Assignment 4

Short Answers

Answer the following questions with complete sentences in your own words. You are encouraged to conduct your own research online or through other methods before answering the questions. If you research online, please consult multiple sources before you write down your answers. You are expected to be able to explain your answers in detail. If you are unable to understand or answer some of the questions asked or the contents you have found, please provide the sources such as the website link or book, and mark your question in red.

1. How to handle exceptions in Python?
   You can use try, except, else, finally to handle exceptions. The try lets you test codes for error and the except handles the error. The error can be handled generally or can be handled by specific error type. Else lets the program execute when there is no error and finally executes regardless of whether there is an error or not.
2. What are escape characters? What is the syntax?
   Escape characters are characters in python that don't get printed as part of the output. The syntax is backslash \ to escape the characters. For example \n is a new line and \\ allows you to input a backslash as a character in the output.
3. What are different modes for opening a file in Python?
   The modes are read, write, and append. Read only allows you to view the file. Write replaces the original contents of the file with new data. Append adds to the file.
4. What is a Matrix?
   A matrix is a 2D array/list, which can be accessed similarly to arrays using rows and column indexing.
5. What is a Node?
   A node is a fundamental unit in a graph. In a linked list it consists of value and next. In a binary tree, it consists of value, left, and right.
6. How to traverse a Simple Linked List?
   You can traverse the linked list by creating a pointer to the head and setting the next value as the pointer value. Or you could also shift the head of the linked list by setting it equal to next.
7. What is a DAG?
   A DAG is a directed acyclic graph, which is a graph with directional connections and where no directed cycles are formed.
8. What's a Binary Search Tree?
   A binary search tree is a binary tree data structure where a node's left child must be less than its parent's and the right child must have a value greater than its parent's.
9. List different ways to traverse a Binary Tree and explain the difference.

There is depth-first search, which traverses a binary tree vertically, going down to the deepest level. Depth-first search has 3 search orders: inorder, preorder and postorder. For inorder traversal, you traverse the left subtree, then the root, then the right subtree. Preorder traverses the root, the left subtree, then the right subtree. Postorder traverses the left subtree, the right subtree, and then the root.

There is also breadth-first search, which traverses the tree by level.

10. Explain the logic behind binary search.

Binary search looks for an element in sorted arrays. It searches for the middle value, then based on the comparison of the middle value and the target, searches either the top half or bottom half of the array, and continues that process until it finds the element.

Coding Practice

Write algorithms in Python to Complete following questions. Please write your own answers. You are highly encouraged to present more than one way to answer the questions. Please follow the syntax, and you need to comment on each piece of code you write to clarify your purpose. Also you need to clearly explain your logic of the code for each question. Clearly state your assumptions if you have any. You may discuss with others about the questions, but please write your own code.

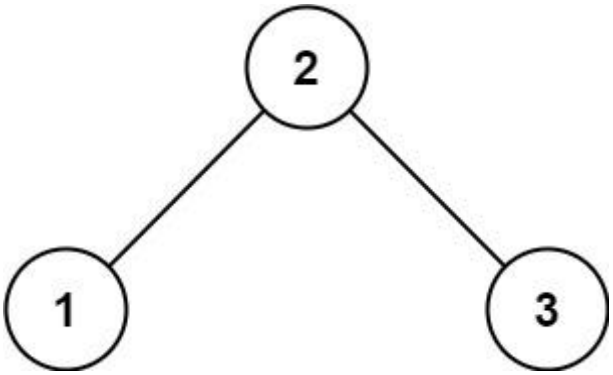
For all questions, list out your time complexity and space complexity.

