Problem 21: Merge Two Sorted Lists

Problem Information

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

Problem Description

You are given the heads of two sorted linked lists

list1

and

list2

.

Merge the two lists into one

sorted

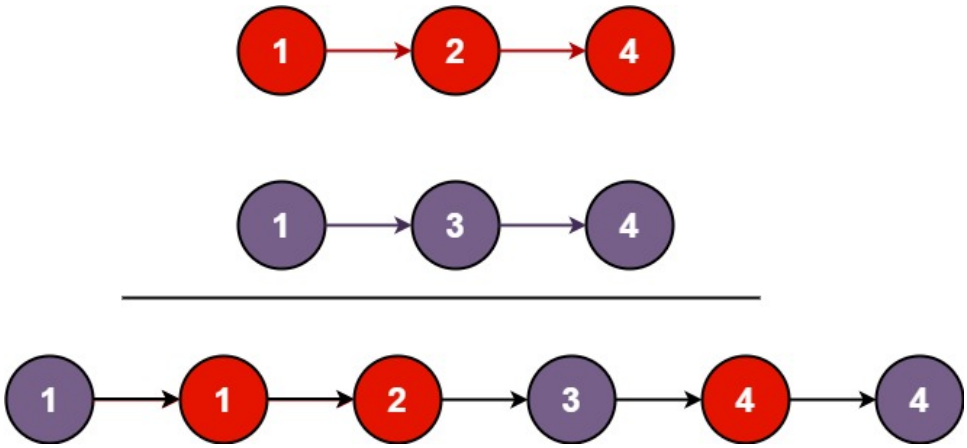
list. The list should be made by splicing together the nodes of the first two lists.

Return

the head of the merged linked list

.

Example 1:



Input:

list1 = [1,2,4], list2 = [1,3,4]

Output:

[1,1,2,3,4,4]

Example 2:

Input:

list1 = [], list2 = []

Output:

[]

Example 3:

Input:

list1 = [], list2 = [0]

Output:

[0]

Constraints:

The number of nodes in both lists is in the range

[0, 50]

.

-100 <= Node.val <= 100

Both

list1

and

list2

are sorted in

non-decreasing

order.

Code Snippets

C++:

```
/**
* Definition for singly-linked list.
* struct ListNode {
* int val;
* ListNode *next;
* ListNode() : val(0), next(nullptr) {}
* ListNode(int x) : val(x), next(nullptr) {}
* ListNode(int x, ListNode *next) : val(x), next(next) {}
* };
*/
class Solution {
public:
ListNode* mergeTwoLists(ListNode* list1, ListNode* list2) {
```

```
}
};
```

Java:

```
/**
* Definition for singly-linked list.
* public class ListNode {
* int val;
* ListNode next;
* ListNode() {}
* ListNode(int val) { this.val = val; }
* ListNode(int val, ListNode next) { this.val = val; this.next = next; }
* }
*/
class Solution {
public ListNode mergeTwoLists(ListNode list1, ListNode list2) {

}
}
```

Python3:

```
# Definition for singly-linked list.
# class ListNode:
# def __init__(self, val=0, next=None):
# self.val = val
# self.next = next
class Solution:
def mergeTwoLists(self, list1: Optional[ListNode], list2: Optional[ListNode])
-> Optional[ListNode]:
```

Python:

```
# Definition for singly-linked list.
# class ListNode(object):
# def __init__(self, val=0, next=None):
# self.val = val
# self.next = next
class Solution(object):
def mergeTwoLists(self, list1, list2):
```

```
"""
:type list1: Optional[ListNode]
:type list2: Optional[ListNode]
:rtype: Optional[ListNode]
"""
```

JavaScript:

```
/**
* Definition for singly-linked list.
* function ListNode(val, next) {
* this.val = (val===undefined ? 0 : val)
* this.next = (next===undefined ? null : next)
* }
*/
/**
* @param {ListNode} list1
* @param {ListNode} list2
* @return {ListNode}
*/
var mergeTwoLists = function(list1, list2) {

};
```

TypeScript:

```
/**
* Definition for singly-linked list.
* class ListNode {
* val: number
* next: ListNode | null
* constructor(val?: number, next?: ListNode | null) {
* this.val = (val===undefined ? 0 : val)
* this.next = (next===undefined ? null : next)
* }
* }
*/

function mergeTwoLists(list1: ListNode | null, list2: ListNode | null):
ListNode | null {

};
```

