Sort numbers stored on different machines

Given N machines. Each machine contains some numbers in sorted form. But the amount of numbers, each machine has is not fixed. Output the numbers from all the machine in sorted non-decreasing form.

```
Example:
       Machine M1 contains 3 numbers: {30, 40, 50}
       Machine M2 contains 2 numbers: {35, 45}
       Machine M3 contains 5 numbers: {10, 60, 70, 80, 100}

       Output: {10, 30, 35, 40, 45, 50, 60, 70, 80, 100}
```

Representation of stream of numbers on each machine is considered as linked list. A Min Heap can be used to print all numbers in sorted order.

Following is the detailed process

1. Store the head pointers of the linked lists in a minHeap of size N where N is number of machines.

2. Extract the minimum item from the minHeap. Update the minHeap by replacing the head of the minHeap with the next number from the linked list or by replacing the head of the minHeap with the last number in the minHeap followed by decreasing the size of heap by 1.

3. Repeat the above step 2 until heap is not empty.

Below is C++ implementation of the above approach.

```
// A program to take numbers from different machines and
#include <stdio.h>

// A Linked List node
struct ListNode
{
    int data;
```

```
    struct ListNode* next;
};

// A Min Heap Node
struct MinHeapNode
{
    ListNode* head;
};

// A Min Heao (Collection of Min Heap nodes)
struct MinHeap
{
    int count;
    int capacity;
    MinHeapNode* array;
};

// A function to create a Min Heap of given capacity
MinHeap* createMinHeap( int capacity )
{
    MinHeap* minHeap = new MinHeap;
    minHeap->capacity = capacity;
    minHeap->count = 0;
    minHeap->array = new MinHeapNode [minHeap->capacity]
    return minHeap;
}

/* A utility function to insert a new node at the begini
   of linked list */
void push (ListNode** head_ref, int new_data)
{
    /* allocate node */
    ListNode* new_node = new ListNode;

    /* put in the data  */
    new_node->data  = new_data;

    /* link the old list off the new node */
    new_node->next = (*head_ref);

    /* move the head to point to the new node */
    (*head_ref)    = new_node;
}

// A utility function to swap two min heap nodes. This f
// is needed in minHeapify
```

```
void swap( MinHeapNode* a, MinHeapNode* b )
{
    MinHeapNode temp = *a;
    *a = *b;
    *b = temp;
}

// The standard minHeapify function.
void minHeapify( MinHeap* minHeap, int idx )
{
    int left, right, smallest;
    left = 2 * idx + 1;
    right = 2 * idx + 2;
    smallest = idx;

    if ( left < minHeap->count &&
         minHeap->array[left].head->data <
         minHeap->array[smallest].head->data
       )
        smallest = left;

    if ( right < minHeap->count &&
         minHeap->array[right].head->data <
         minHeap->array[smallest].head->data
       )
        smallest = right;

    if( smallest != idx )
    {
        swap( &minHeap->array[smallest], &minHeap->array
        minHeapify( minHeap, smallest );
    }
}

// A utility function to check whether a Min Heap is emp
int isEmpty( MinHeap* minHeap )
{
    return (minHeap->count == 0);
}

// A standard function to build a heap
void buildMinHeap( MinHeap* minHeap )
{
    int i, n;
    n = minHeap->count  - 1;
    for( i = (n - 1) / 2; i >= 0; --i )
```

```
        minHeapify( minHeap, i );
}

// This function inserts array elements to heap and then
// buildHeap for heap property among nodes
void populateMinHeap( MinHeap* minHeap, ListNode* *array
{
    for( int i = 0; i < n; ++i )
        minHeap->array[ minHeap->count++ ].head = array[

    buildMinHeap( minHeap );
}

// Return minimum element from all linked lists
ListNode* extractMin( MinHeap* minHeap )
{
    if( isEmpty( minHeap ) )
         return NULL;

    // The root of heap will have minimum value
    MinHeapNode temp = minHeap->array[0];

    // Replace root either with next node of the same li
    if( temp.head->next )
        minHeap->array[0].head = temp.head->next;
    else // If list empty, then reduce heap size
    {
        minHeap->array[0] = minHeap->array[ minHeap->cou
        --minHeap->count;
    }

    minHeapify( minHeap, 0 );
    return temp.head;
}

// The main function that takes an array of lists from N
// and generates the sorted output
void externalSort( ListNode *array[], int N )
{
    // Create a min heap of size equal to number of mach
    MinHeap* minHeap = createMinHeap( N );

    // populate first item from all machines
    populateMinHeap( minHeap, array, N );

    while ( !isEmpty( minHeap ) )
```

```
    {
        ListNode* temp = extractMin( minHeap );
        printf( "%d ",temp->data );
    }
}

// Driver program to test above functions
int main()
{
    int N = 3; // Number of machines

    // an array of pointers storing the head nodes of th
    ListNode *array[N];

    // Create a Linked List 30->40->50 for first machine
    array[0] = NULL;
    push (&array[0], 50);
    push (&array[0], 40);
    push (&array[0], 30);

    // Create a Linked List 35->45 for second machine
    array[1] = NULL;
    push (&array[1], 45);
    push (&array[1], 35);

    // Create Linked List 10->60->70->80 for third machi
    array[2] = NULL;
    push (&array[2], 100);
    push (&array[2], 80);
    push (&array[2], 70);
    push (&array[2], 60);
    push (&array[2], 10);

    // Sort all elements
    externalSort( array, N );

    return 0;
}
```

Output:

```
10 30 35 40 45 50 60 70 80 100
```