Linked List Problems - Easy, Medium, Hard

Middle of a Linked List

Find the middle node of a singly linked list.

```
public ListNode findMiddle(ListNode head) { /* implementation */ }
```

Reverse Linked List

Reverse a singly linked list.

```
public ListNode reverseList(ListNode head) { /* implementation */ }
```

Reverse Doubly Linked List

Reverse a doubly linked list.

```
public DoublyListNode reverseDoublyList(DoublyListNode head) { /* implementation */ }
```

Detect Loop

Detect if a loop exists in a linked list.

```
public boolean detectLoop(ListNode head) { /* implementation */ }
```

Length of Loop

Return the length of the loop in a linked list.

```
public int loopLength(ListNode head) { /* implementation */ }
```

Reverse in Groups

Reverse linked list in groups of k nodes.

```
public ListNode reverseInGroups(ListNode head, int k) { /* implementation */ }
```

LRU Cache

Implement an LRU Cache with get and put in O(1) time.

```
class LRUCache { /* implementation */ }
```

Merge K Sorted Lists

Merge k sorted linked lists into one sorted list.

```
public ListNode mergeKLists(ListNode[] lists) { /* implementation */ }
```

Reverse Alternate K Nodes

Reverse every alternate k nodes in the linked list.

```
public ListNode reverseAlternateKNodes(ListNode head, int k) { /* implementation */ }
```

Flattening Linked List

Flatten a multi-level linked list using bottom pointers.

```
public FlattenNode flatten(FlattenNode root) { /* implementation */ }
```

Clone with Random Pointers

Clone a linked list with random pointers.

```
public RandomListNode cloneRandomList(RandomListNode head) { /* implementation */ }
```