148. Sort List
We will do merge sort for this question. TC is O(nlogn), SC is O(1)

```
var sortList = function(head) {
    if (!head || !head.next) {
        return head;
    }

    let fast = head.next, slow = head;

    while (fast && fast.next) {
        fast = fast.next.next;
        slow = slow.next;
    }
    const second = slow.next;
    slow.next = null;
    let l = sortList(head);
    let r = sortList(second);
    return merge(l, r)
};

const merge = (l, r) => {
    if (!l || !r) {
        return l || r;
    }
    if (l.val > r.val) {
        [l, r] = [r, l];
    }
    let head = l, prev = l;
    l = l.next;

    while (l && r) {
        if (l.val < r.val) {
            prev.next = l;
            l = l.next;
        } else {
            prev.next = r;
            r = r.next;
        }
        prev = prev.next
    }
    prev.next = l || r;
    return head;
}
```

23. Merge k Sorted Lists
We will use heapq to solve this question. TC is O(nlogn), SC is O(n)

```
from heapq import *
class Solution:
    def mergeKLists(self, lists: List[ListNode]) -> ListNode:
        dummy = ListNode(0)
        dummy_mem = dummy
        heap = []
        for idx, node in enumerate(lists):
            if node:
                heappush(heap, [node.val, idx])
        while heap:
            val, idx = heappop(heap)
            dummy.next = lists[idx]
            dummy = dummy.next
            if lists[idx].next:
                lists[idx] = lists[idx].next
                heappush(heap, [lists[idx].val, idx])
        return dummy_mem.next
```

21. Merge Two Sorted Lists
Very straightforward to merge two sorted list. TC is O(n), SC is O(1)

```
var mergeTwoLists = function(l1, l2) {
    let dummy = new ListNode(0)
    const dummy_mem = dummy;
    while (l1 && l2) {
        if (l1.val < l2.val) {
            dummy.next = l1;
            l1 = l1.next;
        } else {
            dummy.next = l2;
            l2 = l2.next;
        }
        dummy = dummy.next;
    }
    dummy.next = l1 || l2;
    return dummy_mem.next;
};
```

147. Insertion Sort List

We will insert our node one by one from beginning to end.
TC is O(n^2), SC is O(1)

```
class Solution:
    def insertionSortList(self, head: ListNode) -> ListNode:
        helper = ListNode(0)
        pre = helper
        cur = head
        next = None
        while cur:
            next = cur.next
            while pre.next and cur.val > pre.next.val:
                pre = pre.next
            cur.next = pre.next
            pre.next = cur

            pre = helper
            cur = next
        return helper.next
```

707. Design Linked List

We will use a linked list with a length to solve this question. TC is O(n), SC is O(n)

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

```
class MyLinkedList:

    def __init__(self):
        """
        Initialize your data structure here.
        """
        self.head = Node(0)
        self.length = 0

    def get(self, index: int) -> int:
        """
        Get the value of the index-th node in the linked list. If the index is invalid, return -1.
        """
        if index >= self.length:
            return -1
        node = self.head.next
        while index > 0:
            node = node.next
            index -= 1
        return node.val

    def addAtHead(self, val: int) -> None:
        """
```

Add a node of value val before the first element of the linked list. After the insertion, the new node will be the first node of the linked list.
"""
next = self.head.next
self.head.next = Node(val)
self.head.next.next = next
self.length += 1

def addAtTail(self, val: int) -> None:
"""
Append a node of value val to the last element of the linked list.
"""
node = self.head
while node.next:
node = node.next
node.next = Node(val)
self.length += 1

def addAtIndex(self, index: int, val: int) -> None:
"""
Add a node of value val before the index-th node in the linked list. If index equals to the length of linked list,

```
the node will be appended to the end of linked list. If index
is greater than the length, the node will not be inserted.
        """
        if index > self.length:
            return
        node = self.head
        while index > 0:
            node = node.next
            index -= 1
        next = node.next
        node.next = Node(val)
        node.next.next = next
        self.length += 1

    def deleteAtIndex(self, index: int) -> None:
        """
        Delete the index-th node in the linked list, if the index
is valid.
        """
        if index >= self.length:
            return
        node = self.head
        while index > 0:
            node = node.next
            index -= 1
        node.next = node.next.next
```

```
self.length -= 1
```