1. Given the root of a binary tree, determine if it is a valid binary search tree (BST).

A valid BST is define as follows:

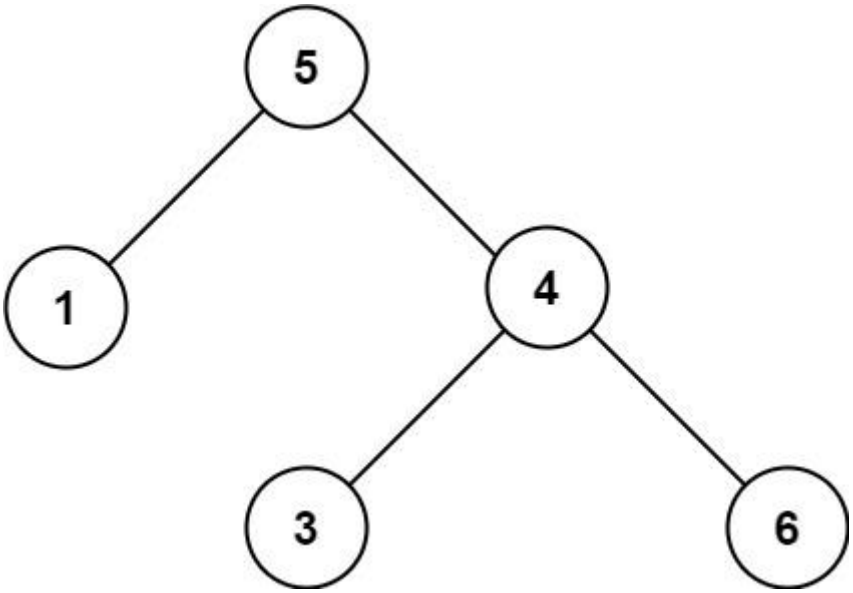
- The left subtree of a node contains only nodes with keys less than the node's key.
- The right subtree of a node contains only nodes with keys greater than the node's key.
- Both the left and right subtree must also be binary search trees.

Example 1:



Input: root = [2,1,3]
Output: true

Example 2:



Input: root = [5,1,4,null,null,3,6]
Output: false

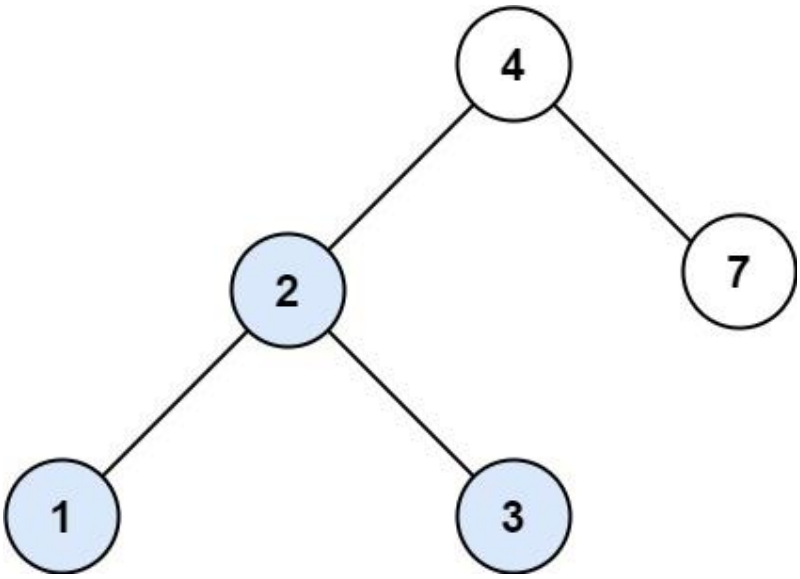
This check can be done recursively by traversing the left and right subtrees, making sure the left is less than the parent and the right is greater than the parent and that the left and right trees are also BSTs. Time complexity is O(n) because every node is checked once and space complexity is O(n) because every node is stored once.

```python
def isValidBST(root: TreeNode, low=float('-inf'), high=float('inf')):
    if not root:
        return True
    elif not low < root.value < high:
        return False
    return isValidBST(root.left,low, root.value) and
isValidBST(root.right,root.value, high)
```

