

Advanced Programming Assignment 3

Name- Vivek Singh

UID-22BCS11457

Section- 607-B

Qs 1. Print Linked List:

Code: class Solution {

public:

// Function to display the elements of a linked list in same line

void printList(Node *head) {

// your code goes here

Node* temp= head;

while(temp!=NULL){

cout<<temp->data<<" ";

temp=temp->next;

}

}

};

Submission:

The screenshot displays a coding platform interface with a dark theme. On the left, the 'Output Window' shows 'Compilation Results' for 'Custom Input: Y.O.G.J. (AI Bot)'. A green checkmark indicates 'Problem Solved Successfully'. Below this, statistics are shown: 'Test Cases Passed: 1112 / 1112', 'Attempts: Correct / Total: 2 / 3', and 'Accuracy: 66%'. The 'Time Taken' is '0.07'. A note at the bottom states: 'You get marks only for the first correct submission. If you solve the problem without viewing the full solution.' On the right, the code editor shows a C++ solution for 'Print Linked List'. The code defines a 'Node' struct and a 'Solution' class with a 'printList' method that traverses the linked list and prints its elements in a single line. The code is as follows:

```
1-  
19-  
20-  
21- struct Node {  
22-     int data;  
23-     struct Node* next;  
24- };  
25- Node(int a) {  
26-     data = a;  
27-     next = nullptr;  
28- }  
29-  
30-  
31-  
32-  
33- // Print elements of a linked list on console  
34- // Node pointer input could be NULL as well for empty list  
35-  
36- class Solution {  
37- public:  
38-     // Function to display the elements of a linked list in same line  
39-     void printList(Node *head) {  
40-         // your code goes here  
41-         Node* temp= head;  
42-         while(temp!=NULL){  
43-             cout<<temp->data<<" ";  
44-             temp=temp->next;  
45-         }  
46-     }  
47- };  
48-  
49-
```

Qs 2. Remove Duplicates from Sorted List:

Code: class Solution {

public:

ListNode* deleteDuplicates(ListNode* head) {

ListNode* temp=head;

while(temp && temp->next){

if(temp->val==temp->next->val){

ListNode* duplicate=temp->next;

temp->next=duplicate->next;

delete(duplicate);

}else{

temp=temp->next;

