Sunday, January 16, 2022 8:14 PM

Sort A “K” Sorted Doubly Linked List

Posted: 8 Feb, 2021

Difficulty: Moderate

PROBLEM STATEMENT

Try Problem

You’re given a doubly-linked list with N nodes, where each node deviates at max K position from its position in the sorted list. Your task is to sort this given doubly linked list.

For Example :
Let us consider K is 3, an element at position 4 in the sorted doubly linked list, can be at positions 1, 2, 3, 4, 5, 6, 7 in the given linked list because the absolute difference of all these indices with 4 is at most 3.

Note :
All elements are distinct.

A doubly linked list is a type of linked list that is bidirectional, that is, it can be traversed in both directions, forward and backward.

Input Format :
The first line of input contains T, the number of test cases.

The first line of each test case contains an integer K, as specified in the problem statement.

The second line contains the elements of the doubly linked list separated by a single space and terminated by -1. Hence, -1 would never be a list element.

Output Format :
For each test case print in a new line the sorted linked list, the elements of the sorted list should be single-space separated, terminated by -1.

Note :
You don’t need to print anything. It has already been taken care of. Just implement the given function.

Constraints :
1 <= T <= 10
1 <= N <= 10000
1 <= K < N

Time Limit: 1 sec

APPROACH 1

We will use Insertion sort to sort the doubly linked list efficiently.

Iterate the given doubly linked list head till the last node n, where n is the size of the doubly linked list.

- Compare the current element (key) to its predecessor.
- If the key element is smaller than its predecessor, compare elements before. Move the greater elements one position up to make space for the swapped element.

Let’s understand this in a detailed way.

- Start by making a Node *current = head and then run a loop till current is not NULL.
- Make a Node *back = current -> prev and an integer variable named key = current->data.
- Now run a loop till back is not NULL and the key is less than current->data.
- Inside this loop update back -> next -> data = back -> data and back= back -> prev.
- After exiting this loop check if the back is NULL or not. If it is NULL, then update head -> data = key;

- Else update back -> next -> data = key, and after this update current = current -> next.
- At last return head, which is our answer.

From <https://www.codingninjas.com/codestudio/problem-details/sort-a-k-sorted-doubly-linked-list_1118118>

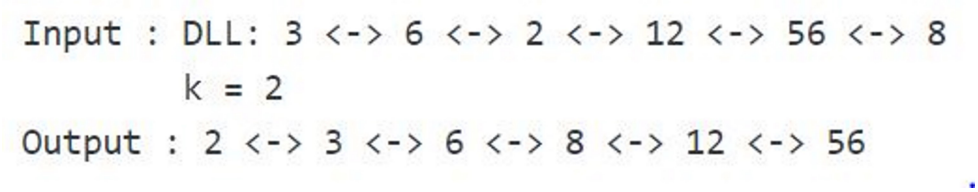
Sort a k sorted doubly linked list

- Difficulty Level : Hard
- Last Updated : 24 Nov, 2021

Given a doubly linked list containing **n** nodes, where each node is at most **k** away from its target position in the list. The problem is to sort the given doubly linked list.

For example, let us consider **k** is 2, a node at position 7 in the sorted doubly linked list, can be at positions 5, 6, 7, 8, 9 in the given doubly linked list.

Examples:



From <https://www.geeksforgeeks.org/sort-k-sorted-doubly-linked-list/>

```
struct Node* sort(struct Node* head, int k)
{
    if (head == NULL)
        return head;

    priority_queue<Node*, vector<Node*>, compare> pq;

    struct Node* newHead = NULL, *cur;

    for (int i = 0; head != NULL && i <= k; i++) {
        pq.push(head);
        head = head->next;
    }

    while (!pq.empty()) {

        if (newHead == NULL) {
            newHead = pq.top();
            newHead->prev = NULL;
            cur = newHead;
        }
        else {
            cur->next = pq.top();
            pq.top()->prev = cur;
            cur = pq.top();
        }

        pq.pop();

        if (head != NULL) {
            pq.push(head);
            head = head->next;
        }
    }
    cur->next = NULL;

    return newHead;
```