Bi-Link Train

In the town of Linked Junction, a young conductor faces a magical challenge: two trains must be merged into one grand, mystical train. Each train's carriages have numbers and are already sorted & can connect to both their previous & next carriages, allowing passengers to travel safely in either direction. To maintain the magic, the conductor must merge the two trains while preserving the sorted order. If the sequence breaks, the magic fades, and the passengers lose their way.

Objective:

Implement MergeTrains and InsertEnd to merge two sorted doubly linked lists into a single sorted doubly linked list, maintaining the sorted order and double-linked nature.

Function Descriptions:

InsertEnd: Inserts a new node with a given value at the end of a doubly linked list, handling both empty and non-empty cases.
MergeTrains: Merges two sorted doubly linked lists into a single sorted doubly linked list.

Input Format

Line 1: The number of carriages in Train A.
Line 2: The carriage numbers for Train A, each on a new line.
Line 3: The number of carriages in Train B.
Line 4: The carriage numbers for Train B, each on a new line.

Constraints

Carriage numbers are greater than -5

Output Format

Merged carriage numbers in sorted order space separated.

Sample Input 0

```
3
2
4
6
4
6
7
8
9
```

Sample Output 0

```
2 4 6 6 7 8 9
```

Contest ends in 39 minutes

Submissions: 66
Max Score: 10

Difficulty: Medium

Rate This Challenge:

More

C

```c
#include <stdio.h>
#include <stdlib.h>

// Structure definition

typedef struct node {
    int val;
    struct node* next;
    struct node* prev;
} NODE;

NODE *getNode(int ele)
{
    NODE *temp = malloc(sizeof(NODE));
    temp->next = NULL;
    temp->val = ele;
    temp->prev = NULL;
    return temp;
}

NODE* InsertEnd(NODE* tail, int value) {
    NODE *temp = getNode(value);
    if (tail == NULL) {
        return temp;
    }
    tail->next = temp;
    temp->prev = tail;
    return temp;
}

void PrintList(NODE* head) {
    NODE* temp = head;
    while (temp != NULL) {
        printf("%d ", temp->val);
        temp = temp->next;
    }
    printf("\n");
}

NODE* MergeTrains(NODE* list1, NODE* list2) {
    NODE* merged = NULL;
    NODE* tail = NULL;

    while (list1 != NULL && list2 != NULL) {
        if (list1->val < list2->val) {
            tail = InsertEnd(tail, list1->val);
            if (merged == NULL) merged = tail;
            list1 = list1->next;
        } else {
            tail = InsertEnd(tail, list2->val);
            if (merged == NULL) merged = tail;
            list2 = list2->next;
        }
    }

    while (list1 != NULL) {
        tail = InsertEnd(tail, list1->val);
        if (merged == NULL) merged = tail;
        list1 = list1->next;
    }

    while (list2 != NULL) {
        tail = InsertEnd(tail, list2->val);
        if (merged == NULL) merged = tail;
```

```
        list2 = list2->next;
    }

    return merged;
}

NODE* FreeList(NODE* head) {
    NODE* temp;
    while (head != NULL) {
        temp = head;
        head = head->next;
        free(temp);
    }
    return head;
}

int main() {
    NODE* list1 = NULL;
    NODE* tail1 = NULL;
    NODE* list2 = NULL;
    NODE* tail2 = NULL;
    int n1, n2, value;

    scanf("%d", &n1);
    for (int i = 0; i < n1; i++) {
        scanf("%d", &value);
        if (list1 == NULL) {
            list1 = tail1 = InsertEnd(NULL, value);
        } else {
            tail1 = InsertEnd(tail1, value);
        }
    }

    scanf("%d", &n2);
    for (int i = 0; i < n2; i++) {
        scanf("%d", &value);
        if (list2 == NULL) {
            list2 = tail2 = InsertEnd(NULL, value);
        } else {
            tail2 = InsertEnd(tail2, value);
        }
    }

    NODE* mergedTrain = MergeTrains(list1, list2);

    PrintList(mergedTrain);

    mergedTrain = FreeList(mergedTrain);

    return 0;
}
```

Line: 115 Col: 2

Upload Code as File Test against custom input Run Code Submit Code

Testcase 0 ✔

Congratulations, you passed the sample test case.

Click the **Submit Code** button to run your code against all the test cases.

Input (stdin)

```
3
2
4
6
4
6
7
```

```
8
9
```

Your Output (stdout)

```
2 4 6 6 7 8 9
```

Expected Output

```
2 4 6 6 7 8 9
```