Linked List: Leetcode

(Part 3)

Q2. Add Two Numbers

```
class Solution {
public:
    ListNode* addTwoNumbers(ListNode* l1, ListNode* l2) {
        ListNode dummy, *tail = &dummy;
        int carry = 0;
        while (l1 || l2 || carry) {
            int sum = carry;
            if (l1) sum += l1->val, l1 = l1->next;
            if (l2) sum += l2->val, l2 = l2->next;
            tail->next = new ListNode(sum % 10);
            tail = tail->next;
            carry = sum / 10;
        }
        return dummy.next;
    }
};
```

Q2095. Delete the Middle Node of a Linked List

```
class Solution {
public:
    ListNode* deleteMiddle(ListNode* head) {
        if (!head || !head->next) return nullptr;
        ListNode* slow = head, *fast = head, *prev =
nullptr;
        while (fast && fast->next) {
            fast = fast->next->next;
            prev = slow;
            slow = slow->next;
        }
        prev->next = slow->next;
        return head;
    }
};
```

Q1290. Convert Binary Number in a Linked List to Integer

```
class Solution {
public:
    int getDecimalValue(ListNode* head) {
        int val = 0;
        while (head) {
            val = (val << 1) | head->val;
            head = head->next;
        }
        return val;
    }
};
```

Q707. Design Linked List

```
class MyLinkedList {
    struct Node {
        int val;
        Node* next;
        Node(int x) : val(x), next(nullptr) {}
    };
    Node* head;
    int size;
public:
    MyLinkedList() {
        head = nullptr;
        size = 0;
    }
    int get(int index) {
        if (index < 0 || index >= size) return -1;
        Node* curr = head;
        while (index--) curr = curr->next;
        return curr->val;
    }
    void addAtHead(int val) {
        Node* node = new Node(val);
        node->next = head;
        head = node;
        size++;
    }
    void addAtTail(int val) {
        Node* node = new Node(val);
        if (!head) head = node;
        else {
            Node* curr = head;
            while (curr->next) curr = curr->next;
```

```
            curr->next = node;
        }
        size++;
    }
    void addAtIndex(int index, int val) {
        if (index < 0 || index > size) return;
        if (index == 0) return addAtHead(val);
        Node* curr = head;
        for (int i = 0; i < index - 1; ++i) curr =
curr->next;
        Node* node = new Node(val);
        node->next = curr->next;
        curr->next = node;
        size++;
    }
    void deleteAtIndex(int index) {
        if (index < 0 || index >= size) return;
        if (index == 0) {
            Node* temp = head;
            head = head->next;
            delete temp;
        } else {
            Node* curr = head;
            for (int i = 0; i < index - 1; ++i) curr =
curr->next;
            Node* temp = curr->next;
            curr->next = curr->next->next;
            delete temp;
        }
        size--;
    }
};
```

Q148. Sort List

```
class Solution {
public:
    ListNode* sortList(ListNode* head) {
        if (!head || !head->next) return head;
        ListNode* slow = head, *fast = head->next;
        while (fast && fast->next) {
            slow = slow->next;
            fast = fast->next->next;
        }
        ListNode* mid = slow->next;
        slow->next = nullptr;
        ListNode* left = sortList(head);
        ListNode* right = sortList(mid);
        return merge(left, right);
    }
    ListNode* merge(ListNode* l1, ListNode* l2) {
        ListNode dummy, *tail = &dummy;
        while (l1 && l2) {
            if (l1->val < l2->val)
                tail->next = l1, l1 = l1->next;
            else
                tail->next = l2, l2 = l2->next;
            tail = tail->next;
        }
        tail->next = l1 ? l1 : l2;
        return dummy.next;
    }
};
```

Q445. Add Two Numbers II

```
class Solution {
public:
    ListNode* reverse(ListNode* head) {
        ListNode* prev = nullptr;
        while (head) {
            ListNode* next = head->next;
            head->next = prev;
            prev = head;
            head = next;
        }
        return prev;
    }
    ListNode* addTwoNumbers(ListNode* l1, ListNode* l2) {
        l1 = reverse(l1);
        l2 = reverse(l2);
        ListNode* head = nullptr;
        int carry = 0;
        while (l1 || l2 || carry) {
            int sum = carry;
            if (l1) sum += l1->val, l1 = l1->next;
            if (l2) sum += l2->val, l2 = l2->next;
            ListNode* node = new ListNode(sum % 10);
            node->next = head;
            head = node;
            carry = sum / 10;
        }
        return head;
    }
};
```

Q2487. Remove Nodes From Linked List

```
class Solution {
public:
    ListNode* reverse(ListNode* head) {
        ListNode* prev = nullptr;
        while (head) {
            ListNode* next = head->next;
            head->next = prev;
            prev = head;
            head = next;
        }
        return prev;
    }
    ListNode* removeNodes(ListNode* head) {
        head = reverse(head);
        ListNode* curr = head, *maxNode = head;
        while (curr && curr->next) {
            if (curr->next->val < maxNode->val)
                curr->next = curr->next->next;
            else {
                curr = curr->next;
                maxNode = curr;
            }
        }
        return reverse(head);
    }
};
```