Delete Middle of Linked List

Easy Accuracy: 54.52% Submissions: 69K+ Points: 2

Given a singly linked list, delete **middle** of the linked list. For example, if given linked list is 1->2->**3**->4->5 then linked list should be modified to 1->2->4->5.

If there are **even** nodes, then there would be **two middle** nodes, we need to delete the second middle element. For example, if given linked list is 1->2->3->4->5->6 then it should be modified to 1->2->3->5->6.

If the input linked list is NULL or has 1 node, then it should return NULL

Example 1:

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

Example 2:


```
// } Driver Code Ends


// Deletes middle of linked list and returns head of the modified list
Node* deleteMid(Node* head)
{
    // Your Code Here
    Node *curr = head;
    Node *slow = head , *fast = head;
    if(curr ->next == NULL ){
        return NULL;
    }
    fast = fast->next->next;
    while(fast != NULL && fast->next != NULL){
        slow = slow ->next;
        fast = fast->next->next;
    }
    curr = slow ->next->next;
    slow ->next = curr;
    return head;
}
```

Problem | Editorial | Submissions | Comments

Linked List that is Sorted Alternatingly

Easy | Accuracy: 33.67% | Submissions: 17K+ | Points: 2

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:

C++ (g++ 5.4) | Average Time: 15m | Start Timer

```
// } Driver Code Ends


// your task is to complete this function
void sort(Node **H)
{
    // Code here
    Node *head=*H;
    while(true){
        int flag=0;
        head=*H;
        while((head)->next!=NULL){
            if((head)->data > (head)->next->data){
                swap((head)->data,(head)->next->data);
                flag=1;
            }
            (head)=(head)->next;
        }
        if(flag==0)
            break;

    }
}
```

Custom Input | Compile & Run | Submit

Intersection of Two Linked Lists

Easy Accuracy: 35.3% Submissions: 47K+ Points: 2

Given two linked lists, the task is to complete the function **findIntersection()**, that returns the intersection of two linked lists. Each of the two linked list contains distinct node values.

Example 1:

Input:
LinkedList1: 9->6->4->2->3->8
LinkedList2: 1->2->8->6
Output: 6 2 8

Your Task:
You task is to complete the function **findIntersection()** which takes the heads of the 2 input linked lists as parameter and returns the head of intersection list. Returned list will be automatically printed by driver code

C++ (g++ 5.4) Average Time: 15m Start Timer

```
// } Driver Code Ends

class Solution{
  public:
    Node* findIntersection(Node* head1, Node* head2)
    {
        unordered_map<int,bool>mp;
        while(head2){
            mp[head2->data]=true;
            head2=head2->next;
        }
        Node *res=new Node(-1);
        Node *temp=res;
        while(head1){
            if(mp[head1->data]){
                temp->next=new Node(head1->data);
                temp=temp->next;
            }
            head1=head1->next;
        }
        return res->next;
    }
};
// } Driver Code Ends
```

Custom Input | Compile & Run | Submit

Nth node from end of linked list

Easy Accuracy: 41.18% Submissions: 280K+ Points: 2

Given a linked list consisting of **L** nodes and given a number **N**. The task is to find the N^{th} node from the end of the linked list.

Example 1:

Input:
N = 2
LinkedList: 1->2->3->4->5->6->7->8->9
Output: 8
Explanation: In the first example, there are 9 nodes in linked list and we need to find 2nd node from end. 2nd node from end is 8.

Example 2:


```
// } Driver Code Ends



//Function to find the data of nth node from the end of a linked list.
int getNthFromLast(Node *head, int n)
{
       // Your code here
       if(head == NULL){
           return -1;
       }
       int cnt = 0;
       Node *temp = head;
       while(temp != NULL){
           cnt++;
           temp = temp->next;
       }
       if(n > cnt){
           return -1;
       }
       temp = head;
       while(temp != NULL && cnt != n){
           temp = temp->next;
           cnt--;
       }
       return (temp->data);
}
```