**Department of Artificial Intelligence, QUEST Nawabshah**

DATA STRUCTURES

**LAB # 01 Introduction to Data Structures**

1. Introduction to Data Structures

**1.1 OBJECTIVES**

By the end of this lab session, you should be able to:

Define data structures and their importance in programming.

Understand the different types of data structures: linear and non-linear.

Identify the applications of various data structures in solving real-world problems.

Analyze the time complexity of basic operations on different data structures.

Implement basic data types like arrays and linked lists in a chosen programming language.

**1.2 THEORY**

**1.2.1 Definition of Data Structures**

A data structure is a way of organizing data in a computer's memory so that it can be accessed and processed efficiently. It provides a way to store, access, and manipulate data in a structured manner. Data structures can be classified into two main categories:

Linear data structures: These data structures follow a sequential order, where each element is connected to its predecessor and successor. Examples include arrays, linked lists, queues, and stacks.

Non-linear data structures: These data structures do not follow a sequential order. They have a hierarchical or non-linear relationship between elements. Examples include trees and graphs.

**1.2.2 Types of Data Structures:**

**Linear Data Structures**:

**Arrays**: Arrays are fixed-size collections of elements of the same type. They store elements contiguously in memory, allowing for fast random access.

**Linked Lists**: Linked lists are dynamic data structures where elements are linked together using pointers. They offer efficient insertion and deletion operations but require more memory overhead than arrays.

**Queues**: Queues are linear data structures that follow the FIFO (First In First Out) principle. New elements are inserted at the back and removed from the front.

**Stacks**: Stacks are linear data structures that follow the LIFO (Last In First Out) principle. New elements are pushed onto the top and removed from the top.

**Non-Linear Data Structures**:

**Trees**: Trees are hierarchical data structures where nodes contain data and references to child nodes. They are used for representing hierarchical relationships and efficient searching and sorting.

**Graphs**: Graphs are collections of nodes (vertices) connected by edges. They are used to represent relationships between entities and solve problems involving networks and maps.

**1.2.3 Applications of Data Structures**

Data structures play a crucial role in various areas of computer science and software development, including:

**Algorithms**: Efficient algorithms often rely on specific data structures to achieve optimal performance.

**Databases**: Databases store and manage large amounts of data using various data structures like B-trees and hash tables.

**Compilers and Operating Systems**: Compilers use data structures for symbol tables and managing code generation. Operating systems use data structures for memory management, process scheduling, and file systems.

**Graphics and Multimedia**: Graphics and multimedia applications use data structures like trees and graphs to represent images, animations, and other media content.

**Networking**: Routing algorithms and data packet management.

**1.3 EXERCISE**

**1.3.1 Analyzing Time Complexity**

**Analyze the time complexity of accessing an element in an array at a specific index.**

Accessing an element in an array at a specific index has a time complexity of : O(1)

**Analyze the time complexity of searching for an element in an unsorted array.**

Searching for an element in an unsorted array has a time complexity of : O(n).

**Analy**z**e the time complexity of adding an element to the beginning and end of a linked list.**

Adding an element to the beginning of a linked list has a time complexity of : O(1)

Adding an element to the end of a linked list has a time complexity of : O(1) (average

case) or O(n) (worst case)

**Analyze the time complexity of searching for an element in a linked list.**

Searching for an element in a linked list has a time complexity of : O (n)

where n is the number of elements in the list.

**1.3.2 Implementing Basic Data Types**

**Implement an array in your chosen programming language and demonstrate its basic operations like accessing, adding, and deleting elements.**

arr = []

# Adding elements to the array

arr.append(1)

arr.append(2)

arr.append(3)

# Accessing elements at specific indices

element\_at\_index\_1 = arr[1]

print("Element at index 1:", element\_at\_index\_1)

# Displaying the array

print("Array:", arr)

# Deleting element at index 0

del arr[0]

print("Element at index 0 deleted.")

# Displaying the updated array

print("Updated Array:", arr)

**Implement a linked list in your chosen programming language and demonstrate its basic operations like adding, deleting, and traversing elements.**

class Node:

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

self.data = data

self.next = None

class LinkedList:

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

self.head = None

# Add a new node at the beginning

def insert\_at\_head(self, new\_node):

new\_node.next = self.head # Connect the new node to the current head

self.head = new\_node # Make the new node the head

# Delete a node with a specific data

def delete\_node(self, data):

current\_node = self.head

previous\_node = None

while current\_node:

if current\_node.data == data: # Node found

if previous\_node:

previous\_node.next = current\_node.next # Skip over the node

else:

self.head = current\_node.next # Update head if deleting the first node

return

previous\_node = current\_node

current\_node = current\_node.next

# Print the linked list data

def print\_list(self):

current\_node = self.head

while current\_node:

print(current\_node.data, end=" -> ")

current\_node = current\_node.next

print("None") # Indicate the end of the list

# Example usage

first\_node = Node("Ahmad")

linked\_list = LinkedList()

linked\_list.insert\_at\_head(first\_node)

second\_node = Node("Amaan")

linked\_list.insert\_at\_head(second\_node)

third\_node = Node("Sami")

linked\_list.insert\_at\_head(third\_node)

linked\_list.print\_list() # Output: sami -> Amaan -> Ahmad -> None

# Delete "Amaan"

linked\_list.delete\_node("Amaan")

linked\_list.print\_list() # Output: sami -> Ahmad -> None

**1.4 REFERENCES**

Goodrich, Michael T., Roberto Tamassia, and Goldwasser Michael. Data Structures and Algorithms in Python. John Wiley & Sons, 2013.

Cormen, Thomas H., et al. Introduction to Algorithms. MIT press, 2009.

Weiss, Mark Allen. Data Structures and Algorithm Analysis in C++ (4th Edition). Addison-Wesley Professional, 2013.