C#:

```
/**
* Definition for singly-linked list.
* public class ListNode {
* public int val;
* public ListNode next;
* public ListNode(int val=0, ListNode next=null) {
* this.val = val;
* this.next = next;
* }
* }
*/
public class Solution {
public ListNode MergeTwoLists(ListNode list1, ListNode list2) {

}
}
```

C:

```
/**
* Definition for singly-linked list.
* struct ListNode {
* int val;
* struct ListNode *next;
* };
*/
struct ListNode* mergeTwoLists(struct ListNode* list1, struct ListNode*
list2) {

}
```

Go:

```
/**
* Definition for singly-linked list.
* type ListNode struct {
* Val int
* Next *ListNode
* }
*/
func mergeTwoLists(list1 *ListNode, list2 *ListNode) *ListNode {
```

```
}
```

Kotlin:

```
/**
* Example:
* var li = ListNode(5)
* var v = li.`val`
* Definition for singly-linked list.
* class ListNode(var `val`: Int) {
* var next: ListNode? = null
* }
*/
class Solution {
fun mergeTwoLists(list1: ListNode?, list2: ListNode?): ListNode? {

}
}
```

Swift:

```
/**
* Definition for singly-linked list.
* public class ListNode {
* public var val: Int
* public var next: ListNode?
* public init() { self.val = 0; self.next = nil; }
* public init(_ val: Int) { self.val = val; self.next = nil; }
* public init(_ val: Int, _ next: ListNode?) { self.val = val; self.next =
next; }
* }
*/
class Solution {
func mergeTwoLists(_ list1: ListNode?, _ list2: ListNode?) -> ListNode? {

}
}
```

Rust:

```
// Definition for singly-linked list.
// #[derive(PartialEq, Eq, Clone, Debug)]
// pub struct ListNode {
// pub val: i32,
// pub next: Option<Box<ListNode>>
// }
//
// impl ListNode {
// #[inline]
// fn new(val: i32) -> Self {
// ListNode {
// next: None,
// val
// }
// }
// }
impl Solution {
pub fn merge_two_lists(list1: Option<Box<ListNode>>, list2:
Option<Box<ListNode>>) -> Option<Box<ListNode>> {

}
}
```

Ruby:

```
# Definition for singly-linked list.
# class ListNode
# attr_accessor :val, :next
# def initialize(val = 0, _next = nil)
# @val = val
# @next = _next
# end
# end
# @param {ListNode} list1
# @param {ListNode} list2
# @return {ListNode}
def merge_two_lists(list1, list2)

end
```

PHP:

```
/**
* Definition for a singly-linked list.
* class ListNode {
* public $val = 0;
* public $next = null;
* function __construct($val = 0, $next = null) {
* $this->val = $val;
* $this->next = $next;
* }
* }
*/
class Solution {

/**
* @param ListNode $list1
* @param ListNode $list2
* @return ListNode
*/
function mergeTwoLists($list1, $list2) {

}
}
```

Dart:

```
/**
* Definition for singly-linked list.
* class ListNode {
* int val;
* ListNode? next;
* ListNode([this.val = 0, this.next]);
* }
*/
class Solution {
ListNode? mergeTwoLists(ListNode? list1, ListNode? list2) {

}
}
```

Scala:

```
/**
* Definition for singly-linked list.
```

```
* class ListNode(_x: Int = 0, _next: ListNode = null) {
* var next: ListNode = _next
* var x: Int = _x
* }
*/
object Solution {
def mergeTwoLists(list1: ListNode, list2: ListNode): ListNode = {

}
}
```

Elixir:

```
# Definition for singly-linked list.
#
# defmodule ListNode do
# @type t :: %__MODULE__{
# val: integer,
# next: ListNode.t() | nil
# }
# defstruct val: 0, next: nil
# end

defmodule Solution do
@spec merge_two_lists(list1 :: ListNode.t | nil, list2 :: ListNode.t | nil)
:: ListNode.t | nil
def merge_two_lists(list1, list2) do

end
end
```

