708. Insert into a Sorted Circular Linked List

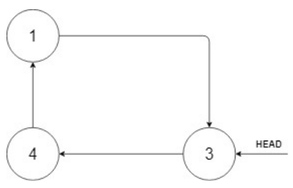
Medium 738 512 Add to List Share

Given a Circular Linked List node, which is sorted in ascending order, write a function to insert a value `insertVal` into the list such that it remains a sorted circular list. The given node can be a reference to any single node in the list and may not necessarily be the smallest value in the circular list.

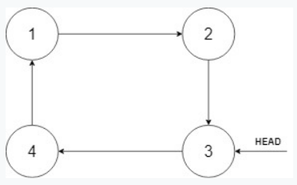
If there are multiple suitable places for insertion, you may choose any place to insert the new value. After the insertion, the circular list should remain sorted.

If the list is empty (i.e., the given node is `null`), you should create a new single circular list and return the reference to that single node. Otherwise, you should return the originally given node.

Example 1:




```
Input: head = [3,4,1], insertVal = 2
Output: [3,4,1,2]
Explanation: In the figure above, there is a sorted circular list of three elements. You are given a reference to the node with value 3, and we need to insert 2 into the list. The new node should be inserted between node 1 and node 3. After the insertion, the list should look like this, and we should still return node 3.
```



Example 2:

```
Input: head = [], insertVal = 1
Output: [1]
Explanation: The list is empty (given head is null). We create a new single circular list and return the reference to that single node.
```

Example 3:

```
Input: head = [1], insertVal = 0
Output: [1,0]
```

Constraints:

- `0 <= Number of Nodes <= 5 * 10^4`
- `-10^6 <= Node.val, insertVal <= 10^6`

Accepted 86,781 | Submissions 258,781

Seen this question in a real interview before? Yes No

Companies

0 ~ 6 months 6 months ~ 1 year 1 year ~ 2 years

Facebook | 15 Google | 5 Microsoft | 5

Related Topics

Linked List

Similar Questions

Insertion Sort List Medium

Java Autocomplete

```java
class Solution {
  public Node insert(Node head, int insertVal) {
    if (head == null) {
      Node newNode = new Node(insertVal, null);
      newNode.next = newNode;
      return newNode;
    }

    Node prev = head;
    Node curr = head.next;
    boolean toInsert = false;

    do {
      if (prev.val <= insertVal && insertVal <= curr.val) {
        // Case 1).
        toInsert = true;
      } else if (prev.val > curr.val) {
        // Case 2).
        if (insertVal >= prev.val || insertVal <= curr.val)
          toInsert = true;
      }

      if (toInsert) {
        prev.next = new Node(insertVal, curr);
        return head;
      }

      prev = curr;
      curr = curr.next;
    } while (prev != head);

    // Case 3).
    prev.next = new Node(insertVal, curr);
    return head;
  }
}
```