Deadline: 22nd December 11.59 pm

1. Write a recursive function, **find_paths**, that **takes a 2D maze represented as a list of lists (0 for open path, 1 for wall)**, and **returns True** if it is possible to travel from the top-left corner to the bottom-right corner, and **False** otherwise.

```
maze_grid = [
    [0, 0, 1, 0],
    [0, 0, 0, 0],
    [1, 0, 1, 0],
    [0, 0, 0, 0]
]
# Example mazes
maze1 = [
    [0, 1, 0, 0],
    [0, 1, 1, 0],
    [0, 0, 1, 0],
    [1, 0, 0, 0]
]

maze2 = [
    [0, 0, 0, 0],
    [1, 1, 1, 1],
    [0, 0, 0, 0],
    [1, 1, 1, 1],
    [0, 0, 0, 0]
]

maze3 = [
    [0, 1, 0, 0],
    [0, 0, 1, 0],
    [1, 0, 0, 0],
    [1, 1, 1, 0]
]

# Test the function with example mazes
print(find_paths(maze1, 0, 0))  # Should return True
print(find_paths(maze2, 0, 0))  # Should return False
print(find_paths(maze3, 0, 0))  # Should return True
```

---

2. Create a recursive function named **generate_parentheses that accepts a positive integer n as input** and **produces a string** representing a specific pattern of parentheses. The pattern should consist of (n-1) pairs of parentheses, all nested within an additional pair.

```
parentheses_combinations = generate_parentheses(3)
print(parentheses_combinations)
'(()())'

parentheses_combinations = generate_parentheses(4)
print(parentheses_combinations)
```

'(()()())'

---

3. Write a recursive function, **digital_root, that takes a positive integer** and **returns its digital root**. The digital root is the recursive sum of all the digits until a single-digit number is obtained.

   number = 942
   root = **digital_root(number)**
   print(root) # Output should be **6**
   Explanation: (9 + 4 + 2 = 15, then 1 + 5 = 6)

   number = 8765
   root = **digital_root(number)**
   print(root) # Output should be **8**
   Explanation: (8 + 7 + 6 + 5 = 26, then 2 + 6 = 8)

---

4. Write a recursive function **named replace_consonants that takes a string** as input and replaces all consonants with the letter 'X'. The function should **return the modified string**.

   text = "Hello, World!"
   replacement = **replace_consonants(text)**
   # Output: 'XeXXo, XoXXd!'

   In this example, the function should replace all consonants in the string "Hello, World!" with 'X', resulting in the modified string 'XeXXo, XoXXd!'. **The implementation should use recursion and not involve any loops**

---

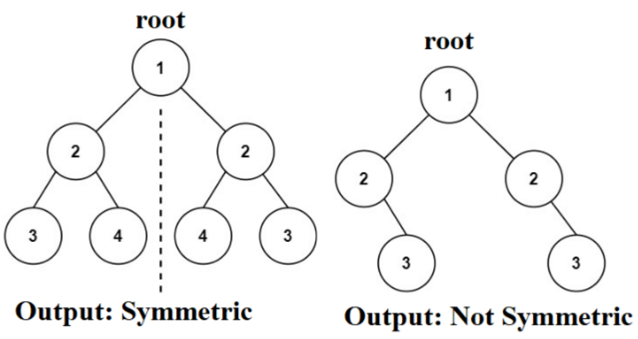
5. Develop a recursive function named **alternating_sum that takes a list of integers as inpu**t and computes the alternating sum by subtracting every other number. The function should **return the result**.

   numbers = [1, 2, 3, 4, 5]
   alt_sum = **alternating_sum(numbers)**
   # Output: **3**

