GeeksMan
Linked List
Lesson 6

Merge Sort for Linked List

Given Pointer/Reference to the head of the linked list, the task is to **Sort the given linked list using Merge Sort**.

Note: If the length of the linked list is odd, then the extra node should go in the first list while splitting.

Example 1:

Input:
N = 5
value[] = {3,5,2,4,1}
Output: 1 2 3 4 5
Constraints:

1 <= T <= 100

1 <= N <= 10^5

```
1.  Node* merge(Node *f,Node *s)
2.   {
3.      if(f==NULL)
4.      return s;
5.      else if(s==NULL)
6.      return f;
7.      else
8.      {
9.         if(f->data<s->data)
10.        {
11.           f->next=merge(f->next,s);
12.           return f;
13.        }
14.        else
```

```
        {
            s->next=merge(f,s->next);
            return s;
        }
    }
}
Node* mergeSort(Node* head) {
  // your code here
  if(head==NULL || head->next==NULL)
  return head;
  Node *fast,*slow,*divid;
  slow=head;
  fast=head->next;
  while(fast && fast->next)
  {
      slow=slow->next;
      fast=fast->next->next;
  }
  divid=slow->next;
  slow->next=NULL;
  return merge(mergeSort(head),mergeSort(divid));
}
```

sort-a-linked-list=

https://sapphireengine.com/@/qlwiie

merge-sort-on-doubly-linked-list

Absolute List Sorting

Given a linked list **L** of **N** nodes, sorted in ascending order based on the absolute values of its data. Sort the linked list according to the actual values.

Ex: **Input** : 1 -> -2 -> -3 -> 4 -> -5

Output: -5 -> -3 -> -2 -> 1 -> 4

Input

The first line of input contains an integer **T** denoting the number of test cases. Then **T** test cases follow. Each test case consists of two lines. The first line of each test case contains a positive integer **N** denoting the size of the linked list. The second line of each test case contains **N** space separated integers denoting the values of **N** nodes.

Output

Corresponding to each test case, the expected output will be space separated values of the sorted linked list.

Constraints

1 <= **T** <= 100

0 < **N** <= 30

-100 <= **L[i]** <= 100

Examples

Input

2

3

1 -3 -4

4

0 -2 3 -10

Output

-4 -3 1

-10 -2 0 3

```
1. void sortList(Node** head)
2. {
3.    Node* prev = (*head);
4.    Node* curr = (*head)->next;
5.    while (curr != NULL)
6.    {
7.       if (curr->data < prev->data)
8.       {
9.          prev->next = curr->next;
10.
11.         curr->next = (*head);
12.         (*head) = curr;
13.         curr = prev;
14.      }
15.         else
16.         prev = curr;
17.         curr = curr->next;
18.   }
19.}
```

absolute-list-sorting=

https://sapphireengine.com/@/uo76rv

https://sapphireengine.com/@/emsi1b (CODE)

Merge 2 sorted linked list in reverse order

Given two linked lists of size **N** and **M**, which are sorted in non-decreasing order. The task is to merge them in such a way that the resulting list is in decreasing order.

Input:

First line of input contains a number of test cases T. For each test case, the first line of input contains the length of both linked lists N and M respectively. Next two lines contains N and M elements of two linked lists.

Output:

For each test case, print the merged linked list which is in non-increasing order.

User Task:

The task is to complete the function **mergeResult**() which takes reference to the heads of both linked lists and returns the pointer to the merged linked list.

Constraints:

1 <=T<= 50

1 <= N, M <= 1000

Example:

Input:

2

4 3

5 10 15 40

2 3 20

2 2

1 1

2 4

Output:

40 20 15 10 5 3 2

4 2 1 1

Explanation:

Testcase 1: After merging the two lists in decreasing order, we have new lists as 40->20->15->10->5->3->2.

merge-2-sorted-linked-list-in-reverse-order=
1)merge(recursively) and reverse https://sapphireengine.com/@/qcmtay
2)iteratively take a dummy node and start adding valueshttps://sapphireengine.com/@/ur071x

Rearrange linked list in-place

Given a singly linked list L0 -> L1 -> … -> Ln-1 -> Ln. Rearrange the nodes in the list so that the new formed list is: L0 -> Ln -> L1 -> Ln-1 -> L2 -> Ln-2.

Input:

You have to complete the method which takes 1 argument: the head of the linked list. You should not read any input from stdin/console. There are multiple test cases. For each test case, this method will be called individually.

Output:

Your function should return a pointer to the rearranged list so obtained.

User Task:

The task is to complete the function **inPlace**() which should rearrange the given linked list as required.

Constraints:

1 <=T<= 50

1 <= size of linked lists <= 100

Example:

Input:

2

4

1 2 3 4

5

1 2 3 4 5

Output:

1 4 2 3

1 5 2 4 3

Explanation:

Testcase 1: After rearranging the linked list as required, we have 1, 4, 2 and 3 as the elements of the linked list.

Rearrange-linked-list-in-place

First method is simply take the head pointer and traverse the LL until you reach at last node , delete that node from last and insert it after the head . now move the head to the next position and repeat it until the head reaches at the last position. **O(N*N)**

Second method is simply create a vector , do the required swapping and make the links accordingly.
https://sapphireengine.com/@/r1dd2l

Third method is using recursion doubt

Hold a pointer to the head node and go till the last node using recursion
Once last node is reached, start swapping the last node to the next of head node
Move the head pointer to the next node
Repeat this until the head and last node meet or come adjacent to each other

Fourth method is using tortoise and hare method.https://sapphireengine.com/@/8vynr0

Merge two sorted linked lists

Merge-two-sorted-linked-lists

Given two sorted linked lists consisting of **N and M** nodes respectively. The task is to merge both of the list (in-place) and return head of the merged list.

Example 1:

Input:
N = 4, M = 3
valueN[] = {5,10,15,40}
valueM[] = {2,3,20}
Output: 2 3 5 10 15 20 40
Example 2:

Input:
N = 2, M = 2
valueN[] = {1,1}
valueM[] = {2,4}
Output:1 1 2 4
Your Task:

The task is to complete the function **sortedMerge**() which takes references to the heads of two linked lists as the arguments and returns the head of the merged linked list.

Expected Time Complexity : *O*(n+m)

Expected Auxiliary Space : *O*(1)

Constraints:

1 <= N, M <= 104

1 <= Node's data <= 105

Recursively:

```
1.    Node* sortedMerge(Node* node_A, Node* node_B)
2.    {
3.      if(node_A==NULL)
4.      return node_B;
5.      else if(node_B==NULL)
6.      return node_A;
7.      else
8.      {
9.
10.     if(node_A->data<node_B->data)
11.     {
12.        node_A->next=sortedMerge(node_A->next,node_B);
13.        return node_A;
14.     }
15.     else
16.     {
17.         node_B->next=sortedMerge(node_A,node_B->next);
18.         return node_B;
19.     }
20.     }
21.     return node_A;
22.     // code here
23.   }
```

Iteratively:

```
1.      struct Node* mergeUtil(struct Node* h1,
2.                             struct Node* h2)
3.      {
4.          if (!h1->next)
5.      {
6.           h1->next = h2;
7.           return h1;
8.        }
9.        struct Node *curr1 = h1, *next1 = h1->next;
10.       struct Node *curr2 = h2, *next2 = h2->next;
11.
12.       while (next1 && curr2)
13.     {
14.          if ((curr2->data) >= (curr1->data) && (curr2->data) <= (next1->data)) {
15.             next2 = curr2->next;
16.             curr1->next = curr2;
17.             curr2->next = next1;
18.
19.             curr1 = curr2;
20.             curr2 = next2;
21.          }
22.          else {
23.                if (next1->next)
24.            {
25.                next1 = next1->next;
26.                curr1 = curr1->next;
27.             }
28.
29.                else {
30.                next1->next = curr2;
```

```
31.                 return h1;
32.             }
33.         }
34.     }
35.     return h1;
36. }
37.
38. struct Node* merge(struct Node* h1,struct Node* h2)
39. {
40.     if (!h1)
41.         return h2;
42.     if (!h2)
43.         return h1;
44.
45.       if (h1->data < h2->data)
46.         return mergeUtil(h1, h2);
47.     else
48.         return mergeUtil(h2, h1);
49. }
```

https://sapphireengine.com/@/ermykq (CODE)

Linked List that is Sorted Alternatingly

Linked-list-that-is-sorted-alternatingly
https://sapphireengine.com/@/45wqn0

Given a Linked list of size **N**, the list is in alternating ascending and descending orders. Sort the given linked list in non-decreasing order.

Example 1:

Input:
LinkedList: 1->9->2->8->3->7
Output: 1 2 3 7 8 9
Explanation: After sorting the given list will be 1-> 2-> 3-> 7-> 8-> 9.

Example 2:

Input:
LinkedList: 13->99->21->80->50
Output: 13 21 50 80 99
Explanation: After sorting the given list will be 12-> 21-> 50-> 80-> 99.

Your Task:

You do not need to read input or print anything. The task is to complete the function **sort**() which should sort the linked list in non-decreasing order.

Expected Time Complexity: *O*(N)

Expected Auxiliary Space: *O*(1)

Constraints:

1 <= Number of nodes <= 100

```
1.  Node*merge(Node *first , Node *second)
2.  {
3.    //if(first == NULL && second == NULL)
4.       // return NULL;
5.    if(first == NULL)
6.       return second;
7.    else if(second == NULL)
8.       return first;
9.    else
10.   {
11.      if(first->data < second->data)
12.      {
13.         first->next = merge(first->next , second);
14.         return first;
15.      }
16.      else
17.      {
```

```
18.          second->next = merge(first , second->next);
19.          return second;
20.      }
21.
22.  }
23.
24.}
25.Node *reverse(Node *root)
26.{
27.  if(root == NULL || root->next == NULL)
28.      return root;
29.  Node *prevp = NULL;
30.  Node *curr = root;
31.  Node *nextp;
32.
33.  while(curr)
34.  {
35.      nextp = curr->next;
36.      curr->next = prevp;
37.      prevp = curr;
38.      curr = nextp;
39.  }
40.  return prevp;
41.}
42.void sort(Node **head)
43.{
44. if(*head==NULL || (*head)->next==NULL)
45.  return ;
46.  Node *curr = *head;
47.  Node *i = curr , *i_temp = curr;
```

```
48.  Node *d = curr->next  , *d_temp = curr->next;
49.  curr = curr->next;
50.  int ip = 0;
51.  while(curr && curr->next)
52.  {
53.     if(ip%2 == 0)
54.        {
55.           i_temp->next = curr->next;
56.           i_temp = i_temp->next;
57.        }
58.     else
59.        {
60.           d_temp->next = curr->next;
61.           d_temp = d_temp->next;
62.        }
63.        ip++;
64.        curr = curr->next;
65.  }
66.
67.  i_temp->next = NULL;
68.  d_temp->next = NULL;
69.  Node *h = reverse(d);
70.  *head = merge(i , h);
71.   // Code here
72.}
```

Flattening a Linked List

Given a Linked List of size N, where every node represents a sub-linked-list and contains two pointers:

(i) a **next** pointer to the next node,

(ii) a **bottom** pointer to a linked list where this node is head.

Each of the sub-linked-lists is in sorted order.

Flatten the Link List such that all the nodes appear in a single level while maintaining the sorted order.

Note: The flattened list will be printed using the bottom pointer instead of next pointer.

Example :

Input:

```
5 -> 10 -> 19 -> 28
|     |       |       |
7    20     22    35
|              |       |
8           50    40
|              |
30         45
```

Output: 5-> 7-> 8- > 10 -> 19-> 20->22-> 28-> 30-> 35-> 40-> 45-> 50.

Expected Time Complexity: *O*(N*M)

Expected Auxiliary Space: *O*(1)

Flattening-a-linked-list = https://sapphireengine.com/@/n7e8ri

QuickSort on Doubly Linked List

Sort the given doubly linked list of size **N** using quicksort. Just complete the partition function using the quicksort technique.

Example 1:

Input:
LinkedList: 4->2->9
Output:
2 4 9

Your Task:

Your task is to complete the given function **partition**(), which accepts the first and last node of the given linked list as input parameters and returns the pivot's address.

Expected Time Complexity: *O*(NlogN)

Expected Auxilliary Space: *O*(1)

Constraints:

1 <= N <= 200

```
1.  int partition (int arr[], int l, int r)
2.  {
3.          int x = arr[r];
4.          int i = (l - 1);
5.
6.
7.          for (int j = l; j <= r- 1; j++)
8.          {
9.                  if (arr[j] <= x)
10.                 {
11.                         i++;
12.                         swap (&arr[i], &arr[j]);
13.                 }
14.         }
15.         swap (&arr[i + 1], &arr[r]);
16.         return (i + 1);
17. }
18.
19.
20.void quickSort(int A[], int l, int r)
21.{
22.         if (l < r)
23.         {
24.                 int p = partition(A, l, r);
25.                 quickSort(A, l, p - 1);
26.                 quickSort(A, p + 1, r);
27.         }
28.}
29.
```

```
1.  void _quickSort(struct Node* l, struct Node *h)
2.  {
3.     if (h != NULL && l != h && l != h->next)
4.     {
5.        struct Node *p = partition(l, h);
6.        _quickSort(l, p->prev);
7.        _quickSort(p->next, h);
8.     }
9.  }
10.
11. void quickSort(struct Node *head)
12. {
13.    // Find last Node
14.    struct Node *h = lastNode(head);
15.
16.    // Call the recursive QuickSort
17.    _quickSort(head, h);
18. }
19. Node* partition(Node *head, Node *l){
20.    //Your code goes here
21.    Node *curr=head->prev;
22.
23.
24.     for(Node *h=head;h!=l;h=h->next)
25.  {
26.      if(h->data<=l->data)
27.      {
28.          if(curr==NULL)
29.          {
30.              curr=head;
```

```
        }
        else
        curr=curr->next;
        swap(&(curr->data),&(h->data));
    }
}
if(curr==NULL)
{
    curr=head;
}
else
    curr=curr->next;

swap(&(curr->data),&(l->data));
return curr;
}
```

quicksort-on-doubly-linked-list=
https://sapphireengine.com/@/4gc0d2