Problem 23: Merge k Sorted Lists

Problem Information

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

Problem Description

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 linked list: 1->1->2->3->4->4->5->6

Example 2:

Input:

lists = []

Output:

[]

Example 3:

Input:

lists = [[]]

Output:

[]

Constraints:

k == lists.length

0 <= k <= 10

4

0 <= lists[i].length <= 500

-10

4

<= lists[i][j] <= 10

4

lists[i]

is sorted in

ascending order

.

The sum of

lists[i].length

will not exceed

10

4

.

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* mergeKLists(vector<ListNode*>& lists) {

}
```

```
};
```

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 mergeKLists(ListNode[] lists) {

}
}
```

Python3:

```
# Definition for singly-linked list.
# 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]:
```

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 mergeKLists(self, lists):
"""
:type lists: List[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[]} lists
* @return {ListNode}
*/
var mergeKLists = function(lists) {

};
```

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 mergeKLists(lists: Array<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 MergeKLists(ListNode[] lists) {

}
}
```

C:

```
/**
* Definition for singly-linked list.
* struct ListNode {
* int val;
* struct ListNode *next;
* };
*/
struct ListNode* mergeKLists(struct ListNode** lists, int listsSize) {

}
```

Go:

```
/**
* Definition for singly-linked list.
* type ListNode struct {
* Val int
* Next *ListNode
* }
*/
func mergeKLists(lists []*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 mergeKLists(lists: Array<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 mergeKLists(_ lists: [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_k_lists(lists: Vec<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[]} lists
# @return {ListNode}
def merge_k_lists(lists)

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[] $lists
* @return ListNode
*/
function mergeKLists($lists) {

}
}
```

Dart:

```
/**
* Definition for singly-linked list.
* class ListNode {
* int val;
* ListNode? next;
* ListNode([this.val = 0, this.next]);
* }
*/
class Solution {
ListNode? mergeKLists(List<ListNode?> lists) {

}
}
```

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 mergeKLists(lists: Array[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_k_lists(lists :: [ListNode.t | nil]) :: ListNode.t | nil
def merge_k_lists(lists) do

end
end
```

Erlang:

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

-spec merge_k_lists(Lists :: [#list_node{} | null]) -> #list_node{} | null.
merge_k_lists(Lists) ->
.
```

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-k-lists lists)
(-> (listof (or/c list-node? #f)) (or/c list-node? #f))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Merge k Sorted Lists
* Difficulty: Hard
* Tags: array, sort, linked_list, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* 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* mergeKLists(vector<ListNode*>& lists) {

}
```

```
};
```

Java Solution:

```
/**
* Problem: Merge k Sorted Lists
* Difficulty: Hard
* Tags: array, sort, linked_list, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* 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 mergeKLists(ListNode[] lists) {

}
}
```

Python3 Solution:

```
"""
Problem: Merge k Sorted Lists
Difficulty: Hard
Tags: array, sort, linked_list, queue, heap

Approach: Use two pointers or sliding window technique
```

```
Time Complexity: O(n) or O(n log n)
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 mergeKLists(self, lists: List[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 mergeKLists(self, lists):
"""
:type lists: List[Optional[ListNode]]
:rtype: Optional[ListNode]
"""
```

JavaScript Solution:

```
/**
* Problem: Merge k Sorted Lists
* Difficulty: Hard
* Tags: array, sort, linked_list, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* 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[]} lists
* @return {ListNode}
*/
var mergeKLists = function(lists) {

};
```

TypeScript Solution:

```
/**
* Problem: Merge k Sorted Lists
* Difficulty: Hard
* Tags: array, sort, linked_list, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* 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 mergeKLists(lists: Array<ListNode | null>): ListNode | null {

};
```

C# Solution:

```
/*
* Problem: Merge k Sorted Lists
* Difficulty: Hard
* Tags: array, sort, linked_list, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* 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 MergeKLists(ListNode[] lists) {

}
}
```

C Solution:

```
/*
* Problem: Merge k Sorted Lists
* Difficulty: Hard
* Tags: array, sort, linked_list, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
```

```
* Definition for singly-linked list.
* struct ListNode {
* int val;
* struct ListNode *next;
* };
*/
struct ListNode* mergeKLists(struct ListNode** lists, int listsSize) {

}
```

Go Solution:

```
// Problem: Merge k Sorted Lists
// Difficulty: Hard
// Tags: array, sort, linked_list, queue, heap
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

/**
* Definition for singly-linked list.
* type ListNode struct {
* Val int
* Next *ListNode
* }
*/
func mergeKLists(lists []*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 mergeKLists(lists: Array<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 mergeKLists(_ lists: [ListNode?]) -> ListNode? {

}
}
```

Rust Solution:

```
// Problem: Merge k Sorted Lists
// Difficulty: Hard
// Tags: array, sort, linked_list, queue, heap
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// 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_k_lists(lists: Vec<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[]} lists
# @return {ListNode}
def merge_k_lists(lists)

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[] $lists
* @return ListNode
*/
function mergeKLists($lists) {

}
}
```

Dart Solution:

```
/**
* Definition for singly-linked list.
* class ListNode {
* int val;
* ListNode? next;
* ListNode([this.val = 0, this.next]);
* }
*/
class Solution {
ListNode? mergeKLists(List<ListNode?> lists) {

}
}
```

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 mergeKLists(lists: Array[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_k_lists(lists :: [ListNode.t | nil]) :: ListNode.t | nil
def merge_k_lists(lists) do

end
end
```

Erlang Solution:

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

-spec merge_k_lists(Lists :: [#list_node{} | null]) -> #list_node{} | null.
merge_k_lists(Lists) ->
.
```

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-k-lists lists)
(-> (listof (or/c list-node? #f)) (or/c list-node? #f))
)
```