**C-DAC Mumbai Date 01/10/2024**

**Subject: Algorithm and Data Structure**

**Assignment 3**

**Solve the assignment with following thing to be added in each question.**

-Program

-Flow chart

-Explanation

-Output

-Time and Space complexity

Submission Date: 3/10/2024

**1. Implement a singly linked list with basic operations: insert, delete, search.**

* **Test Case 1**:  
  Input: Insert 3 → Insert 7 → Insert 5 → Delete 7 → Search 5  
  Output: List = [3, 5], Found = True
* **Test Case 2**:  
  Input: Insert 9 → Insert 4 → Delete 4 → Search 10  
  Output: List = [9], Found = False

**Program:**

class Node {

int data;

Node next;

public Node(int data) {

this.data = data;

this.next = null;

}

}

class Ass4prg1 {

private Node head;

public void insert(int data) {

Node newNode = new Node(data);

if (head == null) {

head = newNode;

System.out.println("Inserted: " + data);

return;

}

Node current = head;

while (current.next != null) {

current = current.next;

}

current.next = newNode;

System.out.println("Inserted: " + data);

}

public void delete(int data) {

if (head == null) return;

if (head.data == data) {

head = head.next;

System.out.println("Deleted: " + data);

return;

}

Node current = head;

while (current.next != null && current.next.data != data) {

current = current.next;

}

if (current.next != null) {

current.next = current.next.next;

System.out.println("Deleted: " + data);

}

}

public boolean search(int data) {

Node current = head;

while (current != null) {

if (current.data == data) {

return true;

}

current = current.next;

}

return false;

}

public void display() {

Node current = head;

System.out.print("List = [");

while (current != null) {

System.out.print(current.data + (current.next != null ? ", " : ""));

current = current.next;

}

System.out.println("]");

}

public static void main(String[] args) {

Ass4prg1 linkedList1 = new Ass4prg1();

// Test Case 1

System.out.println("Test Case 1:");

linkedList1.insert(3);

linkedList1.insert(7);

linkedList1.insert(5);

linkedList1.delete(7);

boolean found = linkedList1.search(5);

linkedList1.display();

System.out.println("Found = " + found);

// Test Case 2

System.out.println("\nTest Case 2:");

Ass4prg1 linkedList2 = new Ass4prg1();

linkedList2.insert(9);

linkedList2.insert(4);

linkedList2.delete(4);

found = linkedList2.search(10);

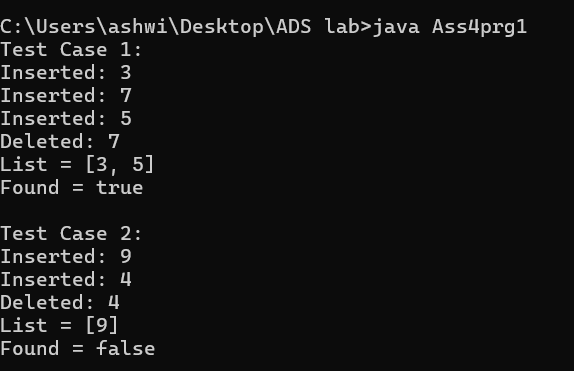
linkedList2.display();

System.out.println("Found = " + found);

}

}

**Output:**



**2. Reverse a singly linked list.**

* **Test Case 1**:  
  Input: List = [1, 2, 3, 4, 5]  
  Output: List = [5, 4, 3, 2, 1]
* **Test Case 2**:  
  Input: List = [10, 20, 30]  
  Output: List = [30, 20, 10]

**Program:**

class Node {

int data;

Node next;

public Node(int data) {

this.data = data;

this.next = null;

}

}

class Ass4prg2 {

private Node head;

public void insert(int data) {

Node newNode = new Node(data);

if (head == null) {

head = newNode;

return;

}

Node current = head;

while (current.next != null) {

current = current.next;

}

current.next = newNode;