Erlang:

```
%% Definition for singly-linked list.
%%
%% -record(list_node, {val = 0 :: integer(),
%% next = null :: 'null' | #list_node{}}).

-spec merge_two_lists(List1 :: #list_node{} | null, List2 :: #list_node{} |
null) -> #list_node{} | null.
merge_two_lists(List1, List2) ->
.
```

Racket:

```
; Definition for singly-linked list:
#|

; val : integer?
; next : (or/c list-node? #f)
(struct list-node
(val next) #:mutable #:transparent)

; constructor
(define (make-list-node [val 0])
(list-node val #f))

|#

(define/contract (merge-two-lists list1 list2)
(-> (or/c list-node? #f) (or/c list-node? #f) (or/c list-node? #f))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Merge Two Sorted Lists
* Difficulty: Easy
* 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 singly-linked list.
* struct ListNode {
* int val;
* ListNode *next;
* ListNode() : val(0), next(nullptr) {}
* ListNode(int x) : val(x), next(nullptr) {}
```

```
* ListNode(int x, ListNode *next) : val(x), next(next) {}
* };
*/
class Solution {
public:
ListNode* mergeTwoLists(ListNode* list1, ListNode* list2) {

}
};
```

Java Solution:

```
/**
* Problem: Merge Two Sorted Lists
* Difficulty: Easy
* 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 singly-linked list.
* public class ListNode {
* int val;
* ListNode next;
* ListNode() {
// TODO: Implement optimized solution
return 0;
}
* ListNode(int val) { this.val = val; }
* ListNode(int val, ListNode next) { this.val = val; this.next = next; }
* }
*/
class Solution {
public ListNode mergeTwoLists(ListNode list1, ListNode list2) {

}
}
```

Python3 Solution:

```
"""
Problem: Merge Two Sorted Lists
Difficulty: Easy
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 singly-linked list.
# class ListNode:
# def __init__(self, val=0, next=None):
# self.val = val
# self.next = next
class Solution:
def mergeTwoLists(self, list1: Optional[ListNode], list2: Optional[ListNode])
-> Optional[ListNode]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
# Definition for singly-linked list.
# class ListNode(object):
# def __init__(self, val=0, next=None):
# self.val = val
# self.next = next
class Solution(object):
def mergeTwoLists(self, list1, list2):
"""
:type list1: Optional[ListNode]
:type list2: Optional[ListNode]
:rtype: Optional[ListNode]
"""
```

JavaScript Solution:

```
/**
* Problem: Merge Two Sorted Lists
* Difficulty: Easy
```

```
* 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 singly-linked list.
* function ListNode(val, next) {
* this.val = (val===undefined ? 0 : val)
* this.next = (next===undefined ? null : next)
* }
*/
/**
* @param {ListNode} list1
* @param {ListNode} list2
* @return {ListNode}
*/
var mergeTwoLists = function(list1, list2) {

};
```

TypeScript Solution:

```
/**
* Problem: Merge Two Sorted Lists
* Difficulty: Easy
* 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 singly-linked list.
* class ListNode {
* val: number
* next: ListNode | null
* constructor(val?: number, next?: ListNode | null) {
```

```
* this.val = (val===undefined ? 0 : val)
* this.next = (next===undefined ? null : next)
* }
* }
*/

function mergeTwoLists(list1: ListNode | null, list2: ListNode | null):
ListNode | null {

};
```

C# Solution:

```
/*
* Problem: Merge Two Sorted Lists
* Difficulty: Easy
* 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 singly-linked list.
* public class ListNode {
* public int val;
* public ListNode next;
* public ListNode(int val=0, ListNode next=null) {
* this.val = val;
* this.next = next;
* }
* }
*/
public class Solution {
public ListNode MergeTwoLists(ListNode list1, ListNode list2) {

}
}
```

C Solution:

```c
/*
* Problem: Merge Two Sorted Lists
* Difficulty: Easy
* 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 singly-linked list.
* struct ListNode {
* int val;
* struct ListNode *next;
* };
*/
struct ListNode* mergeTwoLists(struct ListNode* list1, struct ListNode*
list2) {

}
```

Go Solution:

```go
// Problem: Merge Two Sorted Lists
// Difficulty: Easy
// 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 singly-linked list.
* type ListNode struct {
* Val int
* Next *ListNode
* }
*/
func mergeTwoLists(list1 *ListNode, list2 *ListNode) *ListNode {

}
```

