**Prefix Sum QUESTIONS**

Table of Contents

[**Theory** 2](#_Toc156730153)

[What is Prefix sum? 2](#_Toc156730154)

[Examples of usage of prefix sum 2](#_Toc156730155)

[1. Efficient Range Sum Queries: 2](#_Toc156730156)

[2. Counting occurrence in range: 3](#_Toc156730157)

[3. Equal subarrays: 3](#_Toc156730158)

[**Important questions** 4](#_Toc156730159)

[1. Find the pivot integer 4](#_Toc156730160)

[Binary Search Tree Implementation: 5](#_Toc156730161)

[Traversals: 6](#_Toc156730162)

[Breadth First : 7](#_Toc156730163)

[LEVEL 1: **EASY** 8](#_Toc156730164)

[2. Check if two binary trees are same or not. 8](#_Toc156730165)

[4. Merge two binary trees. 8](#_Toc156730166)

[5. Convert a BST to tree with only right nodes. 8](#_Toc156730167)

[6. Average of levels in binary tree. 9](#_Toc156730168)

[7. Sorted array to BST 9](#_Toc156730169)

[8. Binary Tree tilt 9](#_Toc156730170)

[9. Diameter of binary tree 9](#_Toc156730171)

[LEVEL 2: **Medium** 10](#_Toc156730172)

[LEVEL 3: **Difficult** 11](#_Toc156730173)

[**SOLUTIONS:** 12](#_Toc156730174)

[**LEVEL 1:** 12](#_Toc156730175)

[**LEVEL 2:** 16](#_Toc156730176)

# **Theory**

## What is Prefix sum?

A prefix sum is an array derived from an input array such that the value at each position in the prefix sum array is the sum of all elements up to that position in the original array. Mathematically, if the input array is A and the prefix sum array is P, then *P*[*i*]=∑*j*=0*i*​*A*[*j*].

Usually used for range queries.

Eg:-

If A = [ 2, 3, 5, 9]

Prefix Sum = P

P[0] = A[0] , P[1]=A[0]+A[1] ,P[2] = A[0]+A[1]+A[2] ,P[3] = A[0]+A[1]+A[2]+A[3]

So P = [2, 2+3, 2+3+5 , 2+3+5+9] = [ 2, 5, 10, 19]

It is also equivalent to , P[i] = P[i-1] + A[i]

def fillPrefixSum(arr, n, prefixSum):

    prefixSum[0] = arr[0]

*# Adding present element with previous element*

    for i in range(1, n):

        prefixSum[i] = prefixSum[i - 1] + arr[i]

*# Driver code*

if \_\_name\_\_ == '\_\_main\_\_':

  arr = [10, 4, 16, 20]

  n = len(arr)

  prefixSum = [0 for i in range(n + 1)]

  fillPrefixSum(arr, n, prefixSum)

  print(prefixSum)

*#output: [10,14,30,50,0] ( here we have 0 at end, can have it at start too)*

## Examples of usage of prefix sum

### Efficient Range Sum Queries:

Prefix sums are often used to quickly calculate the sum of elements within a given range in an array. Example: Given an array arr, prefix\_sum[i] can be used to calculate the sum of elements from index 0 to i efficiently.

def range\_sum\_query(prefix\_sum, left, right):

    return prefix\_sum[right] - prefix\_sum[left - 1] if left > 0 else prefix\_sum[right]

*# Example usage:*

arr = [1, 2, 3, 4, 5]

prefix\_sum = [0] + [sum(arr[:i+1]) for i in range(len(arr))]

*#not an effiecient way to find prefix\_sum*

print(range\_sum\_query(prefix\_sum, 1, 3))  *# Output: Sum of elements from index 1 to 3*

### Counting occurrence in range:

Prefix sums can be used to efficiently count the occurrences of a value in a given range.

Example: Given an array arr, prefix\_sum[i] represents the count of a specific value up to index i.