}

public void reverse() {

Node prev = null;

Node current = head;

Node next = null;

while (current != null) {

next = current.next;

current.next = prev;

prev = current;

current = next;

}

head = prev;

}

public void display() {

Node current = head;

System.out.print("List = [");

while (current != null) {

System.out.print(current.data + (current.next != null ? ", " : ""));

current = current.next;

}

System.out.println("]");

}

public static void main(String[] args) {

Ass4prg2 linkedList1 = new Ass4prg2();

// Test Case 1

System.out.println("Test Case 1:");

linkedList1.insert(1);

linkedList1.insert(2);

linkedList1.insert(3);

linkedList1.insert(4);

linkedList1.insert(5);

System.out.print("Before Reversing: ");

linkedList1.display();

linkedList1.reverse();

System.out.print("After Reversing: ");

linkedList1.display();

// Test Case 2

System.out.println("\nTest Case 2:");

Ass4prg2 linkedList2 = new Ass4prg2();

linkedList2.insert(10);

linkedList2.insert(20);

linkedList2.insert(30);

System.out.print("Before Reversing: ");

linkedList2.display();

linkedList2.reverse();

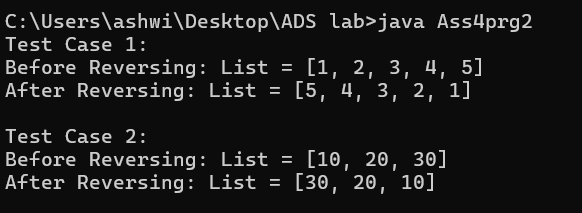
System.out.print("After Reversing: ");

linkedList2.display();

}

}

Output:



**3. Detect a cycle in a linked list.**

* **Test Case 1**:  
  Input: List = [1 → 2 → 3 → 4 → 5 → 3 (cycle)]  
  Output: Cycle Detected
* **Test Case 2**:  
  Input: List = [6 → 7 → 8 → 9]  
  Output: No Cycle

Program:

class Node {

int data;

Node next;

public Node(int data) {

this.data = data;

this.next = null;

}

}

class Ass4prg3 {

private Node head;

public void insert(int data) {

Node newNode = new Node(data);

if (head == null) {

head = newNode;

return;

}

Node current = head;

while (current.next != null) {

current = current.next;

}

current.next = newNode;

}

public void createCycle(int position) {

Node current = head;

Node cycleNode = null;

int count = 0;

while (current != null) {

if (count == position) {

cycleNode = current;

}

current = current.next;

count++;

}

if (cycleNode != null) {

current = head;

while (current.next != null) {

current = current.next;

}

current.next = cycleNode;

}

}

public boolean detectCycle() {

Node slow = head;

Node fast = head;

while (fast != null && fast.next != null) {

slow = slow.next;

fast = fast.next.next;

if (slow == fast) {

return true;

}

}

return false;

}

public void display() {

Node current = head;

System.out.print("List = [");

boolean hasCycle = detectCycle();

if (hasCycle) {

System.out.print("Cycle detected, not displaying full list.");

} else {

while (current != null) {

System.out.print(current.data + (current.next != null ? " → " : ""));

current = current.next;

}

}

System.out.println("]");

}

public static void main(String[] args) {

Ass4prg3 linkedList1 = new Ass4prg3();

// Test Case 1: Cycle Detection

System.out.println("Test Case 1:");

linkedList1.insert(1);

linkedList1.insert(2);

linkedList1.insert(3);

linkedList1.insert(4);

linkedList1.insert(5);

linkedList1.createCycle(2);

linkedList1.display();

System.out.println(linkedList1.detectCycle() ? "Cycle Detected" : "No Cycle");

// Test Case 2: No Cycle Detection

System.out.println("\nTest Case 2:");

Ass4prg3 linkedList2 = new Ass4prg3();

linkedList2.insert(6);

linkedList2.insert(7);

linkedList2.insert(8);

linkedList2.insert(9);

