a = [16, 17, 4, 3, 5, 2]

leaders = []

for i in range(len(a)):

if all(a[i] > x for x in a[i+1:]):

leaders.append(a[i])

print("Leaders:", leaders)

def missing(arr):

n = len(arr)

s1 = (n \* (n + 1)) // 2 # Corrected the multiplication

s2 = sum(arr)

s3 = s1 - s2

return s3

arr = [1, 2, 4, 3, 5, 7, 8]

s3 = missing(arr)

print(s3)

nested = [[0, 1, 2], [3, 4, 5], [6,7, 8]]

nested[0].reverse()

nested[2].reverse()

print(nested)

for row in nested:

for element in row:

print(element, end=' ')

print()

a=[2,5,6,1,3,4,0,9,8,7,11]

for i in range(len(a)):

print(i,a[i])

a[i]=i

a = [4,5,57,3,86,26,9]

output = []

a.sort()

i = 0

j = len(a) - 1

while i < j:

output.extend([a[i], a[j]])

i += 1

j -= 1

if i == j:

output.append(a[i])

print(' '.join(map(str, output)))

#bubble sort

def bubble\_sort(arr):

n = len(arr)

for i in range(n):

for j in range(0, n - i - 1):

if arr[j] > arr[j + 1]:

arr[j], arr[j + 1] = arr[j + 1], arr[j]

# Example usage:

my\_list = [64, 34, 25, 12, 22, 11, 90]

bubble\_sort(my\_list)

print("Sorted array is:", my\_list)

#selection sort

a=[2,4,6,1]

for i in range(len(a)):

min=i

for j in range(i+1,len(a)):

if a[i]>a[j]:

min=j

a[i],a[min]=a[min],a[i]

print(a)

#order of n^3

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

for i in range (len(a)):

for j in range(i+1,len(a)):

for k in range(j+1,len(a)):

if a[i]+a[j]+a[k] == 9:

print(a[i],a[j],a[k])

#order of n^2

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

target\_sum = 9

elements = set()

for i in range(len(a)):

remaining\_sum = target\_sum - a[i]

for j in range(i + 1, len(a)):

if remaining\_sum - a[j] in elements:

print(a[i], a[j], remaining\_sum - a[j])

elements.add(a[i])

a = [1,2,3,4,5,6,7,8]

targets = 9

a.sort()

for i in range(len(a)-2):

left = i + 1

right = len(a) - 1

while left < right:

currents = a[i] + a[left] + a[right]

if currents == targets:

print(a[i], a[left], a[right])

left += 1

right -= 1

elif currents < targets:

left += 1

else:

right -= 1

a=[0,0,1,2,0,1,2,2,1]

for i in range(len(a)):

min=i

for j in range(i+1,len(a)):

if a[i]>a[j]:

min=j

a[i],a[min]=a[min],a[i]

print(a)

# Insertion sort in Python

def insertionSort(array):

for i in range(1, len(array)):

key = array[i]

j = i- 1

while j >= 0 and key < array[j]:

array[j + 1] = array[j]

j = j - 1

array[j + 1] = key

data = [0,0,1,2,0,1,2,2,1]

insertionSort(data)

print(data)

#insertion sort

data = [0, 0, 1, 2, 0, 1, 2, 2, 1]

for i in range(1, len(data)):

key = data[i]

j = i - 1

while j >= 0 and key < data[j]:

data[j + 1] = data[j]

j -= 1

data[j + 1] = key

print(data)

a = [[1, 0, 1, 1],

[1, 1, 1, 0],

[1, 0, 1, 0],

[1, 0, 0, 1]]

for i in range(len(a)):

one=a[i].count(1)

print(a[i])

a = [

[1, 0, 1, 1],

[1, 1, 1, 0],

[1, 0, 1, 0],

[1, 1, 1, 1]

]

max\_ones\_count = -1

row\_with\_max\_ones = -1

for i in range(len(a)):

ones\_count = a[i].count(1)

if ones\_count > max\_ones\_count:

max\_ones\_count = ones\_count

row\_with\_max\_ones = i

print("Row number with the highest number of ones is:", row\_with\_max\_ones)

a = [

[7, 6, 5],

[1, 2, 3],

[8, 4, 9]

]

for i in range(len(a)):

if i % 2 != 0:

for j in range(len(a[i])-1, -1, -1):

print(a[i][j])

else:

for j in range(len(a[i])):

print(a[i][j])

a = [

[7, 6, 5],

[1, 2, 3],

[8, 4, 9]

]

for i in range(len(a)):

for j in range(len(a[i])):

if (a[i][j] % 2 == 1 and j % 2 == 0) or (a[i][j] % 2 == 0 and j % 2 == 0):

print(a[i][j], end=' ')

a = [

[7, 6, 5],

[1, 2, 3],

[8, 4, 9]

]

a[0].sort()

a[2].sort()

print(a)

a = 5

b = 3

result = 0

for i in range(b):

result += a

print(result)

def multiply(a, b):

if b == 0:

return 0

elif b < 0:

return -multiply(a, -b)

else:

return a + multiply(a, b - 1)

result = multiply(5, 3)

print(result)

a = [2, 3, 1,0,4,5,7,6,9,8]

result = []

for i in range(len(a)):

for j in range(len(a)):

if i == a[j]:

result.append(str(i))

output = ', '.join(result)

print(output)

class Node:

def \_\_init\_\_(self, k):

self.key = k

self.left = None

self.right = None

def in\_order\_traversal(node):

if node:

in\_order\_traversal(node.left)

print(node.key, end=" ")

in\_order\_traversal(node.right)

