Problem 708: Insert into a Sorted Circular Linked List

Problem Information

Difficulty: Medium
Acceptance Rate: 0.00%
Paid Only: No

Problem Description

Given a Circular Linked List node, which is sorted in non-descending 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.

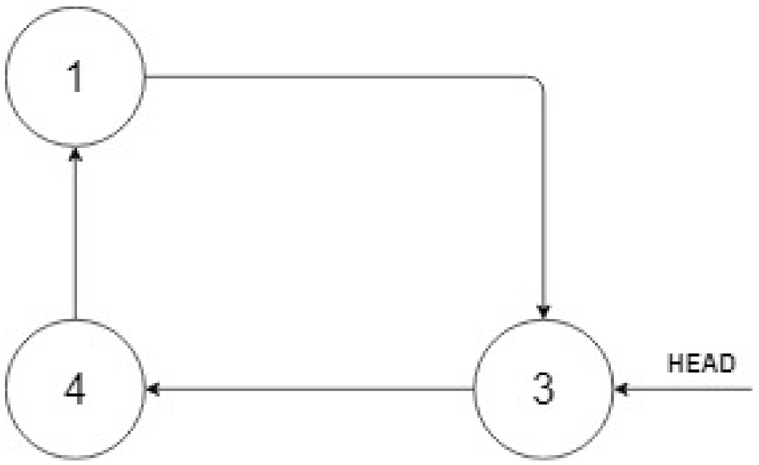
If there are multiple suitable places for insertion, you may choose any place to insert the new value. After the insertion, the circular list should remain sorted.

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:

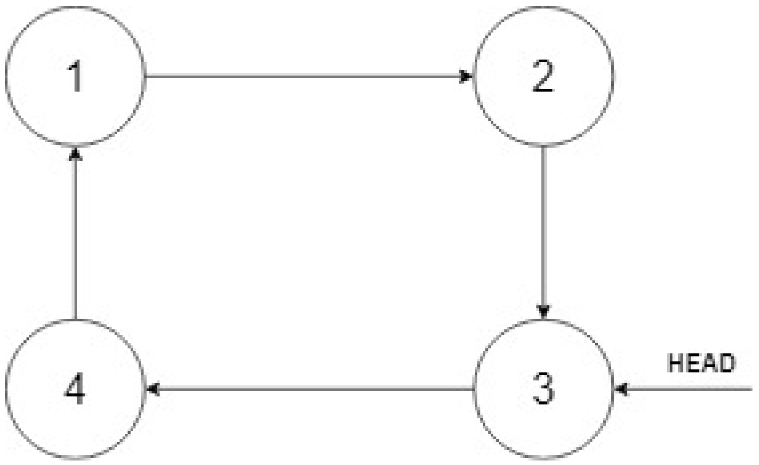
head = [3,4,1], insertVal = 2

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:

The number of nodes in the list is in the range

[0, 5 * 10

4

]

.

-10

6

<= Node.val, insertVal <= 10

6

Code Snippets

C++:

```
/*
// Definition for a Node.
class Node {
public:
int val;
Node* next;

Node() {}

Node(int _val) {
val = _val;
next = NULL;
}

Node(int _val, Node* _next) {
val = _val;
next = _next;
}
};
*/

class Solution {
public:
Node* insert(Node* head, int insertVal) {

}
};
```

Java:

```
/*
// Definition for a Node.
class Node {
public int val;
public Node next;

public Node() {}

public Node(int _val) {
val = _val;
}

public Node(int _val, Node _next) {
val = _val;
next = _next;
}
};
*/

class Solution {
public Node insert(Node head, int insertVal) {

}
}
```

Python3:

```
"""
# Definition for a Node.
class Node:
def __init__(self, val=None, next=None):
self.val = val
self.next = next
"""

class Solution:
def insert(self, head: 'Optional[Node]', insertVal: int) -> 'Node':
```

Python:

```
"""
# Definition for a Node.
class Node(object):
```

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

class Solution(object):
def insert(self, head, insertVal):
"""
:type head: Node
:type insertVal: int
:rtype: Node
"""
```

JavaScript:

```
/**
* // Definition for a _Node.
* function _Node(val, next) {
* this.val = val;
* this.next = next;
* };
*/

/**
* @param {_Node} head
* @param {number} insertVal
* @return {_Node}
*/
var insert = function(head, insertVal) {

};
```

TypeScript:

```
/**
* Definition for _Node.
* class _Node {
* val: number
* next: _Node | null
*
* constructor(val?: number, next?: _Node) {
* this.val = (val===undefined ? 0 : val);
* this.next = (next===undefined ? null : next);
* }
* }
*/



function insert(head: _Node | null, insertVal: number): _Node | null {

}
```

C#:

```
/*
// Definition for a Node.
public class Node {
public int val;
public Node next;

public Node() {}

public Node(int _val) {
val = _val;
next = null;
}

public Node(int _val, Node _next) {
val = _val;
next = _next;
}
}
*/

public class Solution {
public Node Insert(Node head, int insertVal) {

}
}
```

C:

```
/**
* Definition for a Node.
* struct Node {
* int val;
* struct Node *next;
* };
*/

struct Node* insert(struct Node* head, int insertVal) {

}
```

Go:

```
/**
* Definition for a Node.
* type Node struct {
* Val int
* Next *Node
* }
*/

func insert(aNode *Node, x int) *Node {

}
```

Kotlin:

```
/**
* Definition for a Node.
* class Node(var `val`: Int) {
* var next: Node? = null
* }
*/

class Solution {
fun insert(head: Node?, insertVal: Int): Node? {

}
}
```

Swift:

```
/**
* Definition for a Node.
* public class Node {
* public var val: Int
* public var next: Node?
* public init(_ val: Int) {
* self.val = val
* self.next = nil
* }
* }
*/

class Solution {
func insert(_ head: Node?, _ insertVal: Int) -> Node? {

}
}
```

Ruby:

```
# Definition for a Node.
# class Node
# attr_accessor :val, :next
# def initialize(val=nil, next_=nil)
# @val = val
# @next = next_
# end
# end

# @param {Node} head
# @param {Integer} insertVal
# @return {Node}
def insert(head, insertVal)

end
```

PHP:

```
/**
* Definition for a Node.
* class Node {
* public $val = null;
* public $next = null;
* function __construct($val = 0) {
* $this->val = $val;
* $this->next = null;
* }
* }
*/

class Solution {
/**
* @param Node $root
* @param Integer $insertVal
* @return Node
*/
function insert($head, $insertVal) {

}
}
```

Scala:

```
/**
* Definition for a Node.
* class Node(var _value: Int) {
* var value: Int = _value
* var next: Node = null
* }
*/

object Solution {
def insert(head: Node, insertVal: Int): Node = {

}
}
```

Solutions

C++ Solution:

```
/*
* Problem: Insert into a Sorted Circular Linked List
* Difficulty: Medium
* Tags: sort, linked_list
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

/*
// Definition for a Node.
class Node {
public:
int val;
Node* next;

Node() {}

Node(int _val) {
val = _val;
next = NULL;
}

Node(int _val, Node* _next) {
val = _val;
next = _next;
}
};
*/

class Solution {
public:
Node* insert(Node* head, int insertVal) {

}
};
```

Java Solution:

```
/**
* Problem: Insert into a Sorted Circular Linked List
* Difficulty: Medium
* Tags: sort, linked_list
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

/*
// Definition for a Node.
class Node {
public int val;
public Node next;

public Node() {
// TODO: Implement optimized solution
return 0;
}

public Node(int _val) {
val = _val;
}

public Node(int _val, Node _next) {
val = _val;
next = _next;
}
};
*/

class Solution {
public Node insert(Node head, int insertVal) {

}
}
```

Python3 Solution:

```
"""
Problem: Insert into a Sorted Circular Linked List
Difficulty: Medium
Tags: sort, linked_list
```

```
Approach: Optimized algorithm based on problem constraints
Time Complexity: O(n) to O(n^2) depending on approach
Space Complexity: O(1) to O(n) depending on approach
"""