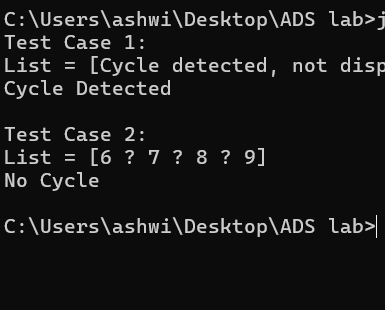
linkedList2.display();

System.out.println(linkedList2.detectCycle() ? "Cycle Detected" : "No Cycle");

}

}

Output:



**4. Merge two sorted linked lists.**

* **Test Case 1**:  
  Input: List1 = [1, 3, 5], List2 = [2, 4, 6]  
  Output: Merged List = [1, 2, 3, 4, 5, 6]
* **Test Case 2**:  
  Input: List1 = [10, 15, 20], List2 = [12, 18, 25]  
  Output: Merged List = [10, 12, 15, 18, 20, 25]

**Program:**

class Node {

int data;

Node next;

public Node(int data) {

this.data = data;

this.next = null;

}

}

class Ass4prg4 {

private Node head;

public void insert(int data) {

Node newNode = new Node(data);

if (head == null) {

head = newNode;

return;

}

Node current = head;

while (current.next != null) {

current = current.next;

}

current.next = newNode;

}

public void display() {

Node current = head;

System.out.print("List = [");

while (current != null) {

System.out.print(current.data + (current.next != null ? " → " : ""));

current = current.next;

}

System.out.println("]");

}

public static Ass4prg4 merge(Ass4prg4 list1, Ass4prg4 list2) {

Ass4prg4 mergedList = new Ass4prg4();

Node current1 = list1.head;

Node current2 = list2.head;

while (current1 != null && current2 != null) {

if (current1.data < current2.data) {

mergedList.insert(current1.data);

current1 = current1.next;

} else {

mergedList.insert(current2.data);

current2 = current2.next;

}

}

while (current1 != null) {

mergedList.insert(current1.data);

current1 = current1.next;

}

while (current2 != null) {

mergedList.insert(current2.data);

current2 = current2.next;

}

return mergedList;

}

public static void main(String[] args) {

// Test Case 1

System.out.println("Test Case 1:");

Ass4prg4 list1 = new Ass4prg4();

list1.insert(1);

list1.insert(3);

list1.insert(5);

Ass4prg4 list2 = new Ass4prg4();

list2.insert(2);

list2.insert(4);

list2.insert(6);

Ass4prg4 mergedList1 = merge(list1, list2);

mergedList1.display(); // Output: Merged List = [1, 2, 3, 4, 5, 6]

// Test Case 2

System.out.println("\nTest Case 2:");

Ass4prg4 list3 = new Ass4prg4();

list3.insert(10);

list3.insert(15);

list3.insert(20);

Ass4prg4 list4 = new Ass4prg4();

list4.insert(12);

list4.insert(18);

list4.insert(25);

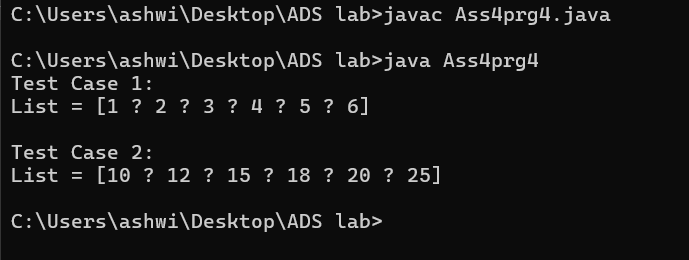
Ass4prg4 mergedList2 = merge(list3, list4);

mergedList2.display();

}

}

Output:



**5. Find the nth node from the end of a linked list.**

* **Test Case 1**:  
  Input: List = [10, 20, 30, 40, 50], n = 2  
  Output: 40
* **Test Case 2**:  
  Input: List = [5, 15, 25, 35], n = 4  
  Output: 5

**Program:**

class Node {

int data;

Node next;

public Node(int data) {

this.data = data;

this.next = null;

}

}

class Ass4prg5 {

private Node head;

public void insert(int data) {

Node newNode = new Node(data);

if (head == null) {

head = newNode;

return;

}

Node current = head;

while (current.next != null) {

current = current.next;

}

current.next = newNode;

}

public Integer findNthFromEnd(int n) {

Node firstPointer = head;

Node secondPointer = head;

for (int i = 0; i < n; i++) {

if (firstPointer == null) {

return null;

}

firstPointer = firstPointer.next;

}

while (firstPointer != null) {

firstPointer = firstPointer.next;

secondPointer = secondPointer.next;

}

return secondPointer != null ? secondPointer.data : null;

}

public void display() {

Node current = head;

System.out.print("List = [");

while (current != null) {

System.out.print(current.data + (current.next != null ? ", " : ""));

current = current.next;

}

System.out.println("]");

}

public static void main(String[] args) {

// Test Case 1

System.out.println("Test Case 1:");

Ass4prg5 list1 = new Ass4prg5();

list1.insert(10);

list1.insert(20);

list1.insert(30);

list1.insert(40);

list1.insert(50);

Integer result1 = list1.findNthFromEnd(2);

System.out.println("Output: " + result1);

// Test Case 2

System.out.println("\nTest Case 2:");

Ass4prg5 list2 = new Ass4prg5();

list2.insert(5);

list2.insert(15);

list2.insert(25);

list2.insert(35);

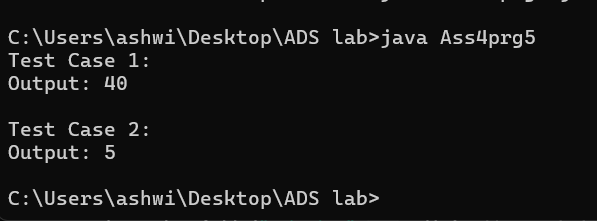
Integer result2 = list2.findNthFromEnd(4);

System.out.println("Output: " + result2);

}

}

**Output:**



**6. Remove duplicates from a sorted linked list.**

* **Test Case 1**:  
  Input: List = [1, 1, 2, 3, 3, 4]  
  Output: List = [1, 2, 3, 4]
* **Test Case 2**:  
  Input: List = [7, 7, 8, 9, 9, 10]  
  Output: List = [7, 8, 9, 10]

**Program:**

class Node {

int data;

Node next;

public Node(int data) {

this.data = data;

this.next = null;

}

}

class Ass4prg6 {

private Node head;

public void insert(int data) {

Node newNode = new Node(data);

if (head == null) {

head = newNode;

return;

}

Node current = head;

while (current.next != null) {

current = current.next;

}

current.next = newNode;

}

public void removeDuplicates() {

Node current = head;

while (current != null && current.next != null) {

if (current.data == current.next.data) {

current.next = current.next.next;

} else {

current = current.next;

}

}

}

public void display() {

Node current = head;

System.out.print("List = [");

while (current != null) {

System.out.print(current.data + (current.next != null ? ", " : ""));

current = current.next;

}

System.out.println("]");

}

public static void main(String[] args) {

// Test Case 1

System.out.println("Test Case 1:");

Ass4prg6 list1 = new Ass4prg6();

list1.insert(1);

list1.insert(1);

list1.insert(2);

list1.insert(3);

list1.insert(3);

list1.insert(4);

list1.removeDuplicates();

list1.display();

// Test Case 2

System.out.println("\nTest Case 2:");

Ass4prg6 list2 = new Ass4prg6();

list2.insert(7);

list2.insert(7);

list2.insert(8);

list2.insert(9);

list2.insert(9);

list2.insert(10);

