In [ ]:
```
DAA LABS
```

In [ ]:
```
NAME        : Guru Vamsi . P
STUDENT ID  : 22WU0106013
CLASS       : BIC - A
```

In [ ]:
```
LAB - 1
```

In [ ]:
```
1 : Leader Array
A leader in an array is an element greater than or equal to all elements to its right. It is
identified by scanning the array from right to left, updating a maximum variable as leaders
are found.
2 : Sorting Array
Given an array, the task is to sort it and create an output sequence that alternates between
the smallest and largest numbers. For example, for the array [3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5],
the output would be [1, 9, 1, 6, 2, 5, 3, 5, 4, 5, 3].
```

In [1]:
```
## Q1

## Leader Numbers : In the array, considering the right numbers, we need to check if left number is
# biggest than the right side numbers

## In a given array, we need to extract the greatest elements if the niumbers after them are smaller thamn that number

def leader(array, size):
    i = 0
    while i < size:
        j = i + 1
        while j < size:
            if array[i] <= array[j]:
                break
            j += 1
        if j == size:
            print(array[i], end=" ")
        i += 1

array = [16, 17, 4, 3, 5, 2]
result = leader(array, len(array))
```

```
17 5 2
```

In [12]:

```
# Q2:

# Input - Array
# Output - Small No. , Big No. , ....

def SmallNoBigNo(arr, n):                                 # Zig Zag Numbers
    arr.sort()                                            # using sort function to sort the array
    for i in range(1, n-1, 2):                            # traverse the array from 1 to n-1
        arr[i], arr[i+1] = arr[i+1], arr[i]               # swap value of current element with next element
    print(arr)

if __name__ == "__main__":
    arr = [4, 3, 7, 8, 6, 2, 1]
    n = len(arr)
    SmallNoBigNo(arr,n)
```

```
[1, 3, 2, 6, 4, 8, 7]
```

In [ ]:

In [ ]:

```
LAB - 2
```

In [ ]:

```
Lab – 2
1 : Triplet and their Sum
A triplet is a set of three elements in a sequence or array. Finding triplets involves identifying
combinations of three elements whose sum meets a specified criterion. (Sorting)
2 : Sorting an Array
In the given example, the array with values [0, 0, 1, 2, 0, 1, 2, 2, 1] is sorted in a way that
groups identical elements together, resulting in the output [0, 0, 0, 1, 1, 1, 2, 2, 2]. This
sorting method is often used for arrays containing a limited set of distinct values.
```

In [13]:

```
# Q1
 # Triplet Numbers and thei Sum

# Ex :

## A = [1, 2, 3, 4, 5]

# Sum needs to be 9 ; add any three no.s from the above array, their sum needs to 9

# 2 + 3 + 5 = 9
# 1 + 3 + 5 = 9

def find_triplets_with_sum(arr, target_sum):
```

```python
    n = len(arr)
    found_triplets = []

    # Sort the array for better efficiency
    arr.sort()

    for i in range(n - 2):
        left = i + 1
        right = n - 1

        while left < right:
            # whule loop is only for condition checking.... it doesnt stay till the end
            current_sum = arr[i] + arr[left] + arr[right]

            if current_sum == target_sum:
                found_triplets.append((arr[i], arr[left], arr[right]))
                left += 1
                right -= 1
            elif current_sum < target_sum:
                left += 1
            else:
                right -= 1

    return found_triplets

# Example usage:
user_sum = int(input("Enter the target sum: "))
user_array = list(map(int, input("Enter space-separated numbers in the array: ").split()))

triplets = find_triplets_with_sum(user_array, user_sum)

if triplets:
    print("Triplets with the sum", user_sum, "are:")
    for triplet in triplets:
        print(triplet)
else:
    print("No triplets found with the given sum.")
```

```
Enter the target sum: 10
Enter space-separated numbers in the array: 1 2 3 4 5
Triplets with the sum 10 are:
(1, 4, 5)
(2, 3, 5)
```

In [15]:

```
# Q2:

# Given array : [0, 0, 1, 2, 0, 1, 2, 2, 1]

# Output : [0, 0, 0, 1, 1, 1, 2, 2, 2]



# Sorting using

def custom_sort(a):
    n = len(a)       # Length of the Array

    # Traverse through all array elements
    for i in range(n):
        # Last i elements are already in place, so we don't need to check them again
        for j in range(0, n-i-1):
            # Swap if the element found is greater than the next element
            if a[j] > a[j+1]:
                a[j], a[j+1] = a[j+1], a[j]      # Here no.s are swapped

# Input array
a = [0, 0, 1, 2, 0, 1, 2, 2, 1]

# Call the custom_sort function to sort the array
custom_sort(a)

# Display the sorted array
print("Sorted Array : ", a)

# Time Complexity : n^2
```

```
Sorted Array :  [0, 0, 0, 1, 1, 1, 2, 2, 2]
```