Approach: here prefix sum will save the count of occurrence of the value

*#get occurences of num in given range[l,r] inclusive(for l,r 1-based index)*

def get\_occurrences(arr,val,left,right):

    n = len(arr)

    prefix\_sum = [0]\*(n+1)   *#will keep 0 in start*

    for i in range(1,n+1):

        if arr[i-1]==val:

            prefix\_sum[i] = prefix\_sum[i-1] + 1

        else:

            prefix\_sum[i] = prefix\_sum[i-1]

    return prefix\_sum[right]-(prefix\_sum[left - 1] if left > 0 else 0)

*# Example usage:*

arr = [1, 2, 2, 3, 4, 2, 5]

*# Output: Count of occurrences of 2 in the range [1, 5] ->[1,2,2,3,4] =2*

print(get\_occurrences(arr, 2, 1, 6))

These examples showcase how prefix sums can be used to efficiently solve various problems related to range queries, subarray sums, and counting occurrences. They help reduce time complexity by precalculating cumulative sums.

### Equal subarrays:

Check if there are two subarrays with same sum.

Example: Given an array arr, prefix\_sum[i] represents the count of a specific value up to index i.

Approach: here prefix

# https://leetcode.com/problems/number-of-ways-to-split-array/

class Solution:

    def waysToSplitArray(*self*, *nums*: List[int]) -> int:

        n = len(*nums*)

        pre = [0]\*n

        pre\_sum=0

*for* i *in* range(n):

            pre\_sum += *nums*[i]

            pre[i] = pre\_sum

        pairs=0

        last = pre[-1]

*for* i *in* range(n-1):   *#only till n-1 to have atleast one value in right subarray*

*if* pre[i] >= last-pre[i]:

                pairs+=1

*return* pairs

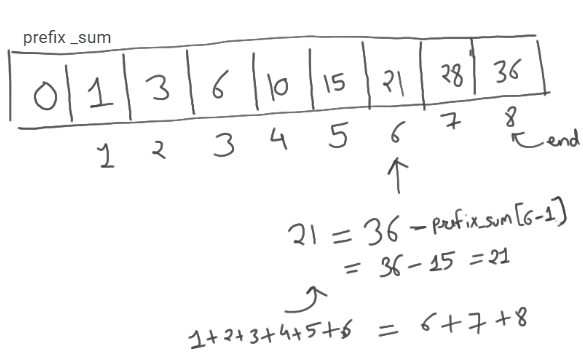
# **Important questions**

### Find the pivot integer

Given a positive integer n, find the pivot integer x such that: The sum of all elements between 1 and x inclusively equals the sum of all elements between x and n inclusively.(easy)

Link: <https://leetcode.com/problems/find-the-pivot-integer/description/>

Approach:-



class Solution:

    def pivotInteger(self, n: int) -> int:

        prefix\_sum = [0]\*(n+1)

        for i in range(1,n+1):

            prefix\_sum[i] = prefix\_sum[i-1] + i

        ans = -1

        end = prefix\_sum[-1]

        for i in range(1,n+1):

            if(prefix\_sum[i] == (end-prefix\_sum[i-1])):

                ans = i

                break

        return ans

Approach 2

By observation, if total sum till n = a perfect square, then pivot integer is square root of sum

n=4

x = (n\*(n+1)/2)\*\*0.5

print(x,x%1)            *#3.1622777..., 0.162277..*

print(-1 if x%1 else int(x))   *#x%1 is way to check if number is whole number*

n=8

x = (n\*(n+1)/2)\*\*0.5

print(x,x%1)          *#6.0 , 0.0*

print(-1 if x%1 else int(x))

[](https://www.youtube.com/watch?v=fFVZt-6sgyo)

## Binary Search Tree Implementation:

class Node:

    def \_\_init\_\_(self,data):

*self*.data = data

*self*.left = None

*self*.right = None

class BST:

    def \_\_init\_\_(self):

*self*.root = None

    def insert(self,data):

        def insert\_recursive(node,data):

            if node is None:

                return Node(data)

            if data < node.data:

                node.left = insert\_recursive(node.left,data)

            else:

                node.right = insert\_recursive(node.right,data)

            return node

*self*.root = insert\_recursive(*self*.root, data)

    def inorder(self):

        def inorder\_traversal(node):

            if node is not None:

                inorder\_traversal(node.left)

                print(node.data,end=" ")

                inorder\_traversal(node.right)

        inorder\_traversal(*self*.root)

    def search(self,data):

        def search\_recursive(node,data):

            if node is None:

                return False

            if node.data == data:

                return True

            if data < node.data:

                return search\_recursive(node.left,data)

            else:

                return search\_recursive(node.right,data)

        return search\_recursive(*self*.root,data)

bst = BST()

bst.insert(3)

bst.insert(5)

bst.insert(0)

bst.insert(9)

bst.inorder()

print(bst.search(9))

