Prompt | Scratchpad | Our Solution(s) | Video Explanation | Run Code

Solution 1

```
            node = node?.next
            if nodeToRemove?.value == value {
                remove(node: nodeToRemove!)
            }
        }
    }

    // O(1) time | O(1) space
    func insertBefore(node: Node, nodeToInsert: Node) {
        if nodeToInsert === head, nodeToInsert === tail {
            return
        }

        remove(node: nodeToInsert)
        nodeToInsert.previous = node.previous
        nodeToInsert.next = node

        if node.previous == nil {
            head = nodeToInsert
        } else {
            node.previous?.next = nodeToInsert
        }

        node.previous = nodeToInsert
    }

    // O(1) time | O(1) space
    func insertAfter(node: Node, nodeToInsert: Node) {
        if nodeToInsert === head, nodeToInsert === tail {
            return
        }

        remove(node: nodeToInsert)
        nodeToInsert.previous = node
        nodeToInsert.next = node.next

        if node.next == nil {
            tail = nodeToInsert
        } else {
            node.next?.previous = nodeToInsert
        }

        node.next = nodeToInsert
    }

    // O(1) time | O(1) space
    func setHead(node: Node) {
        if head == nil {
            head = node
            tail = node
            return
        }

        insertBefore(node: head!, nodeToInsert: node)
    }

    // O(1) time | O(1) space
    func setTail(node: Node) {
        if tail == nil {
            setHead(node: node)
            return
        }

        insertAfter(node: tail!, nodeToInsert: node)
    }

    // O(P) time | O(1) space
    func insertAtPosition(position: Int, nodeToInsert: Node) {
        if position == 1 {
            setHead(node: nodeToInsert)
            return

        }

        var node = head
        var currentPosition = 1
        while node !== nil, currentPosition != position {
            node = node?.next
            currentPosition = currentPosition + 1
        }

        if node !== nil {
            insertBefore(node: node!, nodeToInsert: nodeToInsert)
        } else {
            setTail(node: nodeToInsert)
        }
    }

    func removeNodeBindings(node: Node) {
        if let previous = node.previous {
            previous.next = node.next
        }
```

Your Solutions | Run Code

Solution 1 | Solution 2 | Solution 3

```
class Program {
    class Node {
        var value: Int
        var previous: Node?
        var next: Node?

        init(value: Int) {
            self.value = value
            previous = nil
            next = nil
        }
    }

    class DoublyLinkedList {
        var head: Node?
        var tail: Node?

        init() {
            head = nil
            tail = nil
        }

        func containsNodeWithValue(value: Int) -> Bool {
            // Write your code here.
            return false
        }

        func remove(node: Node) {
            // Write your code here.
        }

        func removeNodesWithValue(value: Int) {
            // Write your code here.
        }

        func insertBefore(node: Node, nodeToInsert: Node) {
            // Write your code here.
        }

        func insertAfter(node: Node, nodeToInsert: Node) {
            // Write your code here.
        }

        func setHead(node: Node) {
            // Write your code here.
        }

        func setTail(node: Node) {
            // Write your code here.
        }

        func insertAtPosition(position: Int, nodeToInsert: Node) {
            // Write your code here.
        }
    }
}

```

Custom Output | Raw Output | Submit Code

```
            }

            if let next = node.next {
                next.previous = node.previous
            }

            node.previous = nil
            node.next = nil
        }
    }
}

```

Run or submit code when you're ready.