Kotlin Solution:

```
/**
* Example:
* var li = ListNode(5)
* var v = li.`val`
* Definition for singly-linked list.
* class ListNode(var `val`: Int) {
* var next: ListNode? = null
* }
*/
class Solution {
fun mergeTwoLists(list1: ListNode?, list2: ListNode?): ListNode? {

}
}
```

Swift Solution:

```
/**
* Definition for singly-linked list.
* public class ListNode {
* public var val: Int
* public var next: ListNode?
* public init() { self.val = 0; self.next = nil; }
* public init(_ val: Int) { self.val = val; self.next = nil; }
* public init(_ val: Int, _ next: ListNode?) { self.val = val; self.next =
next; }
* }
*/
class Solution {
func mergeTwoLists(_ list1: ListNode?, _ list2: ListNode?) -> ListNode? {

}
}
```

Rust Solution:

```
// Problem: Merge Two Sorted Lists
// Difficulty: Easy
// 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 singly-linked list.
// #[derive(PartialEq, Eq, Clone, Debug)]
// pub struct ListNode {
// pub val: i32,
// pub next: Option<Box<ListNode>>
// }
//
// impl ListNode {
// #[inline]
// fn new(val: i32) -> Self {
// ListNode {
// next: None,
// val
// }
// }
// }
impl Solution {
pub fn merge_two_lists(list1: Option<Box<ListNode>>, list2:
Option<Box<ListNode>>) -> Option<Box<ListNode>> {

}
}
```

Ruby Solution:

```
# Definition for singly-linked list.
# class ListNode
# attr_accessor :val, :next
# def initialize(val = 0, _next = nil)
# @val = val
# @next = _next
# end
# end
# @param {ListNode} list1
# @param {ListNode} list2
# @return {ListNode}
def merge_two_lists(list1, list2)
```

```
end
```

PHP Solution:

```
/**
* Definition for a singly-linked list.
* class ListNode {
* public $val = 0;
* public $next = null;
* function __construct($val = 0, $next = null) {
* $this->val = $val;
* $this->next = $next;
* }
* }
*/
class Solution {

/**
* @param ListNode $list1
* @param ListNode $list2
* @return ListNode
*/
function mergeTwoLists($list1, $list2) {

}
}
```

Dart Solution:

```
/**
* Definition for singly-linked list.
* class ListNode {
* int val;
* ListNode? next;
* ListNode([this.val = 0, this.next]);
* }
*/
class Solution {
ListNode? mergeTwoLists(ListNode? list1, ListNode? list2) {
```

```
}
}
```

Scala Solution:

```
/**
* Definition for singly-linked list.
* class ListNode(_x: Int = 0, _next: ListNode = null) {
* var next: ListNode = _next
* var x: Int = _x
* }
*/
object Solution {
def mergeTwoLists(list1: ListNode, list2: ListNode): ListNode = {

}
}
```

Elixir Solution:

```
# Definition for singly-linked list.
#
# defmodule ListNode do
# @type t :: %__MODULE__{
# val: integer,
# next: ListNode.t() | nil
# }
# defstruct val: 0, next: nil
# end

defmodule Solution do
@spec merge_two_lists(list1 :: ListNode.t | nil, list2 :: ListNode.t | nil)
:: ListNode.t | nil
def merge_two_lists(list1, list2) do

end
end
```

Erlang Solution:

```
%% Definition for singly-linked list.
%%
%% -record(list_node, {val = 0 :: integer(),
%% next = null :: 'null' | #list_node{}}).

-spec merge_two_lists(List1 :: #list_node{} | null, List2 :: #list_node{} |
null) -> #list_node{} | null.
merge_two_lists(List1, List2) ->
.
```

Racket Solution:

```
; Definition for singly-linked list:
#|

; val : integer?
; next : (or/c list-node? #f)
(struct list-node
(val next) #:mutable #:transparent)

; constructor
(define (make-list-node [val 0])
(list-node val #f))

|#

(define/contract (merge-two-lists list1 list2)
(-> (or/c list-node? #f) (or/c list-node? #f) (or/c list-node? #f))
)
```