DSA Assignment 1

Linked List
• Implement the following functions for a singly linked list:
o Inserts an element at the specified index.
o Deletes the element at the specified index.
o Returns the size of the linked list.
o Returns true if the linked list is empty, false otherwise.
o Rotates the linked list by k positions to the right.
o Reverses the linked list.
o Appends an element to the end of the linked list.
o Prepends an element to the beginning of the linked list.
o Merges two linked lists into a single linked list.
o Interleaves two linked lists into a single linked list.
o Returns the middle element of the linked list.
o Returns the index of the first occurrence of the specified element in the linked list,
or -1 if the element is not found.
o Splits the linked list into two linked lists at the specified index.

```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

class SinglyLinkedList:
    def __init__(self):
        self.head = None

    def __insert_at_index(self, index, value):
        new_node = Node(value)
        if index == 0:
            new_node.next = self.head
```

```
            self.head = new_node
            return

        current = self.head
        for _ in range(index - 1):
            if current is None:
                raise IndexError("Index out of bounds")
            current = current.next

        new_node.next = current.next
        current.next = new_node


    def __delete_at_index(self,index):
        if index == 0:
            self.head = self.head.next
            return
        current = self.head
        for _ in range(index - 1):
            if current is None or current.next is None:
                raise IndexError("Index out of bounds")
            current = current.next
        current.next = current.next.next


    def return_size(self):
        count, current = 0, self.head
        while current:
            count +=1
            current = current.next
        return count



    def empty(self):
        return self.head is None
```

```
    def rotate(self, k):
        if not self.head or k <= 0:
            return
        length = self.size()
        k %= length
        if k == 0:
            return

        fast, slow = self.head, self.head
        for _ in range(k):
            fast = fast.next

        while fast.next:
            fast = fast.next
            slow = slow.next

        new_head = slow.next
        slow.next = None
        fast.next = self.head
        self.head = new_head



    def reverse(self):
        prev, current = None, self.head
        while current:
            current.next, prev, current = prev, current, current
        self.head = prev


    def append(self, value):
        new_node = Node(value)
        if not self.head:
            self.head = new_node
```

```
            return
        current = self.head
        while current.next:
            current = current.next
        current.next = new_node


    def prepend(self, value):
        new_node = Node(value)
        new_node.next = self.head
        self.head = new_node


    def merge(self, other_list):
        if not self.head:
            self.head = other_list.head
            return
        current = self.head
        while current.next:
            current = current.next
        current.next = other_list.head


    def interleave(self, other_list):
        temp = Node()
        tail = temp
        curr1, curr2 = self.head, other_list.head

        while curr1 or curr2:
            if curr1:
                tail.next = curr1
                tail = tail.next
                curr1 = curr1.next
            if curr2:
                tail.next = curr2
                tail = tail.next
```

```
                curr2 = curr2.next

        self.head = temp.next



    def middle(self):
        slow, fast = self.head, self.head
        while fast and fast.next:
            slow = slow.next
            fast = fast.next.next
        return slow.value if slow else None


    def index_of(self, value):
        current, index = self.head, 0
        while current:
            if current.value == value:
                return index
            current = current.next
            index += 1
        return -1



    def split_at_index(self, index):
        if index < 0:
            raise IndexError("Index out of bounds")
        new_list = SinglyLinkedList()
        if index == 0:
            new_list.head = self.head
            self.head = None
            return new_list
        current = self.head
        for _ in range(index - 1):
            if not current:
```

```
                raise IndexError("Index out of bounds")
            current = current.next
        new_list.head = current.next
        current.next = None
        return new_list
```

Dynamic Arrays

• Implement the following methods for a dynamic array:

o Inserts an element at the specified index.

o Deletes the element at the specified index.

o Returns the size of the dynamic array.

o Returns true if the dynamic array is empty, false otherwise.

o Rotates the dynamic array by k positions to the right.

o Reverses the dynamic array.

o Appends an element to the end of the dynamic array.

o Prepends an element to the beginning of the dynamic array.

o Merges two dynamic arrays into a single dynamic array.

o Interleaves two dynamic arrays into a single dynamic array.

o Returns the middle element of the dynamic array.

o Returns the index of the first occurrence of the specified element in the dynamic array, or -1 if the element is not found.

