Merge K sorted LL

Java

```java
import java.util.PriorityQueue;

class Node{
    int data;
    Node next;
    Node(int val){
        data=val;
        next=null;
    }
}


public class mergeKSortedList {

    //function to merge k sorted linked lists.
    static public Node merge(Node[] arr,int k){

        PriorityQueue<Node> pq=new PriorityQueue<>((a,b)->{
            return a.data-b.data;
        });

        //Pushing the head nodes of all
        //the k lists in 'pq'
        for(int i=0;i<k;i++){
            if(arr[i]!=null){
                pq.add(arr[i]);
            }
        }
        //Handling the case when k=0 or lists are empty.
        if(pq.isEmpty()) return null;

        //Creating dummy node
        Node dummy=new Node(-1);
        Node last=dummy;

        //Run while loop till priorityQueue(pq) is not empty
        while(!pq.isEmpty()){
            //Get the top element of 'pq'
            //which is the minimum of all existing nodes.
            Node curr=pq.remove();

            //Add top element of 'pq'
            // to the resultant merged list
            last.next=curr;
            last=last.next;


            // Check if there is a node
            // next to the 'top' node
            // in the list of which 'top'
            // node is a member
            if(curr.next!=null){
                pq.add(curr.next);
            }
        }
        //return the next of dummy node
        // which the actual head of merged list.
        return dummy.next;
    }

    //function to print the sorted linked list
    static void printList(Node head){
        while(head!=null){
            System.out.print(head.data+" -> ");
            head=head.next;
        }
        System.out.print("none");
    }

    public static void main(String[] args) {
        //Number of linked lists to be merged.
        int k=3;
        //An array of Node type storing head node of linked lists.
        Node[] arr=new Node[k];

        //Linked List 1: 3->5->7
        arr[0]=new Node(3);
        arr[0].next=new Node(5);
        arr[0].next.next=new Node(7);

        //Linked List 2: 0->6
        arr[1]=new Node(0);
        arr[1].next=new Node(6);

        //Linked List 3: 1->6->28
        arr[2]=new Node(1);
        arr[2].next=new Node(6);
        arr[2].next.next=new Node(28);

        //Calling the merge function to merge all the linked lists
        Node head=merge(arr,k);
        printList(head);
    }


}
```

C++

```
#include <bits/stdc++.h>
using namespace std;

struct Node {
      int data;
      struct Node* next;
};

struct Node* newNode(int data) {
      struct Node* new_node = new Node();
      new_node->data = data;
      new_node->next = NULL;

      return new_node;
}

// 'compare' function used to build
// up the priority queue
struct compare {
      bool operator()(
            struct Node* a, struct Node* b) {
            return a->data > b->data;
      }
};

// Function to merge k sorted linked lists
struct Node* mergeKSortedLists(
                  struct Node* arr[], int k)
{
      // Priority_queue 'pq' implemented
      // as min heap with the help of
      // 'compare' function
      priority_queue<Node*, vector<Node*>, compare> pq;

      // Push the head nodes of all
      // the k lists in 'pq'
```

```
        for (int i = 0; i < k; i++)
            if (arr[i] != NULL)
                pq.push(arr[i]);

        // Handles the case when k = 0
        // or lists have no elements in them
        if (pq.empty())
            return NULL;

        struct Node *dummy = newNode(0);
        struct Node *last = dummy;

        // Loop till 'pq' is not empty
        while (!pq.empty()) {
            // Get the top element of 'pq'
            struct Node* curr = pq.top();
            pq.pop();

            // Add the top element of 'pq'
            // to the resultant merged list
            last->next = curr;
            last = last->next;

            // Check if there is a node
            // next to the 'top' node
            // in the list of which 'top'
            // node is a member
            if (curr->next != NULL)

            // Push the next node of top node
            // in 'pq'
            pq.push(curr->next);
        }

        // Address of head node of the required merged list
        return dummy->next;
}

// Function to print the linked list
void printList(struct Node* head) {
        while (head != NULL) {
            cout << head->data << " ";
            head = head->next;
        }
}
```

```
int main() {
      // Number of linked lists
      int k = 3;
      // Number of elements in each list
      int n = 4;

      // An array of pointers storing the head nodes of the linked lists
      Node* arr[k];

      arr[0] = newNode(1);
      arr[0]->next = newNode(3);
      arr[0]->next->next = newNode(5);
      arr[0]->next->next->next = newNode(7);

      arr[1] = newNode(2);
      arr[1]->next = newNode(4);
      arr[1]->next->next = newNode(6);
      arr[1]->next->next->next = newNode(8);

      arr[2] = newNode(0);
      arr[2]->next = newNode(9);
      arr[2]->next->next = newNode(10);
      arr[2]->next->next->next = newNode(11);

      struct Node* head = mergeKSortedLists(arr, k);
      printList(head);

      return 0;
}
```