2. You are given the root of a binary search tree (BST) and an integer val.

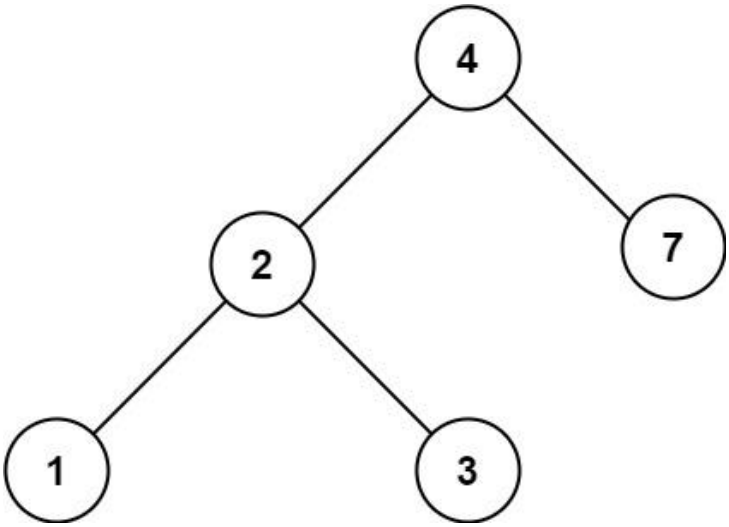
Find the node in the BST that the node's value equals val and return the subtree rooted with that node. If such a node does not exist, return null.

Example 1:



Input: root = [4, 2, 7, 1, 3], val = 2
Output: [2, 1, 3]

Example 2:



Input: root = [4, 2, 7, 1, 3], val = 5
Output: []

To find the value just traverse the BST similarly to the binary search algorithm, where you traverse the half that the value could fall under until you find the value. The time complexity for a BST search is O(logn) and the space complexity is O(n) because all of the nodes are stored in the worst case.

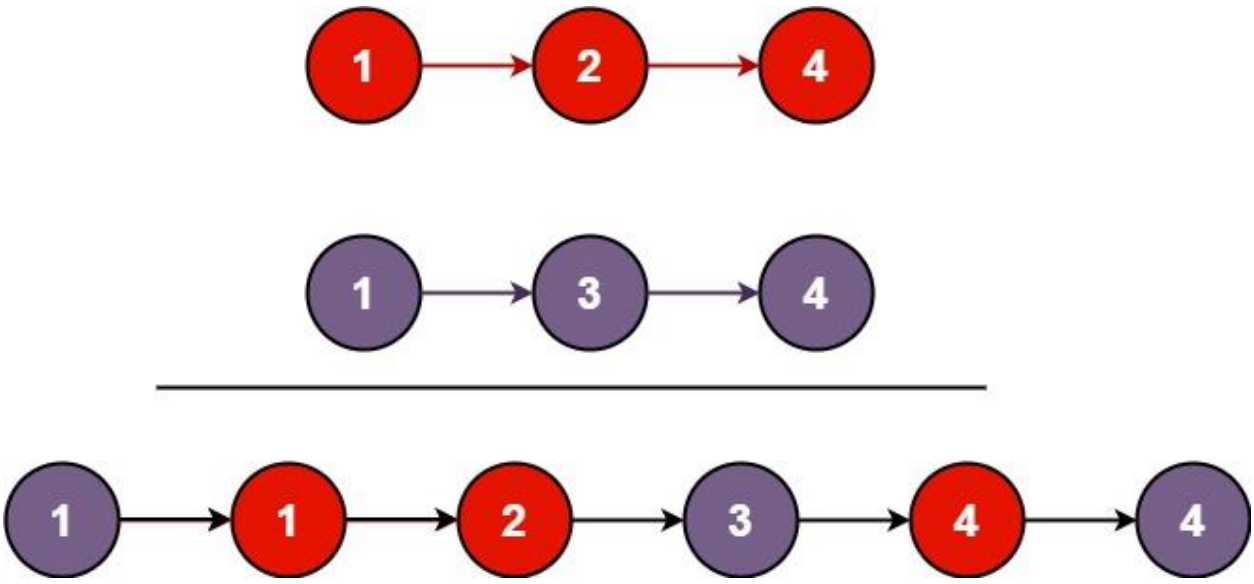
```
def searchBST(root: TreeNode, val):
    if not root:
        return None
    if root.value == val:
        return root
    elif root.value < val:
        return searchBST(root.right, val)
    elif root.value > val:
        return searchBST(root.left, val)
```

3. You are given the heads of two sorted linked lists list1 and list2.

Merge the two lists in a one sorted list. The list should be made by splicing together the nodes of the first two lists.