o Splits the dynamic array into two dynamic arrays at the specified index.

o Resizing the arrays with custom factor given by user (other than 2)

```
class dynamicArray:
    def __init__(self, resize_factor=2):
        self.array = []
        self.size = 0
        self.resize_factor = resize_factor

    def insert(self, index, element):
        if index < 0 or index > self.size:
```

```
            raise IndexError("Index out of bound")
        self.array.insert(index, element)
        self.size += 1

    def delete(self, index):
        if index < 0 or index >= self.size:
            raise IndexError("Index out of bound")
        del self.array[index]
        self.size -= 1

    def get_size(self):
        return self.size

    def is_empty(self):
        return self.size == 0

    def rotate(self, k):
        if self.size == 0 or k == 0:
            return
        k = k % self.size
        self.array = self.array[-k:] + self.array[:-k]

    def reverse(self):
        self.array.reverse()

    def append(self, ele):
        self.array.append(ele)
        self.size += 1

    def prepend(self, ele):
        self.array.insert(0, ele)
        self.size += 1

    def merge(self, other_array):
        self.array.extend(other_array.array)
        self.size += other_array.size
```

```
    def interleave(self, other_array):
        res = []
        i, j = 0, 0
        while i < self.size or j < other_array.size:
            if i < self.size:
                res.append(self.array[i])
                i += 1
            if j < other_array.size:
                res.append(other_array.array[j])
                j += 1
        self.array = res
        self.size = len(res)

    def find_middle(self):
        if self.size == 0:
            return None
        return self.array[self.size // 2]

    def index_of(self, ele):
        for i in range(self.size):
            if self.array[i] == ele:
                return i
        return -1

    def split(self, index):
        if index < 0 or index > self.size:
            raise IndexError("Index out of bound")
        new_array = dynamicArray(self.resize_factor)
        new_array.array = self.array[index:]
        new_array.size = len(new_array.array)
        self.array = self.array[:index]
        self.size = len(self.array)
        return new_array
```

```
    def resize(self, new_resize_factor):
        self.resize_factor = new_resize_factor

    def __resize_up(self):
        new_capacity = int(len(self.array) * self.resize_factor
        new_array = [None] * new_capacity
        for i in range(self.size):
            new_array[i] = self.array[i]
        self.array = new_array

    def __resize_down(self):
        new_capacity = max(1, int(len(self.array) / self.resize_
        new_array = [None] * new_capacity
        for i in range(self.size):
            new_array[i] = self.array[i]
        self.array = new_array
```

Comparison of Linked Lists and Dynamic Arrays

1. Time Complexity of Each Method

Operation	Singly Linked List	Dynamic Array
Insert at index	O(n)	O(n)
Delete at index	O(n)	O(n)
Get size	O(1)	O(1)
Is empty	O(1)	O(1)
Rotate right by k	O(n)	O(n)
Reverse	O(n)	O(n)A
Append	O(1)	O(1)
Prepend	O(1)	O(n)
Merge	O(1)	O(n)
Interleave	O(n)	O(n)

Find Middle	O(n)	O(1)
Index of element	O(n)	O(n)
Split at index	O(n)	O(n)
Resize	—	O(n)

2. Space Complexity of Each Method

Operation	Singly Linked List	Dynamic Array
Insert at index	O(1)	O(1)
Delete at index	O(1)	O(1)
Get size	O(1)	O(1)
Is empty	O(1)	O(1)
Rotate right by k	O(1)	O(1)
Reverse	O(1)	O(1)
Append	O(1)	O(1)
Prepend	O(1)	O(n)
Merge	O(1)	O(n)
Interleave	O(1)	O(n)
Find middle	O(1)	O(1)
Index of element	O(1)	O(1)
Split at index	O(1)	O(n)
Resize (custom factor)	—	O(n)

3. Advantages and Disadvantages

Linked List

Advantages:

1. Dynamic Size

Disadvantages:

1. Slow Access

2. Efficient Insertions/Deletions
3. No wasted space

2. Memory Overhead
3. Cache Performance

Dynamic Arrays

Advantages:

1. Fast Access
2. Efficient Iteration
3. Memory Efficiency

Disadvantages:

1. Resize Overheard
2. Insert/Delete Costs
3. Pre-allocated space