## Traversals:

*# While using global array or variable, always remember to clear it first before use. Else if we run code for more than one output in global array answers get appended for all test cases.*

*# We can also write this without the use of global varialbe, as explained in method 2 and 3.*

*#Method 1 ( use of global variable)*

class Solution:

    res=[]

    def inOrder(self,root):

        if root is None:

            return

*self*.inOrder(root.left)

*self*.res.append(root.val)

*self*.inOrder(root.right)

    def inorderTraversal(self, root: Optional[TreeNode]) -> List[int]:

*self*.res.clear()     *#clear the data*

*self*.inOrder(root)

        return *self*.res

*#Method 2 (without global variable)*

class Solution:

    def inorderTraversal(self, root: Optional[TreeNode]) -> List[int]:

        res = []

        if root is None:

            return res

        res+=*self*.inorderTraversal(root.left)

        res+=[root.val]

        res+=*self*.inorderTraversal(root.right)

        return res

*# Method 3 : One liner*

class Solution:

    def inorderTraversal(self, root: Optional[TreeNode]) -> List[int]:

        return  *self*.inorderTraversal(root.left) + [root.val] + *self*.inorderTraversal(root.right) if root else []

*#For Preorder*

class Solution:

    def preorder\_traversal(self, root: Optional[TreeNode]) -> List[int]:

        return [root.val]+*self*.preorder\_traversal(root.left) + *self*.preorder\_traversal(root.right) if root else []

## https://assets.leetcode.com/uploads/2021/02/19/tree1.jpgBreadth First :

Problem : so if input is given tree root = [3,9,20,null,null,15,7]

[](https://www.youtube.com/watch?v=6ZnyEApgFYg)Output should be = [[3], [9,20], [15,7]]

import collections

class Node:

    def \_\_init\_\_(self, key):

*self*.val = key

*self*.left = None

*self*.right = None

def printLevelOrder(root):

    res=[]

    q=collections.deque()

    q.append(root)

    while(q):

        q\_len=len(q)

        level=[]

        for i in range(q\_len):

            node=q.popleft()

            if node:

                level.append(node.val)

                q.append(node.left)

                q.append(node.right)

        if level:

            res.append(level)

    return res

*# Driver Program to test above function*

if \_\_name\_\_ == '\_\_main\_\_':

    root = Node(3)

    root.left = Node(9)

    root.right = Node(20)

    root.right.left = Node(15)

    root.right.right = Node(7)

    print("Level Order Traversal of binary tree is -")

    print(printLevelOrder(root))

# LEVEL 1: **EASY**

### Check if two binary trees are same or not.

Link: <https://leetcode.com/problems/same-tree/description/>

1. Given the root node of a binary search tree and two integers low and high, return the sum of values of all nodes with a value in the inclusive range [low, high].

Link: <https://leetcode.com/problems/range-sum-of-bst/description/>

### Merge two binary trees.

Link: <https://leetcode.com/problems/merge-two-binary-trees/description/>

### Convert a BST to tree with only right nodes.

Link: <https://leetcode.com/problems/increasing-order-search-tree/description/>

### Average of levels in binary tree.

Link: <https://leetcode.com/problems/average-of-levels-in-binary-tree/description/>

### Sorted array to BST

Link: <https://leetcode.com/problems/convert-sorted-array-to-binary-search-tree/description/>

### Binary Tree tilt

Link: <https://leetcode.com/problems/binary-tree-tilt/description/>

### Diameter of binary tree

Link: <https://leetcode.com/problems/binary-tree-tilt/description/>

# LEVEL 2: **Medium**

https://www.youtube.com/watch?v=80Zug6D1\_r4

# LEVEL 3: **Difficult**

# **SOLUTIONS:**

## **LEVEL 1:**

1. Same tree

class Solution:

    def isSameTree(self, p: TreeNode, q: TreeNode) -> bool:

        self.ans=True

        def helper(p,q):

            if not p and not q:

                return

            elif(p is None) or (q is None):

*self*.ans=False

                return

            else:

                helper(p.left,q.left)

                if(p.val!=q.val):