# Create the binary tree

root = Node(10)

root.left = Node(8)

root.right = Node(20)

root.left.left = Node(7)

root.left.right= Node(9)

root.right.right=Node(21)

root.right.left=Node(15)

# Print the values of the nodes using in-order traversal

in\_order\_traversal(root)

# Assuming you have already created a BinarySearchTree 'bst' and inserted some keys

print(" ")

search\_key = 20 # The key you want to search for

result = bst.search(search\_key)

if result:

print(f"Found {search\_key} in the BST.")

else:

print(f"{search\_key} not found in the BST.")

class Node:

def \_\_init\_\_(self, k):

self.key = k

self.left = None

self.right = None

def in\_order\_traversal(node):

if node:

in\_order\_traversal(node.left)

print(node.key, end=" ")

in\_order\_traversal(node.right)

# Function to compute the sum of leaf nodes

def sum\_leaf\_nodes(node):

if node is None:

return 0

if node.left is None and node.right is None:

return node.key

return sum\_leaf\_nodes(node.left) + sum\_leaf\_nodes(node.right)

# Create the binary tree

root = Node(10)

root.left = Node(8)

root.right = Node(20)

root.left.left = Node(7)

root.left.right = Node(9)

root.right.right = Node(21)

root.right.left = Node(15)

# Print the values of the nodes using in-order traversal

print("Binary Tree:")

in\_order\_traversal(root)

print(" ")

# Compute the sum of leaf nodes

leaf\_sum = sum\_leaf\_nodes(root)

# Print the sum of leaf nodes

print("Sum of leaf nodes:", leaf\_sum)

a = [

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

[6, 7, 8, 9, 10],

[11, 12, 13, 14, 15]

]

result = [row[2] for row in a]

print(result)

b = sum(result)

print(b)

c = a[1]

print(c)

d = sum(c)

print(d)

e = b + d

print(e)

f=a[1][2]

print(f)

g=e-f

print(g)

class Node:

def \_\_init\_\_(self, key):

self.key = key

self.left = None

self.right = None

class BinarySearchTree:

def \_\_init\_\_(self):

self.root = None

def insert(self, key):

self.root = self.\_insert(self.root, key)

def \_insert(self, root, key):

if root is None:

return Node(key)

if key < root.key:

root.left = self.\_insert(root.left, key)

else:

root.right = self.\_insert(root.right, key)

return root

def search(self, key):

return self.\_search(self.root, key)

def \_search(self, root, key):

if root is None or root.key == key:

return root

if key < root.key:

return self.\_search(root.left, key)

return self.\_search(root.right, key)

def inorder\_traversal(self):

result = []

self.\_inorder\_traversal(self.root, result)

return result

def \_inorder\_traversal(self, root, result):

if root:

self.\_inorder\_traversal(root.left, result)

result.append(root.key)

self.\_inorder\_traversal(root.right, result)

# Example usage:

if \_\_name\_\_ == "\_\_main\_\_":

bst = BinarySearchTree()

keys = [50, 30, 70, 20, 40, 60, 80]

for key in keys:

bst.insert(key)

print("Inorder traversal of the BST:")

print(bst.inorder\_traversal())

search\_key = 40

result = bst.search(search\_key)

if result:

print(f"Found {search\_key} in the BST.")

else:

print(f"{search\_key} not found in the BST.")

def sum\_leaf\_nodes(node):

if node is None:

return 0

if node.left is None and node.right is None:

return node.key

return sum\_leaf\_nodes(node.left) + sum\_leaf\_nodes(node.right)

# Compute the sum of leaf nodes

leaf\_sum = sum\_leaf\_nodes(root)

# Print the sum of leaf nodes

print("Sum of leaf nodes:", leaf\_sum)

class Node:

def \_\_init\_\_(self, data):

self.data = data

self.left = None

self.right = None

def insert(node, data):

if node is None:

return Node(data)

if data < node.data:

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

elif data > node.data:

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

return node

if \_\_name\_\_ == '\_\_main\_\_': # Corrected '\_\_name\_\_' to '\_\_name\_\_'

root = None

root = insert(root, 1)

insert(root, 2)

insert(root, 3)

insert(root, 4)

insert(root, 5)

insert(root, 8)

insert(root, 9)

insert(root, 10)

insert(root, 11)

insert(root, 6)

insert(root, 12)

insert(root, 13)

insert(root, 7)

insert(root, 14)

insert(root, 13)

print()

class Node:

def \_\_init\_\_(self, data):

self.data = data

self.left = None

self.right = None

def insert(node, data):

if node is None:

return Node(data)

if data < node.data:

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

elif data > node.data:

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

return node

def in\_order\_traversal(node):

if node:

in\_order\_traversal(node.left)

print(node.data, end=" ")

in\_order\_traversal(node.right)

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

root = None

root = insert(root, 1)

insert(root, 3)

insert(root, 2)

insert(root, 4)

insert(root, 5)

insert(root, 7)

insert(root, 1)

insert(root, 4)

insert(root, 13)

insert(root, 12)

insert(root, 11)

insert(root, 10)

insert(root, 15)

insert(root, 14)

insert(root, 8)

insert(root, 9)

insert(root, 6)

print("In-order traversal of the tree:")

in\_order\_traversal(root)

# i want i want these output 1 3 2 4 5 6 7 15 14 13 12 11 10 9 8 not these Level-order traversal of the tree: