Problem1-Leetcode Q912-Sort an Array-Medium

Given an array of integers nums, sort the array in ascending order and return it.

You must solve the problem without using any built-in functions in O(nlog(n)) time complexity and with the smallest space complexity possible.

Example 1:

Input: nums = [5,2,3,1]
Output: [1,2,3,5]
Explanation: After sorting the array, the positions of some numbers are not changed (for example, 2 and 3), while the positions of other numbers are changed (for example, 1 and 5).
Example 2:

Input: nums = [5,1,1,2,0,0]
Output: [0,0,1,1,2,5]
Explanation: Note that the values of nums are not necessarily unique.

Q912 Pseudocode

```
function merge_sort(array nums, integer l, integer r):
if l equals r:
    Return

mid = (l + r) integer division 2

Call merge_sort(nums, l, mid)
Call merge_sort(nums, mid + 1, r)

Temporary array tmp = []
i = l
j = mid + 1

While i <= mid or j <= r:
    if i > mid or (j <= r and nums[j] < nums[i]):
        Append nums[j] to tmp
        j = j + 1
    else:
        Append nums[i] to tmp
        i = i + 1

Assign nums[l to r] to tmp
```

Q912 Code.py

```
from typing import List
class Solution:
    def merge_sort(self, nums, l, r):
        if l == r:
            return
        mid = (l + r) // 2
```

```
        self.merge_sort(nums, l, mid)
        self.merge_sort(nums, mid + 1, r)
        tmp = []
        i, j = l, mid + 1
        while i <= mid or j <= r:
            if i > mid or (j <= r and nums[j] < nums[i]):
                tmp.append(nums[j])
                j += 1
            else:
                tmp.append(nums[i])
                i += 1
        nums[l: r + 1] = tmp

    def sortArray(self, nums: List[int]) -> List[int]:
        self.merge_sort(nums, 0, len(nums) - 1)
        return nums

# Test case
nums = [5, 1, 1, 2, 0, 0]
solution = Solution()
sorted_nums = solution.sortArray(nums)
print(sorted_nums)
```

```
[0, 0, 1, 1, 2, 5]
```

Problem2-Leetcode Q23-Merge k- Sorted Lists-Hard

You are given an array of k linked-lists lists, each linked-list is sorted in ascending order.

Merge all the linked-lists into one sorted linked-list and return it.

Example 1:

Input: lists = [[1,4,5],[1,3,4],[2,6]]
Output: [1,1,2,3,4,4,5,6]
Explanation: The linked-lists are:
[
  1->4->5,
  1->3->4,
  2->6
]
merging them into one sorted list:
1->1->2->3->4->4->5->6
Example 2:

Input: lists = []
Output: []
Example 3:

Input: lists = [[]]
Output: []

Constraints:

k == lists.length
lists[i] is sorted in ascending order.

Q23 Pseudocode

```
function mergeKLists() -> ListNode:
n = length of lists
If n is 0:
    return None
If n is 1:
    return lists[0]

mid = n // 2
leftMerged = mergeKLists(lists from 0 to mid)
rightMerged = mergeKLists(lists from mid to end)

return mergeTwoLists(leftMerged, rightMerged)
-----------------------------------------------------------------

function mergeTwoLists(node1: ListNode, node2: ListNode) -> ListNode:
Create a dummy ListNode
cur = dummy

While node1 is not None and node2 is not None:
    if node1.val <= node2.val:
        cur.next = node1
        node1 = node1.next
    else:
        cur.next = node2
        node2 = node2.next
    cur = cur.next

if node1 is not None:
    cur.next = node1
else:
    cur.next = node2

return dummy.next
```

Q23 Code.py

```
from typing import List, Optional

class ListNode:
    def __init__(self, val=0, next=None):
        self.val = val
        self.next = next

class Solution:
    def mergeKLists(self, lists: List[Optional[ListNode]]) -> Optional[ListNode]:
        n = len(lists)
        if n == 0:
            return None
```

```
        if n == 1:
            return lists[0]
        mid = n // 2
        return self.mergeTwoLists(self.mergeKLists(lists[:mid]),
self.mergeKLists(lists[mid:]))

    def mergeTwoLists(self, node1, node2):
        dummy = cur = ListNode()
        while node1 and node2:
            if node1.val <= node2.val:
                cur.next = node1
                node1 = node1.next
            else:
                cur.next = node2
                node2 = node2.next
            cur = cur.next

        cur.next = node1 if node1 else node2

        return dummy.next

def create_linked_list(arr):
    if not arr:
        return None
    head = ListNode(arr[0])
    current = head
    for val in arr[1:]:
        current.next = ListNode(val)
        current = current.next
    return head

def print_linked_list(node):
    result = []
    while node:
        result.append(node.val)
        node = node.next
    return result

# Test case
lists = [[1, 4, 5], [1, 3, 4], [2, 6]]
linked_lists = [create_linked_list(l) for l in lists]
solution = Solution()
merged_head = solution.mergeKLists(linked_lists)
print(print_linked_list(merged_head))
```

```
[1, 1, 2, 3, 4, 4, 5, 6]
```

Problem3-Leetcode Q215-Kth Largest Element in an Array-Medium

Given an integer array nums and an integer k, return the kth largest element in the array.

Note that it is the kth largest element in the sorted order, not the kth distinct element.

Can you solve it without sorting?

Example 1:

Input: nums = [3,2,1,5,6,4], k = 2
Output: 5
Example 2:

Input: nums = [3,2,3,1,2,4,5,5,6], k = 4
Output: 4

Q215 Pseudocode

```
function findKthLargest(nums, k):
    function quick_select(nums, k):
    Choose a pivot randomly from nums
    initialize three lists: big, equal, small

for each num in nums:
    if num > pivot:
        append num to big
    else if num < pivot:
        append num to small
    else:
        append num to equal

if k is less than or equal to the length of big:
    return quick_select(big, k)
if k is greater than the length of nums minus the length of small:
    return quick_select(small, k - length of nums + length of small)

return pivot
```

Q215 Code.py

```
import random
class Solution:
    def findKthLargest(self, nums, k):
        def quick_select(nums, k):
            pivot = random.choice(nums)
            big, equal, small = [], [], []
            for num in nums:
                if num > pivot:
                    big.append(num)
                elif num < pivot:
                    small.append(num)
                else:
                    equal.append(num)
            if k <= len(big):
                return quick_select(big, k)
            if len(nums) - len(small) < k:
                return quick_select(small, k - len(nums) + len(small))
            return pivot
```

```
        return quick_select(nums, k)

nums = [3, 2, 3, 1, 2, 4, 5, 5, 6]
k = 4
solution = Solution()
result = solution.findKthLargest(nums, k)
print(result)
```

```
4
```

Problem4-Leetcode Q75-Sort Colors-Medium

Given an array nums with n objects colored red, white, or blue, sort them in-place so that objects of the same color are adjacent, with the colors in the order red, white, and blue.

We will use the integers 0, 1, and 2 to represent the color red, white, and blue, respectively.

You must solve this problem without using the library's sort function.

Example 1:

Input: nums = [2,0,2,1,1,0]
Output: [0,0,1,1,2,2]
Example 2:

Input: nums = [2,0,1]
Output: [0,1,2]

Constraints:

n == nums.length
1 <= n <= 300
nums[i] is either 0, 1, or 2.

Q75 Pseudocode

```
function sortColors(nums):
function swap(nums, index1, index2):
    swap elements at index1 and index2 in nums

size = length of nums
if size < 2:
    return

zero = -1
two = size - 1
i = 0

while i <= two:
    if nums[i] == 0:
        zero = zero + 1
        swap(nums, i, zero)
        i = i + 1
    else if nums[i] == 1:
```

```
            i = i + 1
        else:
            swap(nums, i, two)
            two = two - 1
```

Q75 Code.py

```
from typing import List
class Solution:
    def sortColors(self, nums: List[int]) -> None:
        """
        Do not return anything, modify nums in-place instead.
        """

        def swap(nums, index1, index2):
            nums[index1], nums[index2] = nums[index2], nums[index1]

        size = len(nums)
        if size < 2:
            return

        zero = -1
        two = size - 1
        i = 0

        while i <= two:
            if nums[i] == 0:
                zero += 1
                swap(nums, i, zero)
                i += 1
            elif nums[i] == 1:
                i += 1
            else:
                swap(nums, i, two)
                two -= 1


nums = [2, 0, 2, 1, 1, 0]
solution = Solution()
solution.sortColors(nums)
print(nums)
```

```
[0, 0, 1, 1, 2, 2]
```

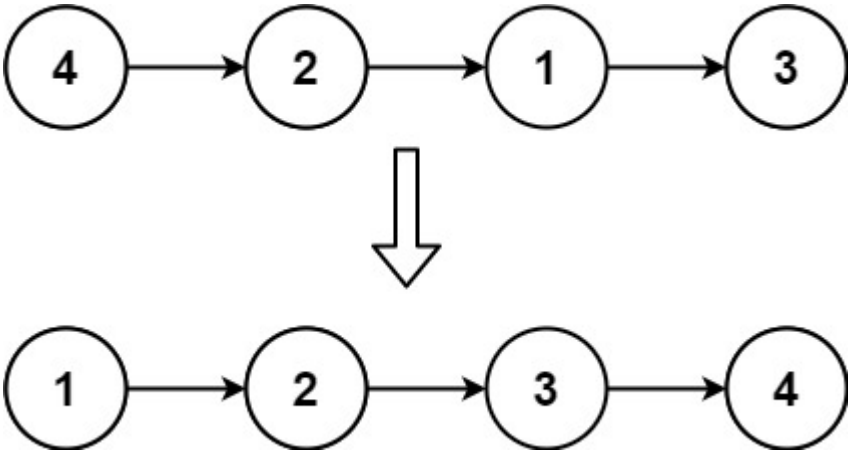
```
```

Problem5:-Leetcode Q147-Insertion Sort List-Medium

Given the head of a singly linked list, sort the list using insertion sort, and return the sorted list's head.

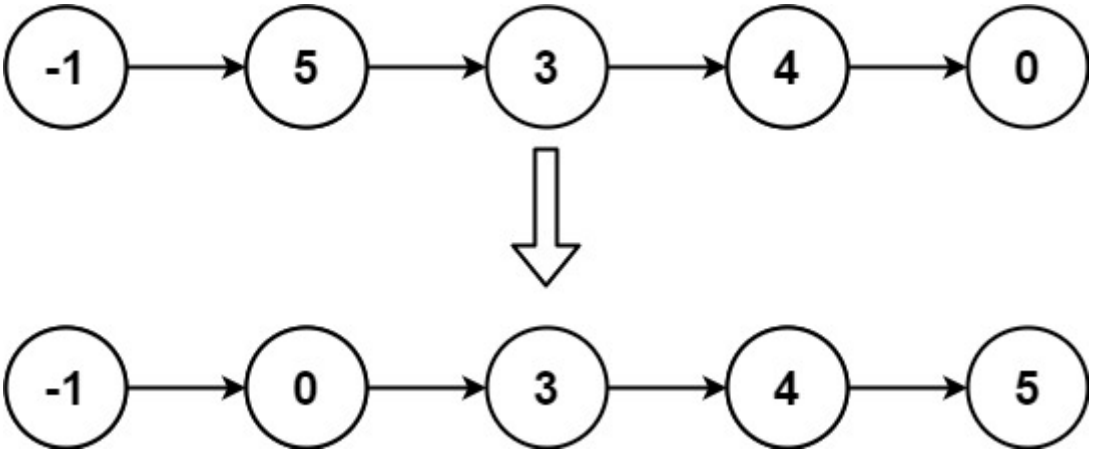
The steps of the insertion sort algorithm:

Insertion sort iterates, consuming one input element each repetition and growing a sorted output list.
At each iteration, insertion sort removes one element from the input data, finds the location it belongs within the sorted list and inserts it there.
It repeats until no input elements remain.



Input: head = [4,2,1,3]

Output: [1,2,3,4]



Input: head = [-1,5,3,4,0]

Output: [-1,0,3,4,5]

Example 3:

Input: head = []

Output: []

#Time: O(n^2)
#Space: O(1)

Add dummy_head before head will help us to handle the insertion easily

Use two pointers

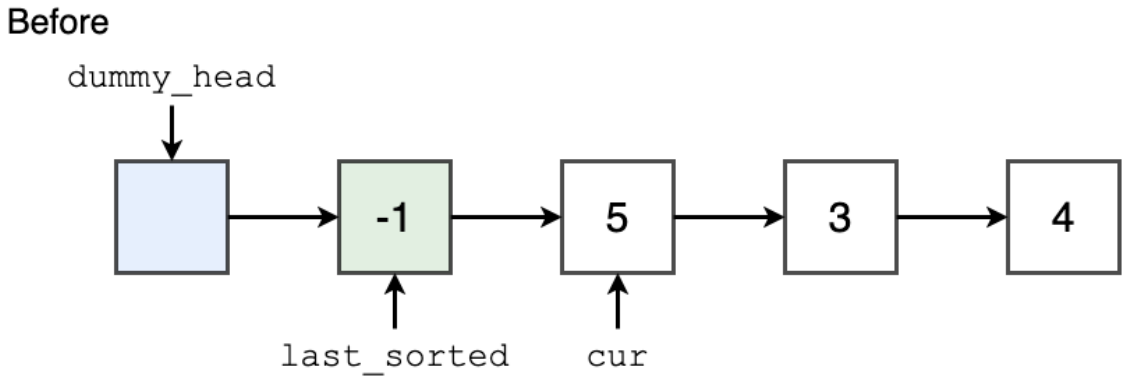
last_sorted: last node of the sorted part, whose value is the largest of the sorted part

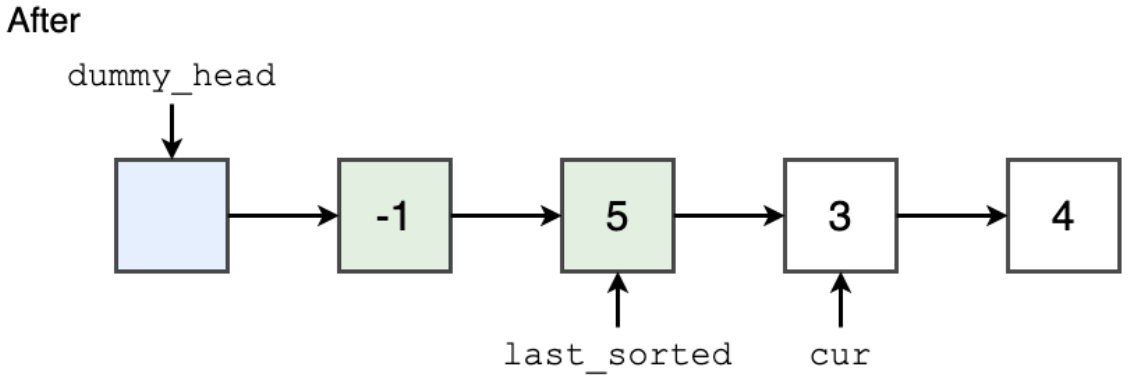
cur: next node of last_sorted, which is the current node to be considered

At the beginning, last_sorted is head and cur is head.next

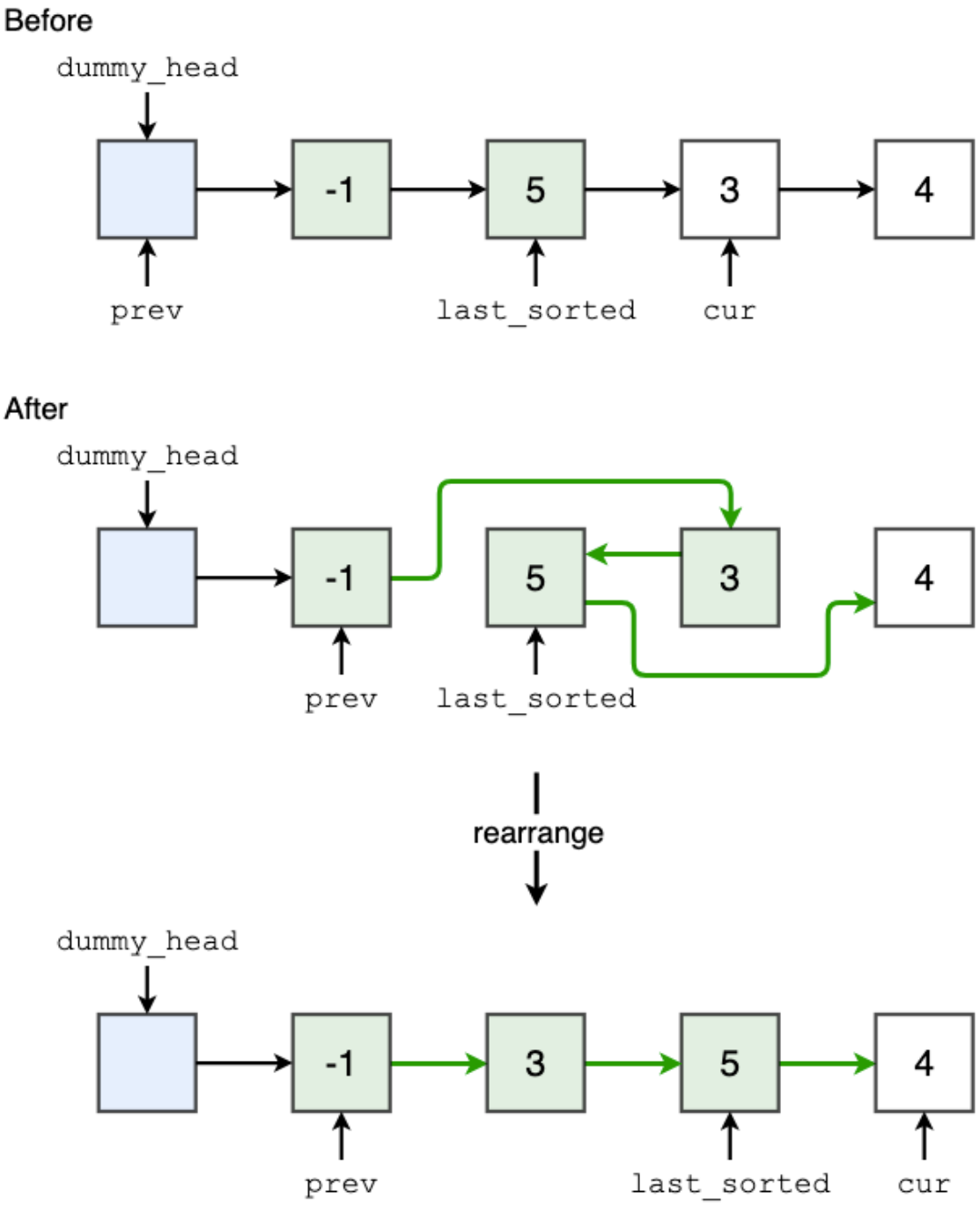
When consider the cur node, there're 2 different cases

last_sorted.val <= cur.val: cur is in the correct order and can be directly move into the sorted part. In this case, we just move last_sorted one step forward





last_sorted.val > cur.val: cur needs to be inserted somewhere in the sorted part. In this case, we let prev start from dummy_head and iteratively compare prev.next.val and cur.val. If prev.next.val > cur.val, we insert cur between prev and prev.next



Q147 Pseudocode

```
function insertionSortList(head):
    create a dummy ListNode with value 0
    set cur to head

while cur is not null:
    set pre to dummy

    while pre.next is not null and pre.next.val is less than or equal to cur.val:
        move pre to pre.next

    set tmp to cur.next
    set cur.next to pre.next
    set pre.next to cur
    set cur to tmp
```

```
return dummy.next
```

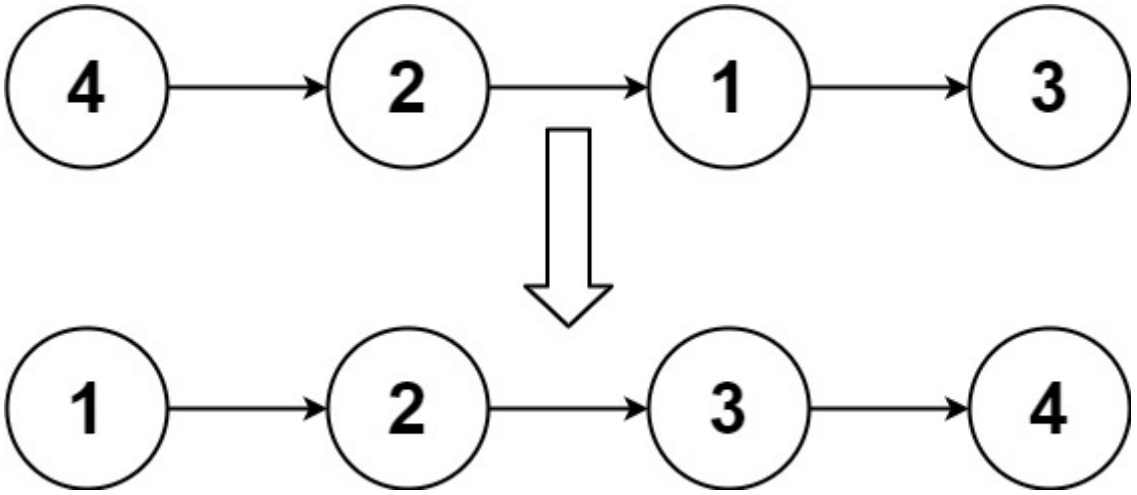
Q147 Code.py

```
class ListNode:
    def __init__(self, val=0, next=None):
        self.val = val
        self.next = next

class Solution:
    def insertionSortList(self, head: ListNode) -> ListNode:
        dummy = ListNode(0)
        cur = head

        while cur:
            pre = dummy
            while pre.next and pre.next.val <= cur.val:
                pre = pre.next
            tmp = cur.next
            cur.next = pre.next
            pre.next = cur
            cur = tmp

        return dummy.next


def create_linked_list(arr):
    if not arr:
        return None
    head = ListNode(arr[0])
    current = head
    for value in arr[1:]:
        current.next = ListNode(value)
        current = current.next
    return head

def print_linked_list(head):
    arr = []
    while head:
        arr.append(head.val)
        head = head.next
    print(arr)


input_list = [-1, 5, 3, 4, 0]
head = create_linked_list(input_list)
solution = Solution()
sorted_head = solution.insertionSortList(head)
print_linked_list(sorted_head)
```

```
[-1, 0, 3, 4, 5]
```