*# print("vd")*

*self*.ans=False

                helper(p.right,q.right)

        helper(p,q)

        return *self*.ans

1. Range sum of bst

[Solution](https://leetcode.com/problems/range-sum-of-bst/solutions/4558641/python-fastest-optimized-with-explanation/)

class Solution:

    def rangeSumBST(self, root: Optional[TreeNode], low: int, high: int) -> int:

        def travel(root):

            global ans

            if (root is None) :

                return

            else:

                if root.val>=low and root.val<=high:

                    ans+=root.val

                if root.val >low:

                    travel(root.left)

                if root.val <high:

                    travel(root.right)

        global ans

        ans=0

        travel(root)

        return ans

1. Pair sum divisible by 5

class Solution:

    def mergeTrees(self, root1: TreeNode, root2: TreeNode) -> TreeNode:

        if root1 is None:

            return root2

        if root2 is None:

            return  root1

        root1.val = root1.val + root2.val

        root1.left = *self*.mergeTrees(root1.left,root2.left)

        root1.right = *self*.mergeTrees(root1.right,root2.right)

        return root1

1. Convert BST to inorder tree with only right node

[Solution](https://leetcode.com/problems/increasing-order-search-tree/solutions/4558733/best-python-solutions-2-approach-optimization/)

class Solution:

    def increasingBST(self, root: TreeNode) -> TreeNode:

        def convert\_to\_inorder(root):

            global node

            if root is None:

                return

            else:

                convert\_to\_inorder(root.left)

                x=node

                while(x.right):

                    x=x.right

                x.right=TreeNode(root.val)

                convert\_to\_inorder(root.right)

        global node

        node=TreeNode()

        convert\_to\_inorder(root)

        return node.right

*#Approach 2*

class Solution:

    def increasingBST(self, root: TreeNode) -> TreeNode:

        dummy = curr =TreeNode(None)

        def dfs(root):

            if not root: return

            nonlocal curr

            dfs(root.left)

            curr.right = root

            curr = root

            curr.left = None

            dfs(root.right)

        dfs(root)

        return dummy.right

1. Average of levels in binary tree

[Solution](https://leetcode.com/problems/average-of-levels-in-binary-tree/solutions/4563843/python-code-explanation-general-approach/)

import collections

class Solution:

    def averageOfLevels(self, root: Optional[TreeNode]) -> List[float]:

        res=[]

        q = collections.deque()

        q.append(root)

        while(q):

            level=[]

            qlen =len(q)

            for i in range(qlen):

                node = q.popleft()

                if node:

                    level.append(node.val)

                    q.append(node.left)

                    q.append(node.right)

            if level:

                res.append(sum(level)/len(level))

        return res

1. Sorted Array to binary Search tree

[Solution](https://leetcode.com/problems/convert-sorted-array-to-binary-search-tree/solutions/4565640/python-code-fastest-solution-explanation/)

[](https://www.youtube.com/watch?v=0K0uCMYq5ng)

class Solution:

    def sortedArrayToBST(self, nums: List[int]) -> Optional[TreeNode]:

        def helper(l,r):

            if l>r:

                return

            else:

                m = (l+r)//2

                root = TreeNode(nums[m])

                root.left = helper(l,m-1)

                root.right = helper(m+1,r)

                return root

        return helper(0,len(nums)-1)

1. Binary tree tilt

[Solution](https://leetcode.com/problems/binary-tree-tilt/solutions/4578246/python-2-solutions-easy-one-optimized-version-with-explanation/)

class Solution:

    def findTilt(self, root: TreeNode) -> int:

        def helper(root):

            if not root: return 0

            lv, rv = helper(root.left), helper(root.right)

*self*.ans += abs(lv - rv)

            return root.val + lv + rv

*self*.ans = 0

        helper(root)

        return *self*.ans

1. Diameter Of Binary tree

[Solution](https://leetcode.com/problems/diameter-of-binary-tree/solutions/1143907/python-thought-process/)

[](https://www.youtube.com/watch?v=bkxqA8Rfv04)

def diameterOfBinaryTree(self, root):

    def recurse(node):

        if not node: return 0

        left, right = recurse(node.left), recurse(node.right)

*self*.result = max(*self*.result, left+right)

        return 1 + max(left, right)

*self*.result = 0

    recurse(root)

    return *self*.result

Similar way try to code for finding height of binary tree.

## **LEVEL 2:**