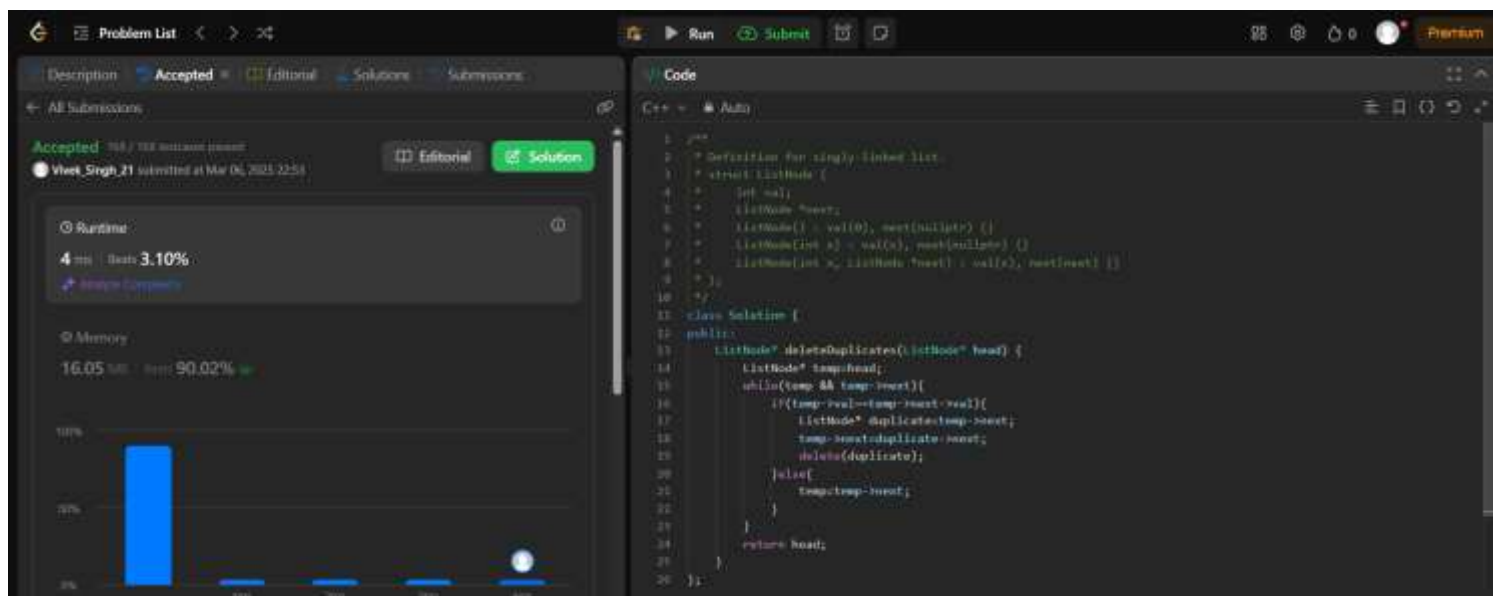
}

}

return head;

}};

Submission:



Qs 3. Reverse a linked list:

Code: class Solution {

public:

ListNode* reverseList(ListNode* head) {

ListNode* prev = nullptr;

ListNode* next = nullptr;

ListNode* curr = head;

while (curr != nullptr) {

next = curr->next;

curr->next = prev;

prev = curr;

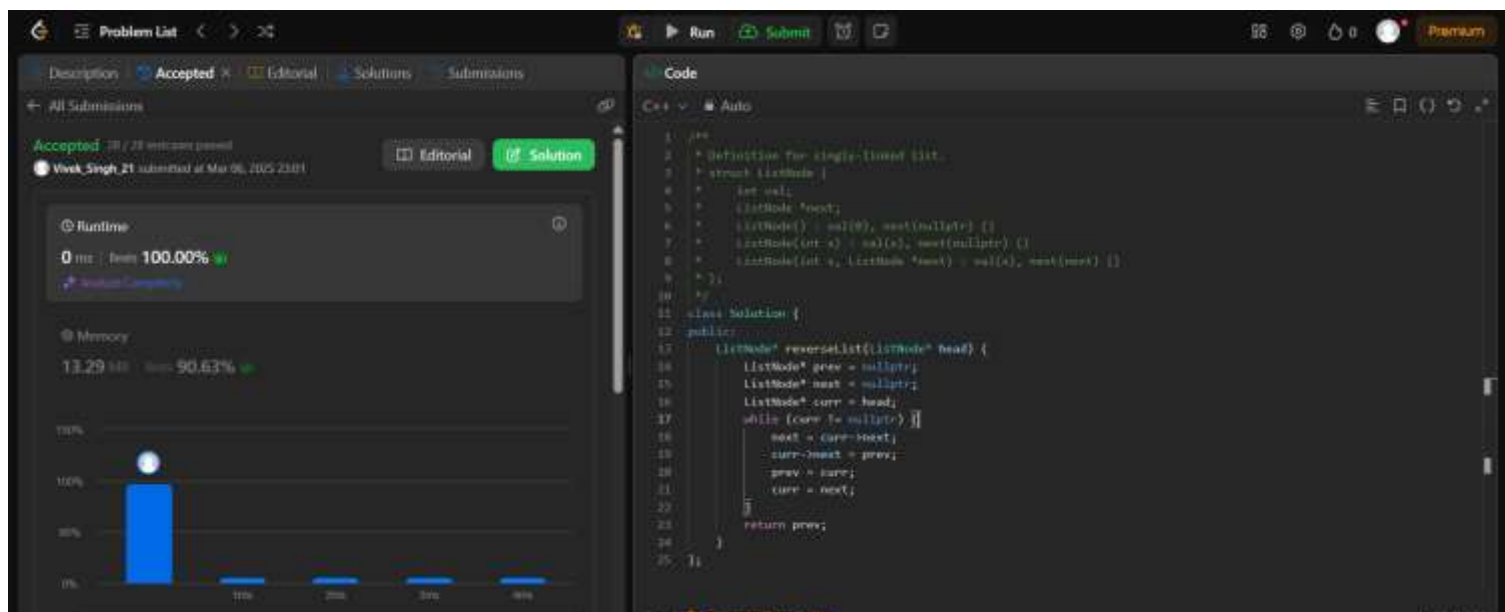
curr = next;

}

return prev;

}

};Submission:



Qs 4. Delete middle node of linked list:

Code: class Solution {

public:

```
ListNode* deleteMiddle(ListNode* head) {
```

```
    if(!head||!head->next){
```

```
        return NULL;
```

```
    }
```

```
    ListNode* slow=head;
```

```
    ListNode* fast=head;
```

```
    ListNode* pre=NULL;
```

```
    while(fast && fast->next){
```

```
        pre=slow;
```

```
        slow=slow->next;
```

```
        fast=fast->next->next;
```

```
    }
```

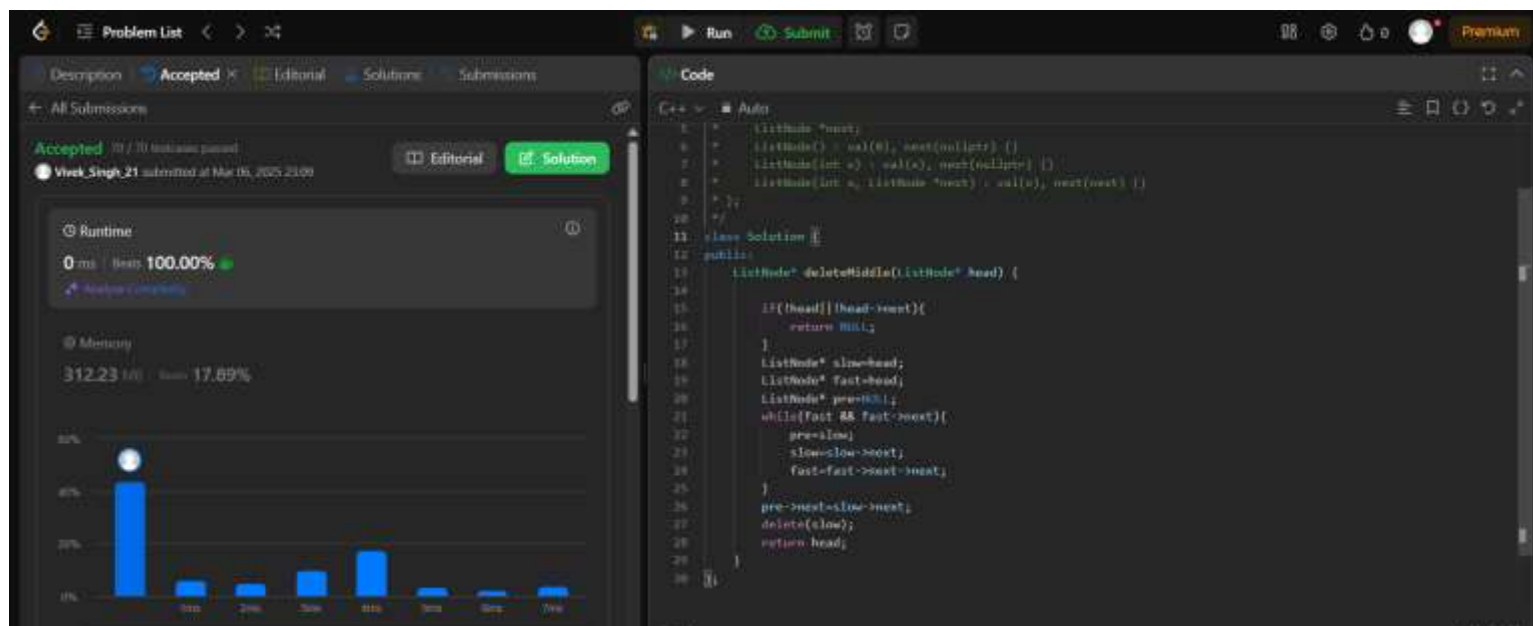
```
    pre->next=slow->next;
```

```
    delete(slow);
```

```
    return head;
```

```
}
```

```
};Submission:
```



Qs 5. Merge two sorted lists:

Code: class Solution {

public:

```
ListNode* mergeTwoLists(ListNode* list1, ListNode* list2) {
```

```
    ListNode *dummy, *temp;
```

```
    dummy = new ListNode();
```

```
    temp = dummy;
```

```
    while(list1 && list2){
```

```
        if(list1->val < list2->val){
```

```
            temp->next = list1;
```

```
            list1 = list1->next;
```

```
        }
```

```
    else{
```

```
        temp->next = list2;
```

```
        list2 = list2->next;
```

```
    }
```

```
    temp = temp->next;
```

```
}
```

```
if(list1) temp->next = list1;
```

```
if(list2) temp->next = list2;
```

```
return dummy->next;
```

```
}
```

};Submission:

The screenshot displays a code editor interface for a C++ solution. The left pane shows the problem description and submission status (Accepted). The right pane shows the code. The code defines a `ListNode` struct and a `mergeTwoLists` function. The function uses a dummy node and a temp pointer to build the merged list by comparing the values of the two input lists and appending the smaller one to the merged list. The submission is accepted with a runtime of 0 ms and a memory usage of 19.59 MB.

```
2 // Definition for singly-linked list.
3 struct ListNode {
4     int val;
5     ListNode *next;
6     ListNode() : val(0), next(nullptr) {}
7     ListNode(int x) : val(x), next(nullptr) {}
8     ListNode(int x, ListNode *next) : val(x), next(next) {}
9 };
10
11 class Solution {
12 public:
13     ListNode* mergeTwoLists(ListNode* list1, ListNode* list2) {
14         ListNode *dummy, *temp;
15         dummy = new ListNode();
16         temp = dummy;
17         while(list1 && list2){
18             if(list1->val < list2->val){
19                 temp->next = list1;
20                 list1 = list1->next;
21             }
22             else{
23                 temp->next = list2;
24                 list2 = list2->next;
25             }
26             temp = temp->next;
27         }
28         if(list1) temp->next = list1;
```

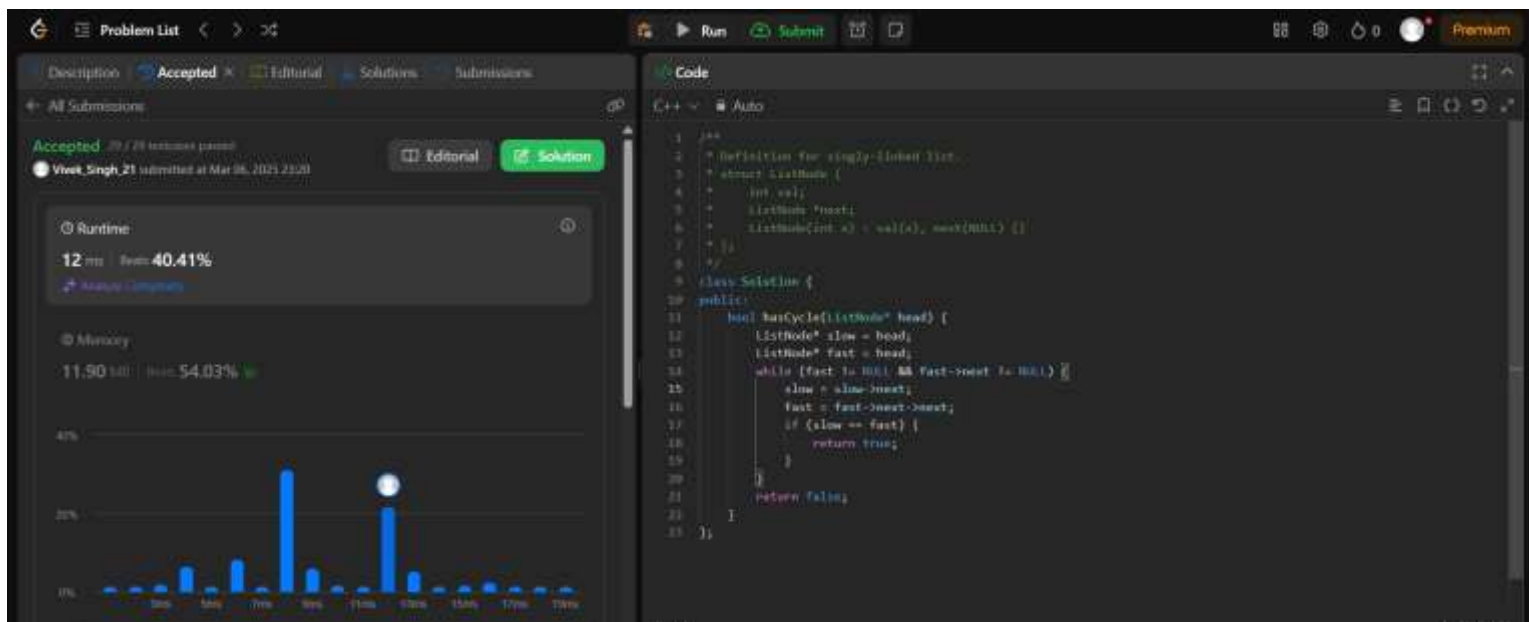
Qs 6. Detect cycle in a linked list:

Code: class Solution {

public:

```
bool hasCycle(ListNode* head) {  
    ListNode* slow = head;  
    ListNode* fast = head;  
    while (fast != NULL && fast->next != NULL) {  
        slow = slow->next;  
        fast = fast->next->next;  
        if (slow == fast) {  
            return true;  
        }  
    }  
    return false;  
}
```

Submission:



Qs 7. Rotate a list:

Code: class Solution {

public:

```
    ListNode* rotateRight(ListNode* head, int k) {  
        if (!head || !head->next || k == 0) return head;  
        ListNode* current = head;  
        int length = 1;  
        while (current->next)  
        {  
            length++;  
            current = current->next;  
        }  
        k %= length;  
        if (k == 0) return head;  
        current->next = head;  
        int newTailPos = length - k;  
        current = head;  
        for (int i = 1; i < newTailPos; i++)  
        {  
            current = current->next;  
        }  
        head = current->next;  
        current->next = nullptr;  
  
        return head;  
    }
```

};

Submission:

The screenshot displays a submission interface for a C++ problem. On the left, the submission status is 'Accepted' with a runtime of 0 ms (Beats 100.00%) and memory usage of 16.53 MB (Beats 4.06%). The right panel shows the C++ code for the `rotateRight` function, which rotates a linked list by `k` positions to the right.

```
22 public:
23     ListNode* rotateRight(ListNode* head, int k) {
24         if (!head || !head->next || k == 0) return head;
25         ListNode* current = head;
26         int length = 1;
27         while (current->next)
28         {
29             length++;
30             current = current->next;
31         }
32         k %= length;
33         if (k == 0) return head;
34         current->next = head;
35         int newTailPos = length - k;
36         current = head;
37         for (int i = 1; i < newTailPos; i++)
38         {
39             current = current->next;
40         }
41         head = current->next;
42         current->next = nullptr;
43         return head;
44     }
45 }
```

Qs 8. Sort List:

Code: class Solution {

public:

ListNode* getmid(ListNode* head) {

 ListNode* slow = head;

 ListNode* fast = head->next;

 while (fast != NULL && fast->next != NULL) {

 slow = slow->next;

 fast = fast->next->next;

 }

 return slow;

}

ListNode* merge(ListNode* left, ListNode* right) {

 if (left == NULL)

 return right;

 if (right == NULL)


```
return left;
```

```
ListNode* dummy = new ListNode(0);
```

```
ListNode* temp = dummy;
```

```
while (left != NULL && right != NULL) {
```

```
    if (left->val < right->val) {
```

```
        temp->next = left;
```

```
        temp = left;
```

```
        left = left->next;
```

```
    } else {
```

```
        temp->next = right;
```

```
        temp = right;
```

```
        right = right->next;
```

```
    }
```

```
}
```

```
while (left != NULL) {
```

```
    temp->next = left;
```

```
    temp = left;
```

```
    left = left->next;
```

```
}
```

```
while (right != NULL) {
```

```
    temp->next = right;
```

```
    temp = right;
```

```
    right = right->next;
```

```
}
```

```
dummy = dummy->next;
```

```
return dummy;
```

```
}
```