Return the head of the merged linked list.

Example 1:



Input: list1 = [1,2,4], list2 = [1,3,4]
Output: [1,1,2,3,4,4]

Example 2:

Input: list1 = [], list2 = []
Output: []

Example 3:

Input: list1 = [], list2 = [0]
Output: [0]

ListNode class is provided below:

```
class ListNode(object):
    def __init__(self, val=0, next=None):
        self.val = val
        self.next = next
```

The most intuitive way was to use recursion to traverse the linked lists, based on which node has a bigger value between the two lists. Time complexity is O(n) because the recursion traverses the two lists node by node.

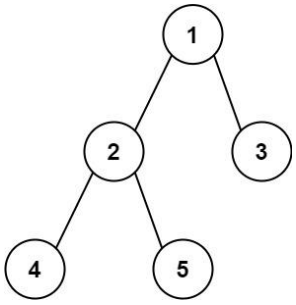
```python
def mergeLists(l1, l2):
    if not l1 or not l2:
        return l1 or l2
    if l1.val < l2.val:
        l1.next = mergeLists(l1.next, l2)
        return l1
    else:
        l2.next = mergeLists(l1, l2.next)
        return l2
```

4. Given the root of a binary tree, return the length of the diameter of the tree.

The diameter of a binary tree is the length of the longest path between any two nodes in a tree. This path may or may not pass through the root.

The length of a path between two nodes is represented by the number of edges between them.

Example 1:



Input: root = [1, 2, 3, 4, 5]
Output: 3
Explanation: 3 is the length of the path [4, 2, 1, 3] or [5, 2, 1, 3].

Example 2:

Input: root = [1, 2]
Output: 1

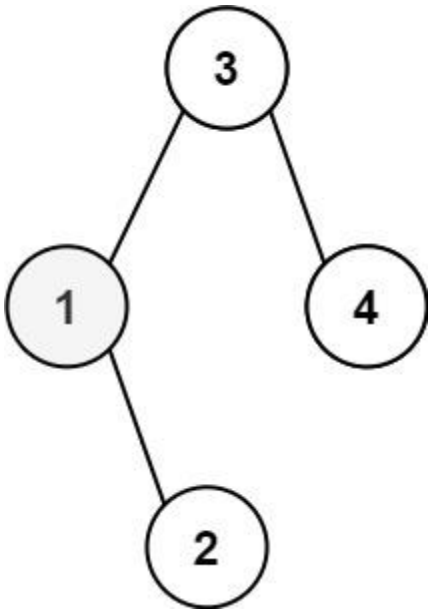
To find the diameter you can recursively find the depth of the nodes and find the max sum of the left and right subtrees. The time complexity is O(n) because you traverse each node once, and the space complexity is also O(n) because you need the whole tree.

```
def diameter(root):
    dia = 0
    #function to find the depth of a node
    def height(p):
        if not p:
            return 0
        left, right = height(p.left), height(p.right)
        # update the value of the diameter
        ans = max(dia, left+right)
        return 1+max(left, right)

    height(root)
    return dia
```

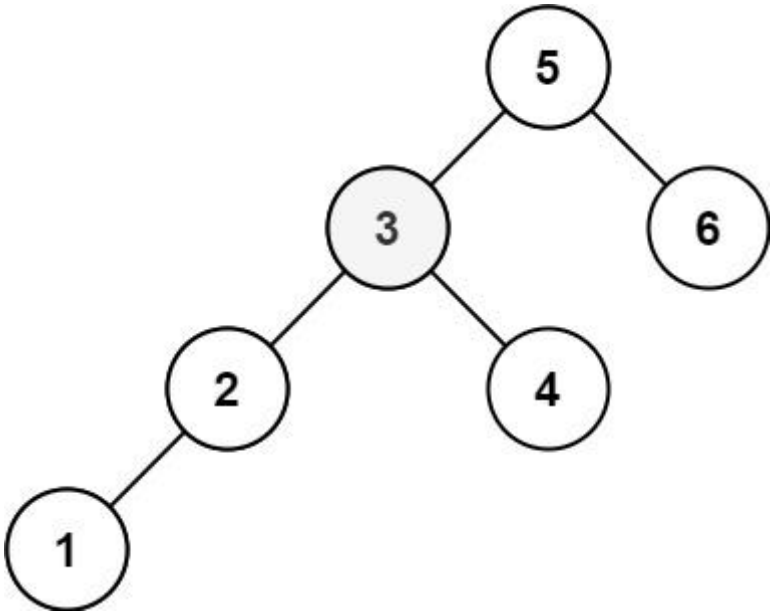
5. Given the root of a binary search tree, and an integer k, return the kth smallest value (1-indexed) of all the values of the nodes in the tree.

Example 1:



Input: root = [3,1,4,null,2], k = 1
Output: 1

Example 2:



Input: root = [5,3,6,2,4,null,null,1], k = 3
Output: 3

For this problem you can do an inorder traversal of the BST and append each node into an list, because inorder traversal returns the nodes sorted in ascending order. Then you can just return the kth value in the list. The time complexity of inorder traversal is O(n). Space complexity is also O(n) because you need all the nodes and a list of length n.

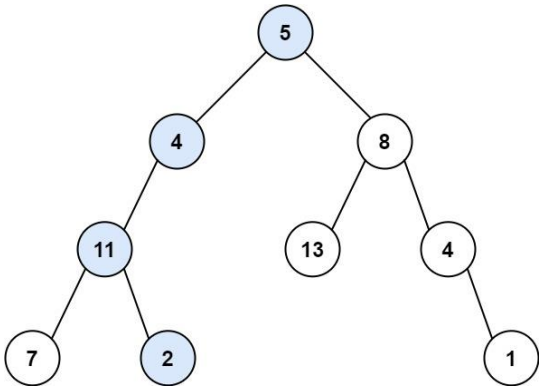
```python
def kthSmallest(root, k):
    nodes = []
    inorder(root, nodes)
    return nodes[k-1]

def inorder(root, nodes):
    if not root:
        return
    inorder(root.left, nodes)
    nodes.append(root.value)
    inorder(root.right, nodes)
```

6. Given the root of a binary tree and an integer targetSum, return true if the tree has a root-to-leaf path such that adding up all the values along the path equals targetSum.

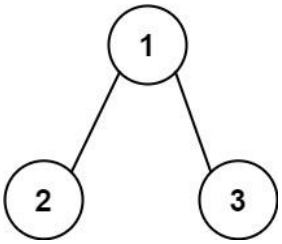
A leaf is a node with no children

Example 1:



Input: root = [5,4,8,11,null,13,4,7,2,null,null,null,1], targetSum = 22
Output: true

Example 2:



Input: root = [1,2,3], targetSum = 5
Output: false

For this problem you can traverse the tree from root to leaf, and subtract the value of each node visited from the targetSum. If the final node is a leaf and the value matches the targetSum, then there is a path that adds up to the target sum. Recurse over the left and right of the BST. The time complexity is O(n) because the worst case is traversing through all the nodes. The space complexity is O(1) because you are keeping track of targetSum.

```
def hasPathSum(root, targetSum):
    if not root:
        return False
    if not root.left and not root.right and root.value == targetSum:
        return True
    targetSum -= root.value
    return hasPathSum(root.left, targetSum) or hasPathSum(root.right, targetSum)
```