**Spring 2020**

**Applied Data Structures and Algorithms**

**Recursion Homework**

**Homework**

1. Given an integer array *nums* and an integer *target*, determine if there is a subset of numbers in *nums* that sums to the given target. Feel free to write a helper (recursive) method.

Examples:

nums = [2, 4, 8] and target = 10 → true

nums = [2, 4, 8] and target = 14 → true

nums = [2, 4, 8] and target = 9 → false

nums = [1, 5, 9, 2, 4] and target = 3 → true

nums = [1, 5, 9, 2, 4] and target = 25 → false

C++: bool subsetSum(const vector<int>& nums, int target)  
Java: class Main{ public static boolean subsetSum(List<Integer> nums, int target) }  
Python: def subset\_sum(nums, target)

1. Given an integer array *nums*, determine if it is possible to divide *nums* in two groups, so that the sums of the two groups are equal. Any of the two groups can be empty. Feel free to write a helper (recursive) method.

Examples:

nums = [4, 4] → true

nums = [5, 2] → false

nums = [5, 2, 3] → true

nums = [-2, 2] → true

nums = [] → true

C++: bool canDivide(const vector<int>& nums)  
Java: class Main{ public static boolean canDivide(List<Integer> nums) }  
Python: def can\_divide(nums)

1. Given the root node of a binary search tree (with no duplicates), return the sum of values of all nodes with value between minVal and maxVal (inclusive).

Examples:

7

/ \

3 10

/ \ / \

-1 5 9 12

minVal = 2, maxVal = 10 → 3 + 5 + 7 + 9 + 10 = 34

minVal = 3, maxVal = 8 → 3 + 5 + 7 = 15

minVal = 10, maxVal = 20 → 10 + 12 = 22

minVal = 20, maxVal = 30 → 0

C++

/\*\*

\* struct TreeNode {

\* int val;

\* TreeNode\* left;

\* TreeNode\* right;

\* TreeNode(int x) : val(x), left(NULL), right(NULL) {}

\* };

\*/

int rangeSum(const TreeNode\* root, int minVal, int maxVal)

Java

/\*\*

\* public class TreeNode {

\* int val;

\* TreeNode left;

\* TreeNode right;

\* TreeNode(int x) { val = x; }

\* }

\*/

class Main { public int rangeSum(TreeNode root, int minVal, int maxVal) }

Python

# class TreeNode:

# def \_\_init\_\_(self, x):

# self.val = x

# self.left = None

# self.right = None

def range\_sum(root, min\_val, max\_val)

1. Given *n* pairs of parentheses, write a function to generate all combinations of well-formed parentheses.

Example:

n = 3 →

[

"((()))",

"(()())",

"(())()",

"()(())",

"()()()"

]

C++: std::vector<string> generateParentheses(int n)  
Java: class Main{ public List<String> generateParentheses(int n) }  
Python: def generate\_parentheses(n)

1. Write a function that receives the root of a binary tree as input and inverts it. Do not create a new tree, modify the one that was passed as input. Feel free to write a helper (recursive) method.

Examples:

|  |  |
| --- | --- |
| Tree (before method call) | Tree (after method call) |
| 7  / \  3 10  / \ / \  -1 5 9 12 | 7  / \  10 3  / \ / \  12 9 5 -1 |
| 2  / \  3 1 | 2  / \  1 3 |
| 7  / \  3 10  / \  9 12 | 7  / \  10 3  / \  12 9 |

C++

/\*\*

\* struct TreeNode {

\* int val;

\* TreeNode\* left;

\* TreeNode\* right;

\* TreeNode(int x) : val(x), left(NULL), right(NULL) {}

\* };

\*/

void invert(TreeNode\* root)

Java

/\*\*

\* public class TreeNode {

\* int val;

\* TreeNode left;

\* TreeNode right;

\* TreeNode(int x) { val = x; }

\* }

\*/

class Main { public void invert(TreeNode root) }

Python

# class TreeNode:

# def \_\_init\_\_(self, x):

# self.val = x

# self.left = None

# self.right = None

def invert(root)