"""
# Definition for a Node.
class Node:
def __init__(self, val=None, next=None):
self.val = val
self.next = next
"""

class Solution:
def insert(self, head: 'Optional[Node]', insertVal: int) -> 'Node':
# TODO: Implement optimized solution
pass
```

Python Solution:

```
"""
# Definition for a Node.
class Node(object):
def __init__(self, val=None, next=None):
self.val = val
self.next = next
"""

class Solution(object):
def insert(self, head, insertVal):
"""
:type head: Node
:type insertVal: int
:rtype: Node
"""
```

JavaScript Solution:

```
/**
* Problem: Insert into a Sorted Circular Linked List
* Difficulty: Medium
* Tags: sort, linked_list
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* // Definition for a _Node.
* function _Node(val, next) {
* this.val = val;
* this.next = next;
* };
*/

/**
* @param {_Node} head
* @param {number} insertVal
* @return {_Node}
*/
var insert = function(head, insertVal) {

};
```

TypeScript Solution:

```
/**
* Problem: Insert into a Sorted Circular Linked List
* Difficulty: Medium
* Tags: sort, linked_list
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* Definition for _Node.
* class _Node {
* val: number
* next: _Node | null
*
* constructor(val?: number, next?: _Node) {
* this.val = (val===undefined ? 0 : val);
* this.next = (next===undefined ? null : next);
* }
* }
*/


function insert(head: _Node | null, insertVal: number): _Node | null {

}
```

C# Solution:

```
/*
* Problem: Insert into a Sorted Circular Linked List
* Difficulty: Medium
* Tags: sort, linked_list
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

/*
// Definition for a Node.
public class Node {
public int val;
public Node next;

public Node() {}

public Node(int _val) {
val = _val;
next = null;
}

public Node(int _val, Node _next) {
val = _val;
next = _next;
}
}
*/

public class Solution {
public Node Insert(Node head, int insertVal) {

}
}
```

C Solution:

```
/*
* Problem: Insert into a Sorted Circular Linked List
* Difficulty: Medium
* Tags: sort, linked_list
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* Definition for a Node.
* struct Node {
* int val;
* struct Node *next;
* };
*/

struct Node* insert(struct Node* head, int insertVal) {

}
```

Go Solution:

```
// Problem: Insert into a Sorted Circular Linked List
// Difficulty: Medium
// Tags: sort, linked_list
//
```

```
// Approach: Optimized algorithm based on problem constraints
// Time Complexity: O(n) to O(n^2) depending on approach
// Space Complexity: O(1) to O(n) depending on approach

/**
* Definition for a Node.
* type Node struct {
* Val int
* Next *Node
* }
*/

func insert(aNode *Node, x int) *Node {

}
```

Kotlin Solution:

```
/**
* Definition for a Node.
* class Node(var `val`: Int) {
* var next: Node? = null
* }
*/

class Solution {
fun insert(head: Node?, insertVal: Int): Node? {

}
}
```

Swift Solution:

```
/**
* Definition for a Node.
* public class Node {
* public var val: Int
* public var next: Node?
* public init(_ val: Int) {
* self.val = val
* self.next = nil
* }
* }
*/

class Solution {
func insert(_ head: Node?, _ insertVal: Int) -> Node? {

}
}
```

Ruby Solution:

```
# Definition for a Node.
# class Node
# attr_accessor :val, :next
# def initialize(val=nil, next_=nil)
# @val = val
# @next = next_
# end
# end

# @param {Node} head
# @param {Integer} insertVal
# @return {Node}
def insert(head, insertVal)

end
```

PHP Solution:

```
/**
* Definition for a Node.
* class Node {
* public $val = null;
* public $next = null;
* function __construct($val = 0) {
* $this->val = $val;
* $this->next = null;
* }
* }
*/
```

```
class Solution {
/**
* @param Node $root
* @param Integer $insertVal
* @return Node
*/
function insert($head, $insertVal) {

}
}
```

Scala Solution:

```
/**
* Definition for a Node.
* class Node(var _value: Int) {
* var value: Int = _value
* var next: Node = null
* }
*/

object Solution {
def insert(head: Node, insertVal: Int): Node = {

}
}
```