In [ ]:

In [ ]:

```
LAB:3
```

In [ ]:

```
Lab - 3
1 : Removing Duplicate Values
2 : To find whether the linked list has a loop or NOT
3 : Time Complexity: N log N
4 : Maximum Sum
```

In [16]:

```
# Q1:
# Removing Duplicate Values
def remove_dup_values(input_list):
    output_list = []
    for item in input_list:
        if item not in output_list:
            output_list.append(item)
    return output_list

user_input = input("Enter a list of numbers separated by commas: ")
user_list = [int(x) for x in user_input.split(',')]

result_list = remove_dup_values(user_list)

print("Original List:", user_list)
print("List with Duplicates Removed:",result_list)
```

```
Enter a list of numbers separated by commas: 1,10,11,12,11,1
Original List: [1, 10, 11, 12, 11, 1]
List with Duplicates Removed: [1, 10, 11, 12]
```

In [27]:

```
# Q2:
# To find the whether  The linked list has a loop or NOT .
class ListNode:
    def __init__(self, value):
        self.value = value
        self.next = None

def has_loop(head):
    if not head or not head.next:
        return False

    slow_ptr = head
    fast_ptr = head

    while fast_ptr and fast_ptr.next:
        slow_ptr = slow_ptr.next
        fast_ptr = fast_ptr.next.next

        if slow_ptr == fast_ptr:
            return True

    return False

# Helper function to create a linked list with a loop
```

```
def create_linked_list_with_loop(values, loop_index):
    if not values:
        return None

    head = ListNode(values[0])
    current = head
    loop_node = None

    for i in range(1, len(values)):
        current.next = ListNode(values[i])
        current = current.next

        if i == loop_index:
            loop_node = current

    if loop_node:
        current.next = loop_node

    return head

# Example usage:
values = [1, 2, 3, 4, 5, 6]
loop_index = 2  # Change this value to create a loop at a different index
head = create_linked_list_with_loop(values, loop_index)

if has_loop(head):
    print("The linked list has a loop.")
else:
    print("The linked list does not have a loop.")
```

```
The linked list has a loop.
```

In [19]:

```
# Q3:

# Merge Sort

# Time Complexity : N log N

# Normal while loop where i, i, i = N
# if there is patterns like i/2 = log N
# if there is a loop i and then another loop in it i.e., iteration or is there is while loop = N log N


def merge_sort(arr, start, end):
    if start < end:
# start < end, thsi is due to if there is a single element in the array,
# starting index = ending index and then if there is a single element, there is nothing to sort in the array.
# So, to avoid that condition we make sure starting index is less than ending index
        mid = (start + end) // 2

        # Sort the first/left half
        merge_sort(arr, start, mid)

        # Sort the second half
        merge_sort(arr, mid + 1, end)

        # Merge the two sorted halves
        merge(arr, start, mid, end)

def merge(arr, start, mid, end):
    left_half = arr[start:mid + 1]
    right_half = arr[mid + 1:end + 1]

    i = j = 0
    k = start

    while i < len(left_half) and j < len(right_half):
        if left_half[i] <= right_half[j]:
            arr[k] = left_half[i]
            i += 1
        else:
            arr[k] = right_half[j]
            j += 1
        k += 1

    while i < len(left_half):
        arr[k] = left_half[i]
        i += 1
        k += 1

    while j < len(right_half):
        arr[k] = right_half[j]
        j += 1
        k += 1

# Example Usage
arr = list(map(int, input("Enter space-separated numbers in the array: ").split()))
merge_sort(arr, 0, len(arr)-1)
print(arr)
```

```
Enter space-separated numbers in the array: 1 10 4 7 8
[1, 4, 7, 8, 10]
```

