Assignment 1 - Linked List:

Problem a:

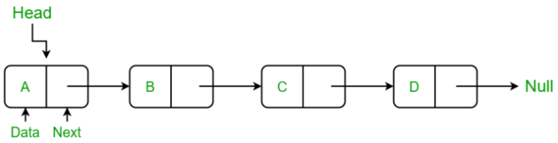
- Implement a singly linked list in Python and perform basic operations: insert at the beginning, end, and in the middle, and delete a node.

```
class Node:

    def __init__(self, data):

        self.data = data

        self.next = None
```



```
class SinglyLinkedList:

    def __init__(self):

        self.head = None



    def insert_at_beginning(self, data):

        new_node = Node(data)

        new_node.next = self.head

        self.head = new_node



    def insert_at_end(self, data):
        new_node = Node(data)
        if not self.head:
            self.head = new_node
            return
        current = self.head
        while current.next:
            current = current.next
        current.next = new_node

    def insert_in_middle(self, previous_node_data, data):
        new_node = Node(data)
        current = self.head
        while current:
            if current.data == previous_node_data:
                new_node.next = current.next
                current.next = new_node
                return
            current = current.next

    def delete_node(self, key):
        current = self.head
        if current and current.data == key:
            self.head = current.next
            return
        prev = None
        while current and current.data != key:
            prev = current
            current = current.next
        if not current:
            return
        prev.next = current.next

    def display(self):
        current = self.head
        while current:
            print(current.data, end=" -> ")
            current = current.next
        print("None")

# Example Usage
linked_list = SinglyLinkedList()
linked_list.insert_at_end(1)
linked_list.insert_at_end(2)

linked_list.insert_at_end(4)



linked_list.insert_at_beginning(0)

linked_list.insert_in_middle(2, 3)

linked_list.delete_node(2)



linked_list.display()
```

Problem b:

- Extend the linked list implementation to a doubly linked list and demonstrate its advantages over a singly linked list.

Problem c:

- Implement a circular linked list and use it to solve a specific problem where circular traversal is beneficial.

Problem d:

- Implement a function to detect if a linked list has a cycle and, if so, determine the length of the cycle.

Problem e:

- Create a program to reverse a linked list, considering both iterative and recursive approaches.

Assignment 2 - Stack and Queue:

Problem a:

- Implement a singly linked list in Python and perform basic operations: insert at the beginning, end, and in the middle, and delete a node

Problem b:

- Implement a queue using two stacks and analyze its time complexity for enqueue and dequeue operations.

Problem c:

- Solve a problem using a stack, like checking for balanced parentheses in an expression.

Problem d:

- Implement a circular queue using an array and perform basic operations: enqueue and dequeue.

Problem e:

- Design a priority queue using heaps and use it for sorting elements.

Assignment 3 - Binary Search Tree:

Problem a:

- Implement a binary search tree in Python and write a program to find the kth smallest and kth largest elements.

Problem b:

- Modify the binary search tree to balance it using AVL rotations.

Problem c:

- Write a function to check if a binary tree is a binary search tree.

Problem d:

- Implement a program to find the lowest common ancestor in a binary search tree.

Problem e:

- Build a binary search tree from a sorted array and demonstrate its construction.

Assignment 4 - Hash Table:

Problem a:

- Create a simple hash table using arrays and implement basic operations: insertion, deletion, and search.

Problem b:

- Implement a hash map with collision handling using techniques like chaining or open addressing.

Problem c:

- Use a hash table to solve a problem, such as finding duplicates in an array or implementing a frequency counter.

Problem d:

- Discuss and implement different hash functions and their impact on the performance of a hash table.

Problem e:

- Implement a hash table that dynamically resizes to maintain a low load factor and analyze its benefits.