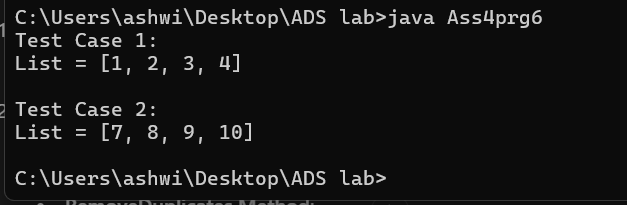
list2.removeDuplicates();

list2.display(); // Expected Output: [7, 8, 9, 10]

}

}

Output:



**7. Implement a doubly linked list with insert, delete, and traverse operations.**

* **Test Case 1**:  
  Input: Insert 10 → Insert 20 → Insert 30 → Delete 20  
  Output: List = [10, 30]
* **Test Case 2**:  
  Input: Insert 1 → Insert 2 → Insert 3 → Delete 1  
  Output: List = [2, 3]

**Program:**

class Node {

int data;

Node next;

Node prev;

public Node(int data) {

this.data = data;

this.next = null;

this.prev = null;

}

}

class Ass4prg7 {

private Node head;

public void insert(int data) {

Node newNode = new Node(data);

if (head == null) {

head = newNode;

return;

}

Node current = head;

while (current.next != null) {

current = current.next;

}

current.next = newNode;

newNode.prev = current;

public void delete(int data) {

Node current = head;

while (current != null) {

if (current.data == data) {

if (current.prev != null) {

current.prev.next = current.next;

} else {

head = current.next;

}

if (current.next != null) {

current.next.prev = current.prev; reference

}

return;

}

current = current.next;

}

}

public void traverse() {

Node current = head;

System.out.print("List = [");

while (current != null) {

System.out.print(current.data + (current.next != null ? ", " : ""));

current = current.next;

}

System.out.println("]");

}

public static void main(String[] args) {

// Test Case 1

System.out.println("Test Case 1:");

Ass4prg7 list1 = new Ass4prg7();

list1.insert(10);

list1.insert(20);

list1.insert(30);

list1.delete(20);

list1.traverse();]

// Test Case 2

System.out.println("\nTest Case 2:");

Ass4prg7 list2 = new Ass4prg7();

list2.insert(1);

list2.insert(2);

list2.insert(3);

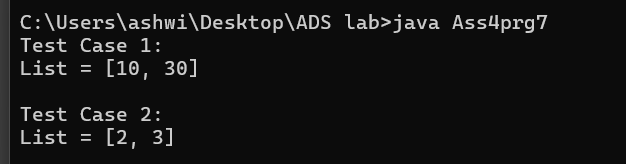
list2.delete(1);

list2.traverse();

}

}

**Output:**



**8. Reverse a doubly linked list.**

* **Test Case 1**:  
  Input: List = [5, 10, 15, 20]  
  Output: List = [20, 15, 10, 5]
* **Test Case 2**:  
  Input: List = [4, 8, 12]  
  Output: List = [12, 8, 4]

**Program:**

class Node {

int data;

Node next;

Node prev;

public Node(int data) {

this.data = data;

this.next = null;

this.prev = null;

}

}

class Ass4prg8 {

private Node head;

public void insert(int data) {

Node newNode = new Node(data);

if (head == null) {

head = newNode;

return;

}

Node current = head;

while (current.next != null) {

current = current.next;

}

current.next = newNode;

newNode.prev = current;

}

public void reverse() {

Node current = head;

Node temp = null;

while (current != null) {

temp = current.prev;

current.prev = current.next;

current.next = temp;

current = current.prev;

}

if (temp != null) {

head = temp.prev;

}

}

linked list

public void traverseAsSingly() {

Node current = head;

System.out.print("List = ");

while (current != null) {

System.out.print(current.data + (current.next != null ? " ->" : ""));

current = current.next;