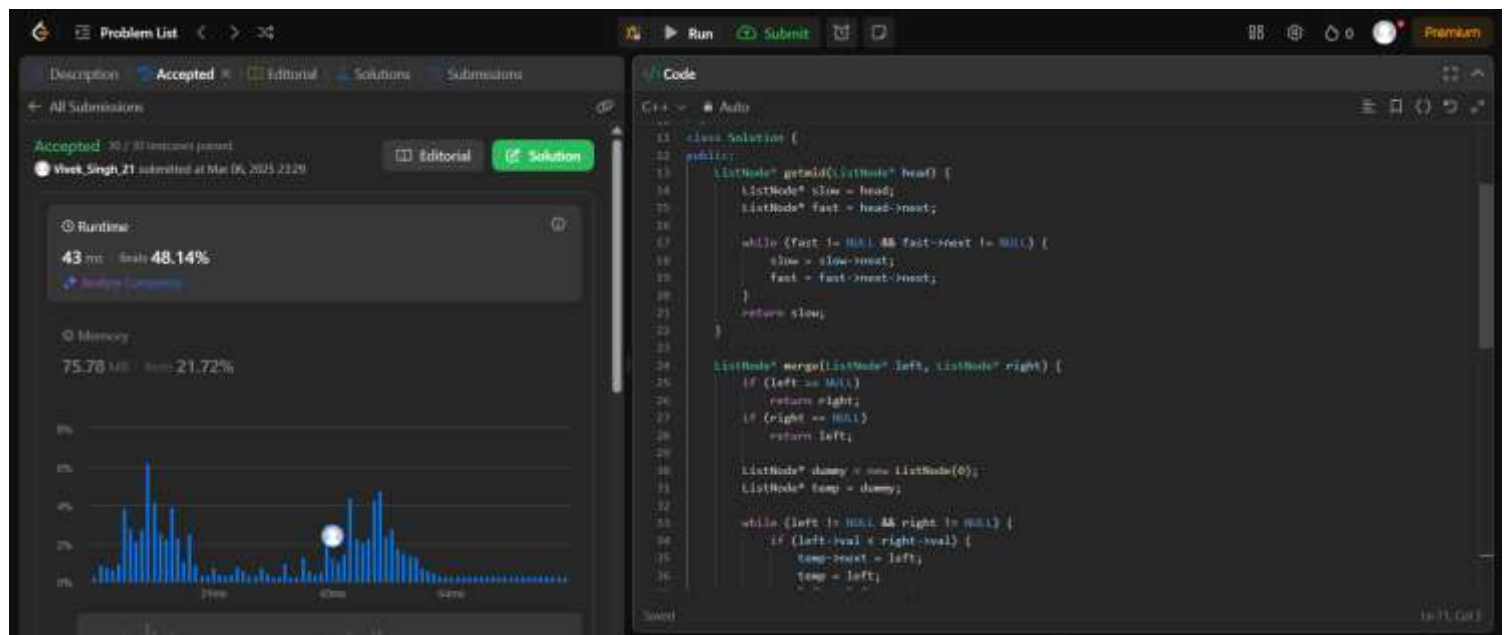
```

ListNode* sortList(ListNode* head) {
    if (head == NULL || head->next == NULL) return head;

    ListNode* mid = getmid(head);
    ListNode* left = head;
    ListNode* right = mid->next;
    mid->next = NULL;
    left = sortList(left);
    right = sortList(right);
    ListNode* result = merge(left, right);

    return result;
}
};Submission:

```



Qs 9. Merge K sorted list:

```

Code: class Solution {
public:
    ListNode* mergeKLists(vector<ListNode*>& lists) {
        priority_queue<pair<int,ListNode*>, vector<pair<int, ListNode*>>,
        greater<pair<int,ListNode*>>> pq;

```

```

for(int i=0;i<lists.size();i++){
    if(lists[i]) pq.push({lists[i]->val , lists[i]});
}

ListNode* dummy = new ListNode(-1);
ListNode* temp = dummy;

while(!pq.empty()){
    auto it = pq.top();
    if(it.second->next){
        pq.push({it.second->next->val,it.second->next});
    }
    pq.pop();
    temp->next = it.second;
    temp=temp->next;
}

return dummy->next;
}

};Submission:

```

The screenshot displays a code editor with a C++ solution for merging k sorted lists. The left sidebar shows submission details: 'Accepted' status, 134/134 test cases passed, and runtime/memory usage (4 ms, 18.79 MB). The main editor shows the C++ code implementing a priority queue approach to merge the lists.

```

1  /**
2   * Definition for singly-linked list.
3   * struct ListNode {
4   *     int val;
5   *     ListNode *next;
6   *     ListNode() : val(0), next(nullptr) {}
7   *     ListNode(int x) : val(x), next(nullptr) {}
8   *     ListNode(int x, ListNode *next) : val(x), next(next) {}
9   * };
10
11  class Solution {
12  public:
13      ListNode* mergeKLists(vector<ListNode*>& lists) {
14          priority_queue<pair<int, ListNode*>, vector<pair<int, ListNode*>>, greater<pair<int, ListNode*>>> pq;
15          for(int i=0;i<lists.size();i++){
16              if(lists[i]) pq.push({lists[i]->val , lists[i]});
17          }
18          ListNode* dummy = new ListNode(-1);
19          ListNode* temp = dummy;
20
21          while(!pq.empty()){
22              auto it = pq.top();
23              if(it.second->next){
24                  pq.push({it.second->next->val,it.second->next});
25              }
26              pq.pop();
27          }
28          return dummy->next;
29      }
30  };

```