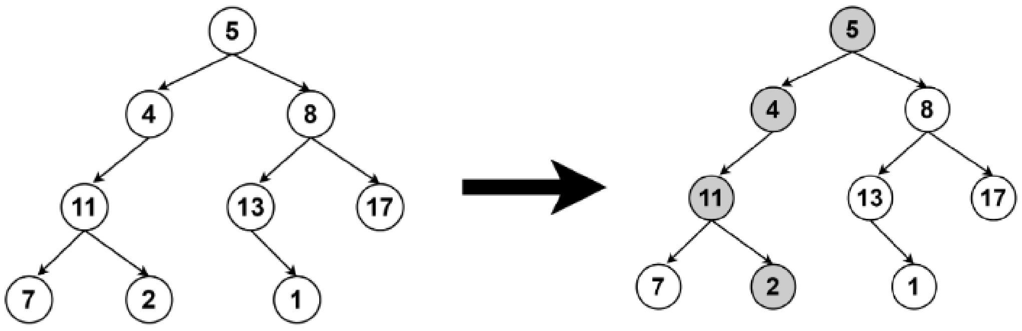
   In this example, the function should compute the alternating sum of the list [1, 2, 3, 4, 5] by subtracting every other number. The calculation is performed as follows: 1 - 2 + 3 - 4 + 5, resulting in an alternating sum of 3. **The implementation should utilize recursion and should not involve any loops.**

---

6. Write a Python function **isSymmetric(root)** that takes the root of a binary tree, check whether it is a mirror of itself (i.e., symmetric around its center).



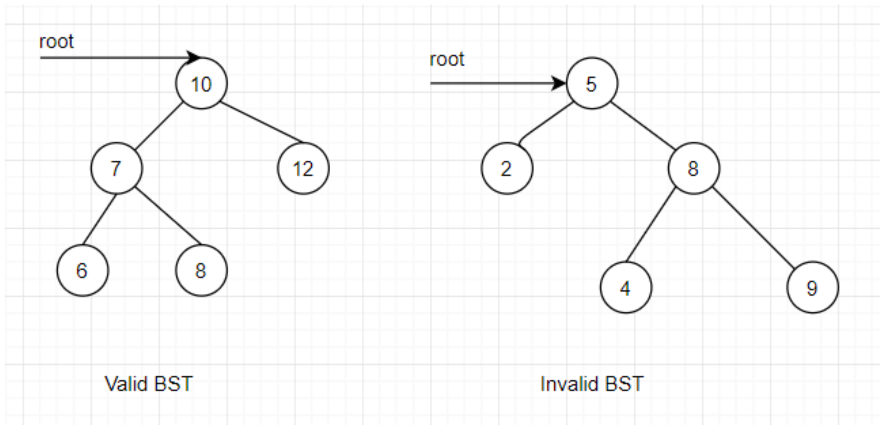
7. Write a method **hasPathSum which takes a root of a binary tree and a targetSum as input** parameters, **return True** if the tree has a root-to-leaf path such that adding up all the values along the path equals targetSum, **False** otherwise.



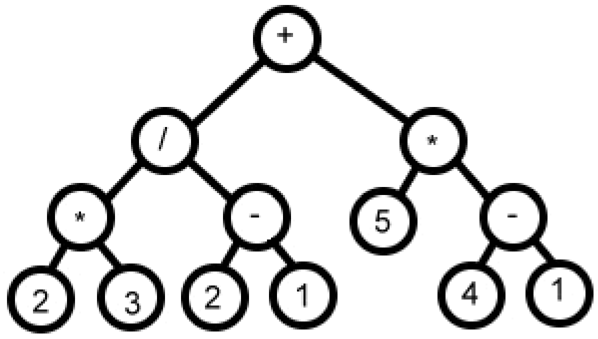
Sample Output:
print(**hasPathSum(root, 22)**) # Output: **True**
Explanation: 5 -> 4 -> 11 -> 2

print(**hasPathSum(root, 27)**) # Output: **True**
Explanation: 5 -> 4 -> 11 -> 7

8. Create a function that checks whether a given binary tree is a valid Binary Search Tree (BST). A BST is a binary tree where the left subtree of a node contains only nodes with values less than the node's value, and the right subtree contains only nodes with values greater than the node's value.



9. Evaluate a given binary expression tree representing algebraic expressions. A binary expression tree is a binary tree, where the operators are stored in the tree's internal nodes, and the leaves contain constants. Assume that each node of the binary expression tree has zero or two children. The supported operators are +, -, * and /.



Expression tree for 2*3/(2-1)+5*(4-1)

So, given the expression tree output of your code should be the final value of the represented expression, which is 6+5*3 = 2