International Journal of Electrical Electronics & Computer Science Engineering
Special Issue - NCSCT-2018 | E-ISSN : 2348-2273 | P-ISSN : 2454-1222
March, 2018 | Available Online at www.ijeecse.com

Recursive Approach to Implement Merge Sort on Linked List

Rahul Gupta[1], Pranav Pandey[2], Puneet Mathur[3]
[1-3]UG Scholar, Department of CSE, Poornima Institute of Engineering and Technology, Jaipur, Rajasthan, India

Abstract: Sorting is one of the most fascinating problems in the area of computer science. Merge sort is one of the most efficient sorting algorithms based on divide and conquer approach. A recursive approach to implement the merge sort algorithm on the linked list is proposed in this paper.In most implementations it is stable, i.e. it preserves the input order of identical elements in the sorted output. Merge sort works in two stages: At first stage list is divided into two halves and the two halves are sorted separately and in the second stage these two sorted sub lists are merged into a single sorted list. The paper also discusses the comparison between implementation of merge sort on arrays and dynamic lists as well.

Keywords: Divide and Conquer, Merge Sort, Linked List, Traversing, Swapping, Recursion.

1. INTRODUCTION

Linked list is a linear and dynamic data structure and it consists of a group of nodes each pointing to the next node through pointer.

Merge sort is a divide and conquer sorting algorithm, used for placing the elements in either ascending or descending order. In this algorithm, we divide the list into two halves, about the middle of list. So, we first find out the middle of list and divide the list about that mid point(inclusive in the first half). Then, the two halves are sorted separately, and then merged into a single fully sorted list. For a given list or array, this dividing and merging process is recursive, and repeated until one element is reached, which is already sorted in itself and then merging takes place.

So, the merge sort is a combination of two algorithms, namely; Sort and Merge. "Sort" cut the list about middle (recursively) and "merge" append or join the two sorted sub-lists such that the final list is sorted.

II. SOURCE CODES FOR IMPLEMENTING MERGE SORT ON LINKED LIST

A. Sorting:

```
node* sort( node *head,  node *start, node *end)
{
node *slow,*fast,*temp2,*temp1,*temp3;
   temp3=end->next;
if(start==end)
return;
for(slow=fast=start;fast!=end;)
   {
fast=fast->next;
if(fast!=end)
      {
fast=fast->next;
slow=slow->next;
      }
   }
   temp2=slow->next;
start=sort(start,slow);
for(temp1=start;temp1->next!=temp2;temp1=temp1->next);
   temp1->next=sort(temp2,end);
for(end=start;end->next!=temp3;end=end->next);
return merge(start,temp1,end);
}
```

B. Merging:

```
node* merge ( node *head,  node *start, node *slow, node  *end )
{
node *cur1, *cur2, *start2, *pre1, *pre2 ;
if ( start == end )
return ;
if ( start == head )
      pre1 = 0 ;
else
for ( pre1 = head ; pre1->next != start ; pre1 = pre1->next ) ;
if ( start == slow )
   {
if ( start->info > end->info )
      {
head = rem_node ( head, start ) ;
head = insert ( head, start, end ) ;
start = end ;
      }
return start ;
   }
else if ( slow->next == end )
   {
      If ( end->info < start->info )
      {
head = rem_node ( head, end ) ;
head = insert ( head, end, pre1 ) ;
start = end ;
      }
```

```
else if ( end->info < slow->info )
      {
head = rem_node ( head, end ) ;
head = insert ( head, end, start ) ;
      }
return start ;
   }
else
   {
for ( cur1 = start, cur2 = start2 = slow->next ; cur1 != start2 ; pre1 = cur1 , cur1 = cur1->next, cur2 = start2 )
      {
         If ( cur1->info > cur2->info )
         {
for ( pre2 = slow ; cur2 != end->next && cur1->info > cur2->info ; pre2 = cur2 , cur2 = cur2->next ) ;
if ( cur1 == slow )
            {
If(cur2==start2)
               {
head = rem_node( head, cur1);
head = insert( head, cur1,start2);
               }
else
               {
head = rem_node( head,cur1);
head = insert(head, cur1,pre2);
               }
return start;
            }
else if(cur2==start2)
            {
head = rem_node( head,cur1);
head = rem_node(head, start2);
head = insert(head, start2,pre1);
head = insert( head, cur1,slow->next);
if(cur1==start)
start=start2;
            }
else
            {
head = rem_node(head, cur1);
head = rem_node(head, start2);
head = insert(head, start2,pre1);
head = insert(head, cur1,pre2);
            }
If(start==cur1)
start=start2;
if(end==pre2)
end=cur1;
               cur1=start2;
               start2=slow->next;
         }
      }
   }
return start;
}
```

III. SUPPORTING CODES

1. To remove a node:

```
node*rem_node(node *head, node *ptr)
{
node *cur;
if(ptr==head)
head=ptr->next;
else
   {
for(cur=head;cur->next!=ptr ; cur=cur->next);
cur->next=ptr->next;
   }
ptr->next =0;
return head;
}
```

2. To insert a node:

```
node* insert ( node *head, node *ptr, node *pre )
{
if ( !pre )
   {
ptr->next = head ;
head = ptr ;
   }
else
   {
ptr->next = pre->next ;
pre->next = ptr ;
   }
return head;
}
```

IV. CONCLUSION

1. Application of merge sort on linked list can be a burden as compared to arrays. The reasons are;
   a. The cost of traversing a linked list is certainly higher than indexing an element in an array.

b. The arrays have contiguous memory allocation, which makes it easy to access the elements through indexing, whereas in linked list the nodes may be present at random locations in memory, which makes it laborious to access the nodes.

c. Linked lists are dynamic in nature, so nodes can be added or removed at run time.

2. The insertion and deletion (which is needed in merge sort) can be done in constant time in case of linked list, as we just have to modify the links, whereas in arrays we have to shift all succeeding elements.

3. Complexity.

V. REFERENCES

[1] Sedgewick, Algorithms in C++, pp.98-100, ISBN 0-201-51059- 6, Addison-Wesley, 1992.

[2] Sedgewick, Algorithms in C++, pp.100-104, ISBN 0-201- 51059-6, Addison-Wesley, 1992.

[3] Owen Astrachan, Bubble Sort: An Archaeological Algorithmic Analysis, SIGCSE 2003, http://www.cs.duke.edu/~ola/papers/bubble.pdf.