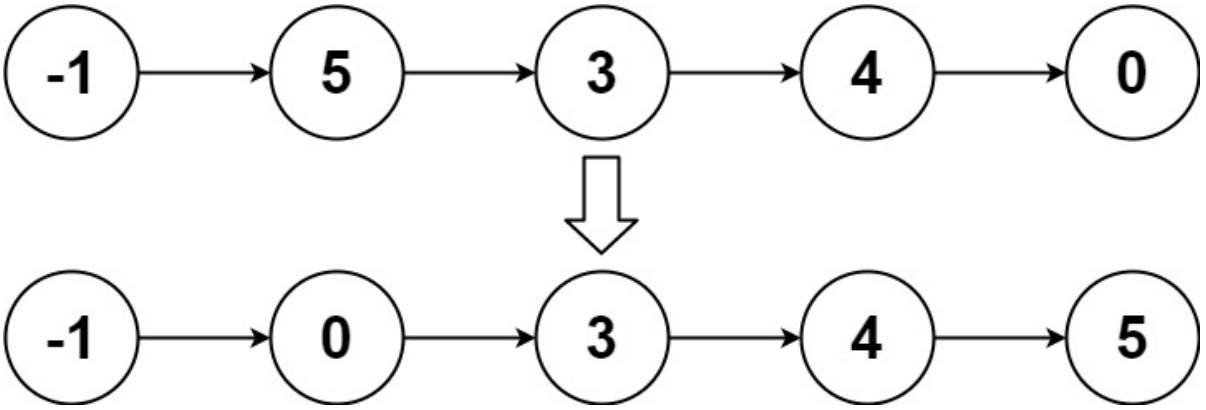
Problem6:-Leetcode Q148-Sort List-Medium

Given the head of a linked list, return the list after sorting it in ascending order.



Input: head = [4,2,1,3]

Output: [1,2,3,4]



Input: head = [-1,5,3,4,0]

Output: [-1,0,3,4,5]

Example 3:

Input: head = []

Output: []

Q148 Pseudocode

```
function sortList(head):
if head is null or head.next is null:
    return head

slow = head
fast = head.next

while fast is not null and fast.next is not null:
    fast = fast.next.next
    slow = slow.next

mid = slow.next
slow.next = null

left = sortList(head)
```

```
right = sortList(mid)

h = new ListNode(0)
res = h

while left is not null and right is not null:
    if left.val < right.val:
        h.next = left
        left = left.next
    else:
        h.next = right
        right = right.next
    h = h.next

if left is not null:
    h.next = left
else:
    h.next = right

return res.next
```

Q148 Code.py

```
class ListNode:
    def __init__(self, val=0, next=None):
        self.val = val
        self.next = next

class Solution:
    def sortList(self, head: ListNode) -> ListNode:
        if not head or not head.next:
            return head

        slow, fast = head, head.next
        while fast and fast.next:
            fast, slow = fast.next.next, slow.next
        mid, slow.next = slow.next, None

        left, right = self.sortList(head), self.sortList(mid)


        h = res = ListNode(0)
        while left and right:
            if left.val < right.val:
                h.next, left = left, left.next
            else:
                h.next, right = right, right.next
            h = h.next

        h.next = left if left else right

        return res.next

def create_linked_list(arr):
```

```
    if not arr:
        return None
    head = ListNode(arr[0])
    current = head
    for val in arr[1:]:
        current.next = ListNode(val)
        current = current.next
    return head


def print_linked_list(node):
    result = []
    while node:
        result.append(node.val)
        node = node.next
    return result


arr = [-1,5,3,4,0]
head = create_linked_list(arr)
solution = Solution()
sorted_head = solution.sortList(head)
print(print_linked_list(sorted_head))
```

```
[-1, 0, 3, 4, 5]
```