|  |  |  |
| --- | --- | --- |
| NAME | : | Neha vuppala |
| STUDENT ID | : | 22WU0101124 |
| CLASS**/**SECTION | : | B**.**Tech CSE-b |
|  |  |  |
| *## LAB - 1 ##* |  |  |

In [ ]:

In [ ]:

|  |
| --- |
| *## Q1 - LEADING NUMBERS*  *## Leader Numbers : In the array, considering the right numbers, we need to check i*  *# biggest than the right side numbers ## In a given array, we need to extract the greatest elements if the niumbers after*  **def** leader\_numbers(arr):  n **=** len(arr) *# len(arr) givens the no. of elements of the a*  leaders **=** [ ]  max\_right **=** arr[n **-** 1] *# Displays (n-1)th index of the array since in*    leaders**.**append(max\_right) *# Rightmost or last element of the array is th*  *# because there are no numbers after it to com*  **for** i **in** range(n **-** 2, **-**1, **-**1) :  **if** arr[i] **>** max\_right: max\_right **=** arr[i] leaders**.**append(max\_right)    **return** leaders[::**-**1]    sample **=** [16, 17, 4, 3, 5, 2] result **=** leader\_numbers(sample) print("Leader Numbers : ", result) |

In [1]:

|  |
| --- |
| *## Q2 - SORTING IN SMALL NUMBER & BIG NUMBER MANNER ## Sorting in Small Number & Big Number Manner*  *# Input - Array*  *# Output - Small No. , Big No. , .....*  **def** SmallNoBigNo(arr, n): *# Zig Zag Numbers*  arr**.**sort() *# using sort function to*  **for** i **in** range(1, n**-**1, 2): *# traverse the array from*  arr[i], arr[i**+**1] **=** arr[i**+**1], arr[i] *# swap value of current e*  print(arr)  **if** \_\_name\_\_ **==** "\_\_main\_\_":  arr **=** [4, 3, 7, 8, 6, 2, 1]  n **=** len(arr)  SmallNoBigNo(arr, n) |

Leader Numbers : [17, 5, 2] In [2]:

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

In [ ]: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

In [ ]: *## LAB - 2 ##*

In [3]: *## Q1 - SUM & THEIR SET OF TRIPLET NUMBERS*

*# Triplet Numbers and their 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*  *# 1 Loop # Time Complexity = n (since only 1 loop)*  **def** find\_triplets\_with\_sum(arr, target\_sum):  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*  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: ")**.**  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.") |

sp

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)

|  |
| --- |
| *## Q2 - DEFAULT SORT WITHOUT ANY INSERTING ALGORITHM*  *# Given array : [0, 0, 1, 2, 0, 1, 2, 2, 1]*  *# Output : [0, 0, 0, 1, 1, 1, 2, 2, 2]* |

(2, 3, 5) In [4]:

|  |
| --- |
| *# Sorting using # Sorting using*  **def** default\_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 agai*  **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* default\_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]

|  |
| --- |
| *## 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) |

In [ ]: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ In [ ]: *## LAB - 3 ##* In [5]:

Enter a list of numbers separated by commas: 1,10,11,12,11,1

Original List: [1, 10, 11, 12, 11, 1]

|  |
| --- |
| *## Q2 - GIVEN 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** |

List with Duplicates Removed: [1, 10, 11, 12] In [6]:

|  |
| --- |
| 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.") |

|  |
| --- |
| *## Q3 - MERGE SORT*  *# 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 w*  **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 not # So, to avoid that condition we make sure starting index is less than ending index*  mid **=** (start **+** end) **//** 2  *# Sort the first/left half* |

The linked list has a loop. In [7]:

|  |
| --- |
| 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

|  |
| --- |
| *## 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 |

[1, 4, 7, 8, 10] In [8]:

|  |
| --- |
| *# 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]

|  |
| --- |
| *## Q1 - INTERCHANGING DIAGONALS*  *# 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*  **for** i **in** range(n):  matrix[i][i], matrix[i][n**-**i**-**1] **=** matrix[i][n**-**i**-**1], matrix[i][i] *# Swappin*  *# since we are representing the no.s & matrix in the form of array, the ind*  **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) |

In [ ]: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ In [ ]: *## LAB - 4 ##* In [3]:

[2, 1, 0]

[3, 4, 5]

|  |
| --- |
| *## Q2 - INDEX ARRAY # 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] |

[8, 7, 6] In [2]:

|  |
| --- |
| 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) |

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

|  |
| --- |
| *## Q1 - ROWS WITH MOST NO. OF 1'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) 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.") |

In [ ]: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ In [ ]: *## LAB - 5 ##* In [4]:

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

|  |
| --- |
| *## Q2 - SUM OF MIDDLE ROW & 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]) |

Number of 1s: 4 In [5]:

|  |
| --- |
| 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):", resul |

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

|  |
| --- |
| *## LAB - 6 ##* |

In [ ]: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ In [ ]:

|  |
| --- |
| *## Q1 - SORTING A MATRIX & 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(fla **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* |

In [6]:

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

|  |
| --- |
| *## Q2 - MULTIPLICATION OF NUMBERS*  *# 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*  a **=** int(input("Enter 1st Integer : ")) b **=** int(input("Enter 2nd Integer : "))  result **=** multiplication(a,b) print("Product: ", result) |

0 8 0 In [7]:

Enter 1st Integer : 2

Enter 2nd Integer : 3

Product: 6

In [ ]: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ In [ ]: *## LAB - 7 ##*

|  |
| --- |
| *## Q1 - SEARCH THE NODE IN BST*  *# Search the User Input Node from the Binary Search Tree*  *# If the node is present, return True*  *# If not there, return False*  **class** Node: **def** \_\_init\_\_(self, value): |

In [1]:

|  |
| --- |
| 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: 8

|  |
| --- |
| *## 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 |

True In [5]:

|  |
| --- |
| **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 : 8

8 FOUND

|  |
| --- |
| *## Q3 - PRINTING THE BINARY TREE NODES IN A SPIRAL 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: |

Sum of Leaf Nodes : 52 In [10]:

|  |
| --- |
| 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 [ ]: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_