d", 4)
getValueFromKey("b")
insertKeyValuePair("a", 5)
getValueFromKey("a")

Sample output:
-
-
-
"c"
1
"a"
-
None
-
5

VERY HARD SERIES

Number Of Possible Binary Tree Topologies

Write a function that takes in a non-negative integer n and that returns the number of possible Binary Tree topologies that can be created with exactly n nodes. A Binary Tree topology is defined as any Binary Tree configuration, irrespective of node values. For instance, there exist only two Binary Tree topologies when n is equal to 2: a root node with a left node, and a root node with a right node. Node that when n is equal to 0, there is one topology that can be created: the None (null) node.

Sample input: 3
Sample output: 5

VERY HARD SERIES

Merge Sort

Write a function that takes in an array of integers and returns a sorted version of that array. Use the Merge Sort algorithm to sort the array.

Sample input: [8, 5, 2, 9, 5, 6, 3]
Sample output: [2, 3, 5, 5, 6, 8, 9]

THE END