}

System.out.println();

}

public static void main(String[] args) {

// Test Case 1

System.out.println("Test Case 1:");

Ass4prg8 list1 = new Ass4prg8();

list1.insert(5);

list1.insert(10);

list1.insert(15);

list1.insert(20);

list1.reverse();

list1.traverseAsSingly();

// Test Case 2

System.out.println("\nTest Case 2:");

Ass4prg8 list2 = new Ass4prg8();

list2.insert(4);

list2.insert(8);

list2.insert(12);

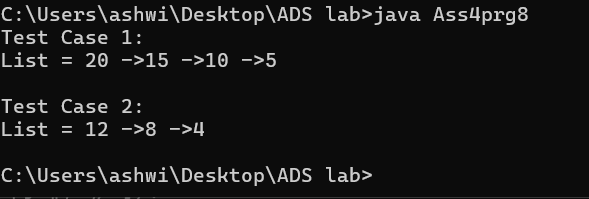
list2.reverse();

list2.traverseAsSingly();

}

}

Output:



**9. Add two numbers represented by linked lists.**

* **Test Case 1**:  
  Input: List1 = [2 → 4 → 3], List2 = [5 → 6 → 4] (243 + 465)  
  Output: Sum List = [7 → 0 → 8]
* **Test Case 2**:  
  Input: List1 = [9 → 9 → 9], List2 = [1] (999 + 1)  
  Output: Sum List = [0 → 0 → 0 → 1]

**Program:**

class Node {

int data;

Node next;

public Node(int data) {

this.data = data;

this.next = null;

}

}

class Ass4prg9 {

Node head;

public void insert(int data) {

Node newNode = new Node(data);

if (head == null) {

head = newNode;

return;

}

Node current = head;

while (current.next != null) {

current = current.next;

}

current.next = newNode;

}

public void printList() {

Node current = head;

while (current != null) {

System.out.print(current.data + (current.next != null ? " ->" : ""));

current = current.next;

}

System.out.println();

}

public static LinkedList addTwoLists(LinkedList list1, LinkedList list2) {

LinkedList result = new LinkedList();

Node p1 = list1.head;

Node p2 = list2.head;

int carry = 0;

while (p1 != null || p2 != null || carry != 0) {

int sum = carry;

if (p1 != null) {

sum += p1.data;

p1 = p1.next;

}

if (p2 != null) {

sum += p2.data;

p2 = p2.next;

}

carry = sum / 10;

result.insert(sum % 10);

}

return result;

}

public static void main(String[] args) {

// Test Case 1

System.out.println("Test Case 1:");

LinkedList list1 = new LinkedList();

list1.insert(2);

list1.insert(4);

list1.insert(3);

LinkedList list2 = new LinkedList();

list2.insert(5);

list2.insert(6);

list2.insert(4);

LinkedList sumList1 = LinkedList.addTwoLists(list1, list2);

System.out.print("Sum List = ");

sumList1.printList();

// Test Case 2

System.out.println("\nTest Case 2:");

LinkedList list3 = new LinkedList();

list3.insert(9);

list3.insert(9);

list3.insert(9);

LinkedList list4 = new LinkedList();

list4.insert(1);

LinkedList sumList2 = LinkedList.addTwoLists(list3, list4);

System.out.print("Sum List = ");

sumList2.printList();

}

}

Output:



**10. Rotate a linked list by k places.**

* **Test Case 1**:  
  Input: List = [10, 20, 30, 40, 50], k = 2  
  Output: List = [30, 40, 50, 10, 20]
* **Test Case 2**:  
  Input: List = [5, 10, 15, 20], k = 3  
  Output: List = [20, 5, 10, 15]

**Program:**

class Node {

int data;

Node next;

public Node(int data) {

this.data = data;

this.next = null;

}

}

class Ass4prg10 {

Node head;

public void insert(int data) {

Node newNode = new Node(data);

if (head == null) {

head = newNode;

return;