In [28]:

```python
# Q4 - MAXIMUM SUM

def find_max_sum_subsets(arr):
    n = len(arr)
    max_sum = float("-inf")
    max_subsets = []

    # Generate all possible subsets of the array
    for i in range(1 << n):
        subset = [arr[j] for j in range(n) if (i & (1 << j)) > 0]

        # Calculate the sum of the current subset
        current_sum = sum(subset)

        # Check if the current sum is greater than the maximum sum
        if current_sum > max_sum:
            max_sum = current_sum
            max_subsets = [subset]
        elif current_sum == max_sum:
            max_subsets.append(subset)

    return max_sum, max_subsets

# Example usage
user_input = input("Enter the array elements separated by spaces: ")
arr = list(map(int, user_input.split()))
max_sum, max_subsets = find_max_sum_subsets(arr)

print("Maximum Sum:", max_sum)
print("Different Possible Element Sets:")
for subset in max_subsets:
 print(subset)
```

```
Enter the array elements separated by spaces: 1 2 3 4 5
Maximum Sum: 15
Different Possible Element Sets:
[1, 2, 3, 4, 5]
```

In [ ]:

In [ ]:

```
LAB - 4
```

In [ ]:

```
Lab - 4
1 : Diagonal Interchange
The task is to interchange the values of the left and right diagonals in a square matrix,
achieved by swapping the corner elements of the matrix.
2 : Index Finder
Given an array, the task is to display the indices of a specified number in the array. This
involves finding the occurrences of the number and returning their respective indices.
```

In [30]:

```
# Q1

# Interchange the Diagonal

#  0 1 2                          2 1 0
#  3 4 5    ---------------->     3 4 5
#  6 7 8                          8 7 6

# Input : [0, 4, 8] is the LEFT diagonal & [2, 4, 6] is the RIGHT diagonal
# Output : [0, 4, 8] is the RIGHT diagonal & [2, 4, 6] is the LEFT diagonal

# Trick : Swap the corners

def interchange_diagonals(matrix):
    n = len(matrix)          # len(matrix) gives the number of rows in the matrix i.e., n = 3
    for i in range(n):
        matrix[i][i], matrix[i][n-i-1] = matrix[i][n-i-1], matrix[i][i]   # Swapping the corners
        # since we are representing the no.s & matrix in the form of array, the indexing starts from 0, 1, 2...
    return matrix

original_matrix = [[0, 1, 2],
                   [3, 4, 5],
                   [6, 7, 8]]
result_matrix = interchange_diagonals(original_matrix)

for row in result_matrix:
    print(row)

## For i = 0:
# matrix[0][0], matrix[0][2] = matrix[0][2], matrix[0][0]
# This swaps the elements in the first row: matrix[0][0] = (0) & matrix[0][2] = (2).

## For i = 1:
# matrix[1][1], matrix[1][1] = matrix[1][1], matrix[1][1]
# This is effectively a no-op. It doesn't change the matrix.
```

```
##For i = 2:
# matrix[2][2], matrix[2][0] = matrix[2][0], matrix[2][2]
# This swaps the elements in the third row: matrix[2][2] (8) and matrix[2][0] (6).
```

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

In [31]:

```
# Q2
# Display the index no. in the array by finding that number in the array

# A[i] = i

def index_array(A):
    n = len(A)

    for i in range(n):
        while A[i] != i:                # != - not equal to
            temp = A[i]
            A[i], A[temp] = A[temp], A[i]
    return A

A = [2, 3, 1, 0, 4, 5, 7, 6, 9, 8]
result = index_array(A)
print(result)


def index_array(A):
    n = len(A)

    for i in range(n):
# This line starts a loop that iterates over the indices from 0 to n-1.
# In each iteration, i will take on one of these values.
        while A[i] != i:                # != - not equal to
# This line starts a while loop. It continues as long as the element at index i in the list A is not equal to i.
# This condition checks if the element at index i is in its correct position.
            temp = A[i]
# Inside the while loop, this line assigns the value of A[i] to the variable temp.
# This temporarily stores the value of the element at index i.
            A[i], A[temp] = A[temp], A[i]
# This line swaps the values at indices i and temp in the list A.
# This effectively moves the element to its correct position.
    return A

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

```
result = index_array(A)
print(result)

# In summary, this code takes a list as input, and for each element in the list,
# it swaps the element with the element at its correct index until all elements are in their correct positions.
# The modified list is then returned and printed.
```

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

In [ ]:

In [ ]:

```
# LAB - 5
```

In [ ]:

```
Lab - 5
1 :ROWS With Most No of One's
2 :Middle Row and Column Sum
The task is to calculate the sum of the middle row and middle column in a matrix, involving
finding the central elements and adding up their respective rows and columns. This problem
is relevant in array manipulation and matrix operations.
```

In [35]:

