Sort a k sorted doubly linked list

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.

```
import heapq


class Node:
    def __init__(self, val):
        self.val = val
        self.next = None
        self.prev = None


def sortKsortedDLL(head, k):
    if head is None or head.next is None:
        return head
    heap = []
    dummyHead = Node(-1)
    last = dummyHead
    curr = head
    while k >= 0 and curr != None:
        heapq.heappush(heap, (curr.val, curr))
        curr = curr.next
        k = k - 1
    while len(heap) > 0:
        value, node = heapq.heappop(heap)
        last.next = node
        node.prev = last
        last = node
        if curr != None:
            heapq.heappush(heap, (curr.val, curr))
            curr = curr.next
    last.next = None
    return dummyHead.next


head = Node(3)
```

```
head.next = Node(6)
head.next.next = Node(2)
head.next.next.next = Node(12)
head.next.next.next.next = Node(56)
head.next.next.next.next.next = Node(8)
# head.next.next.next.next.next.next = Node(5)
# head.next.next.next.next.next.next.next = Node(1)

head = sortKsortedDLL(head, 2)
while head != None:
    print(head.val, end=' ')
    head = head.next
```