}

Node current = head;

while (current.next != null) {

current = current.next;

}

current.next = newNode;

}

public void printList() {

Node current = head;

while (current != null) {

System.out.print(current.data + (current.next != null ? " ->" : ""));

current = current.next;

}

System.out.println();

}

public void rotate(int k) {

if (head == null || k <= 0) return;

Node current = head;

int length = 1;

while (current.next != null) {

current = current.next;

length++;

}

k = k % length;

if (k == 0) return;

current.next = head;

int newTailPosition = length - k;

Node newTail = head;

for (int i = 1; i < newTailPosition; i++) {

newTail = newTail.next;

}

head = newTail.next;

newTail.next = null;

}

public static void main(String[] args) {

System.out.println("Test Case 1:");

LinkedList list1 = new LinkedList();

list1.insert(10);

list1.insert(20);

list1.insert(30);

list1.insert(40);

list1.insert(50);

list1.rotate(2);

System.out.print("List = ");

list1.printList();

System.out.println("\nTest Case 2:");

LinkedList list2 = new LinkedList();

list2.insert(5);

list2.insert(10);

list2.insert(15);

list2.insert(20);

list2.rotate(3);

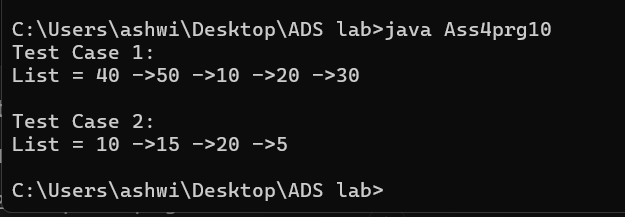
System.out.print("List = ");

list2.printList();

}

}

**Output:**



**11. Flatten a multilevel doubly linked list.**

* **Test Case 1**:  
  Input: List = [1 → 2 → 3, 3 → 7 → 8, 8 → 10 → 12]  
  Output: Flattened List = [1 → 2 → 3 → 7 → 8 → 10 → 12]
* **Test Case 2**:  
  Input: List = [1 → 2 → 3, 2 → 5 → 6, 6 → 7 → 9]  
  Output: Flattened List = [1 → 2 → 5 → 6 → 7 → 9 → 3]

**12. Split a circular linked list into two halves.**

* **Test Case 1**:  
  Input: Circular List = [1 → 2 → 3 → 4 → 5 → 6 → (back to 1)]  
  Output: List1 = [1 → 2 → 3], List2 = [4 → 5 → 6]
* **Test Case 2**:  
  Input: Circular List = [10 → 20 → 30 → 40 → (back to 10)]  
  Output: List1 = [10 → 20], List2 = [30 → 40]

**13. Insert a node in a sorted circular linked list.**

* **Test Case 1**:  
  Input: Circular List = [10 → 20 → 30 → 40 → (back to 10)], Insert 25  
  Output: Circular List = [10 → 20 → 25 → 30 → 40 → (back to 10)]
* **Test Case 2**:  
  Input: Circular List = [5 → 15 → 25 → (back to 5)], Insert 10  
  Output: Circular List = [5 → 10 → 15 → 25 → (back to 5)]

**14. Check if two linked lists intersect, and find the intersection point if they do.**

* **Test Case 1**:  
  Input: List1 = [1 → 2 → 3 → 4 → 5], List2 = [6 → 7 → 4 → 5]  
  Output: Intersection Point = 4
* **Test Case 2**:  
  Input: List1 = [10 → 20 → 30 → 40], List2 = [15 → 25 → 35]  
  Output: No Intersection

**15. Find the middle element of a linked list in one pass.**

* **Test Case 1**:  
  Input: List = [1, 2, 3, 4, 5]  
  Output: Middle = 3
* **Test Case 2**:  
  Input: List = [11, 22, 33, 44, 55, 66]  
  Output: Middle = 44