```
#Q1
# ROWS WITH MOST NO OF ONE'S

def count_ones(row):
    return row.count(1)

def find_max_ones(matrix):
    max_ones = 0
    max_row = -1

    for i, row in enumerate(matrix):
        ones_count = count_ones(row)
        if ones_count > max_ones:
            max_ones = ones_count
            max_row = i

    return max_row

matrix = []
print("Please enter a 4x4 matrix with only 1s and 0s (with spaces between each no.s):")
for _ in range(4):
    row = list(map(int, input().split()))
    matrix.append(row)
```

```python
max_row = find_max_ones(matrix)

if max_row != -1:
    print(f"The row with the highest number of 1s is Row = {max_row+1}")
    print(f"Number of 1s: {matrix[max_row].count(1)}")
else:
    print("No row contains any 1s.")
```

```
Please enter a 4x4 matrix with only 1s and 0s (with spaces between each no.s):
1 1 1 1
2 2 1 1
4 1 3 2
0 0 0 9
The row with the highest number of 1s is Row = 1
Number of 1s: 4
```

In [37]:

```python
# Q2:
# SUM OF MIDDLE ROW AND MIDDLE COLUMN
def sum_middle_row_and_column(matrix):
    rows = len(matrix)
    cols = len(matrix[0])

    middle_row = rows // 2
    middle_col = cols // 2

    middle_value = matrix[middle_row][middle_col]

    row_sum = sum(matrix[middle_row])
    col_sum = sum(row[middle_col] for row in matrix)
    total_sum = row_sum + col_sum - middle_value
    return total_sum

matrix = [
    [1, 2, 3, 4, 5],
    [6, 7, 8, 9, 10],
    [11, 12, 13, 14, 15]
]

result = sum_middle_row_and_column(matrix)
print("Sum of middle row and middle column values (excluding middle value):",result)
```

```
Sum of middle row and middle column values (excluding middle value): 56
```

In [ ]:

```
```

In [ ]:

```
LAB - 6
```

In [ ]:

```
Lab – 6
1 : Matrix Sorting and Diagonal Replacement
The task is to sort a matrix without using any Python built-in libraries and subsequently
replace both left and right diagonals with zeros. This problem involves custom sorting
algorithms and matrix manipulation.
2 : Integer Multiplication without Operators
```

In [38]:

```
# Q1: - SORTING THE MATRIX AND REPLACING THE DIAGONALS WITH 0'S.

# Sort the matrix without any inbuilt libraries of python

# After that, replace the left and right diagonals with 0's


def get_matrix_from_user():
    rows = int(input("Enter the number of rows: "))
    cols = int(input("Enter the number of columns: "))

    matrix = []
    for i in range(rows):
        row = []
        for j in range(cols):
            element = int(input(f"Enter element at position ({i+1}, {j+1}): "))
            row.append(element)
        matrix.append(row)

    return matrix

def print_matrix(matrix):
    for row in matrix:
        print(' '.join(map(str, row)))

def sort_matrix(matrix):
    flattened_matrix = [item for sublist in matrix for item in sublist]
    flattened_matrix.sort()
    sorted_matrix = [flattened_matrix[i:i+len(matrix[0])] for i in range(0, len(flattened_matrix), len(matrix[0]))]

    return sorted_matrix
```

```
def replace_diagonals(matrix):
    size = len(matrix)
    for i in range(size):
        matrix[i][i] = 0
        matrix[i][size - i - 1] = 0

    return matrix

# Get the matrix from the user
matrix = get_matrix_from_user()

# Sort the matrix
sorted_matrix = sort_matrix(matrix)

# Print the sorted matrix
print("Sorted Matrix:")
print_matrix(sorted_matrix)

# Replace diagonals with 0's
modified_matrix = replace_diagonals(sorted_matrix)

# Print the modified matrix
print("\nMatrix with Diagonals Replaced:")
print_matrix(modified_matrix)
```

```
Enter the number of rows: 3
Enter the number of columns: 3
Enter element at position (1, 1): 7
Enter element at position (1, 2): 6
Enter element at position (1, 3): 5
Enter element at position (2, 1): 1
Enter element at position (2, 2): 2
Enter element at position (2, 3): 3
Enter element at position (3, 1): 8
Enter element at position (3, 2): 4
Enter element at position (3, 3): 9
Sorted Matrix:
1 2 3
4 5 6
7 8 9

Matrix with Diagonals Replaced:
0 2 0
4 0 6
0 8 0
```

In [40]:

```
# Q2:

# Multiply 2 integers

# DO NOT USE multiplication, division, for loops, bitwise operators

# Can be done using Recursion


def multiplication(a, b):
    if b == 0:
        return 0
    return a + multiplication(a, b - 1)   # Recursion - funtion calling itself until the values are decremented to 0
# Recursion is happening in the above line & 'multiplication(a, b-1)' causes recursion and until b = 0, recursion happe
# This is resultion in 'repeated addition' which is the same as 'multiplication'

a = int(input("Enter 1st Integer : "))
b = int(input("Enter 2nd Integer : "))

result = multiplication(a,b)
print("Product: ",result)
```

```
Enter 1st Integer : 2
Enter 2nd Integer : 3
Product:  6
```

In [ ]:

```
# LAB 7
```

In [ ]:

```
Lab - 7
1 : Binary Search Tree Node Search
2 : Leaf Node Sum in a Tree
3 : Spinal (Zig-Zag) Order Tree Traversal
The task involves printing the nodes of a binary tree in a spiral (zigzag) manner, alternating
between left-to-right and right-to-left traversal at each level. This problem explores tree
traversal techniques.
```

In [ ]:

In [2]:

```
# Q1
# SEARCH THE NODE BST

class Node:
    def __init__(self, value):
        self.value = value
        self.left = None
        self.right = None

def insert_node(root, value):
    if root is None:
        return Node(value)
    else:
        if root.value < value:
            root.right = insert_node(root.right, value)
        else:
            root.left = insert_node(root.left, value)
    return root

def search_node(root, value):
    if root is None or root.value == value:
        return root is not None

    if root.value < value:
        return search_node(root.right, value)

    return search_node(root.left, value)

# Creating the BST with the provided nodes: 10, 8, 20, 9, 7, 21, 15
root = None
nodes = [10, 8, 20, 9, 7, 21, 15]
for node in nodes:
    root = insert_node(root, node)

# Taking user input for the node to search
user_input = int(input("Enter the value to search: "))

# Searching for the user input node
result = search_node(root, user_input)

# Printing the result
print(result)
```

```
Enter the value to search: 21
True
```

In [ ]:

In [6]:

```
# Q2
# ADD ALL THE LEAF NODES
```

```
class Node:

    def __init__(self, key):
        self.key = key
        self.left = None
        self.right = None

def insert(node, key):
    if node is None:
        return Node(key)

    if key < node.key:
        node.left = insert(node.left, key)
    elif key > node.key:
        node.right = insert(node.right, key)

    return node

def search(root, key):
    if root is None or root.key == key:
        return root

    if root.key < key:
        return search(root.right, key)

    return search(root.left, key)

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)

if __name__ == '__main__':
    root = None
    root = insert(root, 50)
    insert(root, 10)
    insert(root, 8)
    insert(root, 7)         # Leaf Node
    insert(root, 9)         # Leaf Node
    insert(root, 20)
    insert(root, 15)        # Leaf Node
    insert(root, 21)        # Leaf Node

    key = int(input("Enter the node to be searched: "))

    if search(root, key) is None:
        print(key, "not found")
    else:
        print(key, "found")

    sum_leaf = sum_leaf_nodes(root)
    print("Sum of leaf nodes:", sum_leaf)
```

```
Enter the node to be searched: 21
21 found
Sum of leaf nodes: 52
```

In [7]:

```
# PRINTING THE BINARY TREE NODES IN A SPINAL MANNER

class TreeNode:
    def __init__(self, val):
        self.val = val
        self.left = None
        self.right = None

def build_tree(nums):
    if not nums:
        return None

    root = TreeNode(nums.pop(0))
    queue = [root]

    while queue and nums:
        node = queue.pop(0)

        left_val = nums.pop(0)
        if left_val is not None:
            node.left = TreeNode(left_val)
            queue.append(node.left)

        if nums:
            right_val = nums.pop(0)
            if right_val is not None:
                node.right = TreeNode(right_val)
                queue.append(node.right)
    return root

def spiral_traversal(root):
    if not root:
        return []

    result = []
    level = 1
    queue = [root]

    while queue:
        level_size = len(queue)
        level_nodes = []

        for _ in range(level_size):
            node = queue.pop(0)

            if level % 2 == 1:
                level_nodes.append(node.val)
            else:
                level_nodes.insert(0, node.val)

            if node.left:
                queue.append(node.left)
            if node.right:
                queue.append(node.right)

        result.extend(level_nodes)
        level += 1

    return result

# Input list
input_list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]

# Build the binary tree
root = build_tree(input_list)

# Perform spiral traversal
output = spiral_traversal(root)

# Print the result
print(output)
```

```
[1, 3, 2, 4, 5, 6, 7, 15, 14, 13, 12, 11, 10, 9, 8]
```

In [ ]: