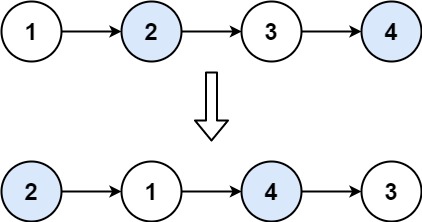
[**24. Swap Nodes in Pairs**](https://leetcode.com/problems/swap-nodes-in-pairs/)

Given a linked list, swap every two adjacent nodes and return its head. You must solve the problem without modifying the values in the list's nodes (i.e., only nodes themselves may be changed.)

**Example 1:**



**Input:** head = [1,2,3,4]

**Output:** [2,1,4,3]

**Example 2:**

**Input:** head = []

**Output:** []

**Example 3:**

**Input:** head = [1]

**Output:** [1]

**Constraints:**

* The number of nodes in the list is in the range [0, 100].
* 0 <= Node.val <= 100

Approach 1:

If swap the value of node but not say in the question:

1. We consider two node and both initialise with head and next head.
2. Traversal the node by pair and swap the value.

Code:

Class ListNode

{

Public:

Int val;

ListNode \* next;

ListNode(int val)

{

This.val=val;

This.next=NULL;  
}  
};

Int main ()

{

ListNode \*first,\*temp,\*second;

temp=new ListNode(Arr[0]);

first=temp;

second=temp;

second->next=NULL:

for(int i=1;i<n;i++)

{

temp=new ListNode(Arr[i]);

second->next=temp;

second=temp;

second->next=NULL;

}

swapPairs(first);

return 0;

}

ListNode\* swapPairs(ListNode\* head) {

if(head==NULL)return head;

ListNode \*temp=head,\*temp1=head->next;

while(temp1!=NULL&&temp!=NULL)

{

swap(temp->val,temp1->val);

temp=temp1->next;

if(temp!=NULL)temp1=temp->next;

}

return head;

}

Time Complexity: O (N)

Space Complexity : O (1)

Approach 2:

By without swap the value with swap the node:

1. We consider the four node type variable and in first assign the head.
2. Also create one dummy Lisklist assign the node prev.
3. Prev next assign to head.
4. After traversal all the node and swap the node.
5. Return dummy next.

Code:

Edge case:

1. If number of node is odd.
2. If head is empty return head.

 ListNode\* swapPairs(ListNode\* head) {

      if(head==NULL) return head;

      ListNode \*first=head,\*prev;

      ListNode \*dummy=new ListNode(0);

      prev=dummy;

      prev->next=head;

      while(first!=NULL&&first->next!=NULL)

      {

         ListNode \*second=first->next;

         ListNode \*future=first->next->next;

         second->next=first;

         prev->next=second;

         first->next=future;

         prev=first;

         first=future;

      }

      return dummy->next;

    }

Time Complexity: O (N)

Space Complexity : O (1)

==================================================================================

[**143. Reorder List**](https://leetcode.com/problems/reorder-list/)

You are given the head of a singly linked-list. The list can be represented as:

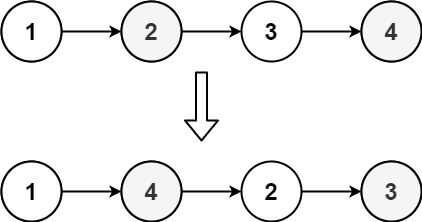
L0 → L1 → … → Ln - 1 → Ln

*Reorder the list to be on the following form:*

L0 → Ln → L1 → Ln - 1 → L2 → Ln - 2 → …

You may not modify the values in the list's nodes. Only nodes themselves may be changed.

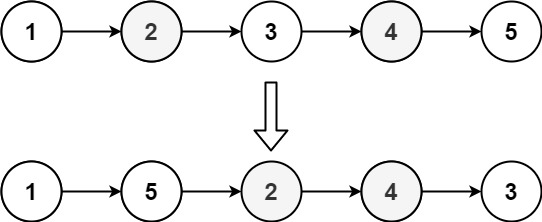
**Example 1:**



**Input:** head = [1,2,3,4]

**Output:** [1,4,2,3]

**Example 2:**



**Input:** head = [1,2,3,4,5]

**Output:** [1,5,2,4,3]

**Constraints:**

* The number of nodes in the list is in the range [1, 5 \* 104].
* 1 <= Node.val <= 1000

Approach 1:

1. Find the middle of the link list node.
2. Reverse the last half of the link list.
3. After that combine both the link list.

Edge case:

1. If list having one or two node return same list.
2. We can also manage the case of total number of odd or even link list. (carefully)

Code:

Reverse link list using three list pointer:

 ListNode\* reverse(ListNode \*head)

    {

        ListNode \*prev=NULL,\*secprev=NULL, \*curr=head;

        while(curr!=NULL)

        {

            secprev=prev;

            prev=curr;

            curr=curr->next;

            prev->next=secprev;

        }

        return prev;

    }

Find the middel of list node:

* 1. In the case if list having two middle node it will be return the first node of the list.
* Input
* head =[1,2,3,4]
* Output [2,3,4]
* Input
* head =[1,2,3,4,5]
* Output[3,4,5]
* head =[1,2]
* Output[1,2]

 ListNode\* middle(ListNode \*head)

    {

        ListNode \*slow ,\*fast;

        slow=head;

        fast=head;

        while(fast->next!=NULL&&fast->next->next!=NULL)

        {

            slow=slow->next;

            fast=fast->next->next;

        }

        return slow;

    }

2. In the case if list having two middle node it will be return the second node of the list.

 ListNode\* middleNodecase(ListNode\* head) {

        if(head==NULL)return head;

        ListNode \*slow ,\*fast;

        slow=head;

        fast=head;

        while(fast!=NULL&&fast->next!=NULL)

        {

            slow=slow->next;

            fast=fast->next->next;

        }

        return slow;

    }

* Input
* head =[1,2,3,4]
* Output [3,4]
* Input
* head =[1,2,3,4,5]
* Output[3,4,5]
* head =[1,2]
* Output[2]

Now create list node with array:

#include <iostream>

using namespace std;

class ListNode

{

public:

int val;

ListNode \*next;

ListNode(int val)

{

this->val=val;

this->next=NULL;

}

};

void print(ListNode \*head)

{

while(head)

{

cout<<head->val<<" ";

head=head->next;

}

}

int main() {

// Write C++ code here

// std::cout << "Hello world!";

int arr[]={1,2,3,4,5,6};

ListNode \*head,\*temp,\*second;

temp =new ListNode(arr[0]);

head=temp;

second=temp;

second->next=NULL;

for(int i=1;i<6;i++)

{

temp= new ListNode(arr[i]);

second->next=temp;

second=temp;

second->next=NULL;

}

print(head);

reorderList(ListNode\* head)

return 0;

}

* 1. Merge two list node with the help of four node pointer.
  2. After revere function call we must assign mid next NULL.
  3. Middle function reverse the first node if two node are middle.

void reorderList(ListNode\* head) {

        if(head==NULL&&head->next==NULL)return;

        ListNode \*mid=middle(head);

        ListNode \*rev=reverse(mid->next);

        mid->next=NULL;

        ListNode \*c1=head, \*c2=rev,\*f1=NULL,\*f2=NULL;

        while(c1!=NULL&&c2!=NULL)

        {

            f1=c1->next;

            f2=c2->next;

            c1->next=c2;

            c2->next=f1;

            c1=f1;

            c2=f2;

        }

    }

**Time Complexity :**O(N) [Middle of List] + O(N/2) [Reversing Second Half] + O(N/2) [Connecting both lists]  = O(2N)  = **O(N)**

**Space Complexity :**O(1)

==================================================================================

**Reverse a Doubly Linked List::**

Given a **doubly linked list**of **n**elements. Your task is to **reverse**the doubly linked list **in-place**.

**Example 1:**

**Input:**

LinkedList: 3 <--> 4 <--> 5

**Output:** 5 4 3

**Example 2:**

**Input:**

LinkedList: 75 <--> 122 <--> 59 <--> 196

**Output:** 196 59 122 75

**Your Task:**  
Your task is to complete the given function **reverseDLL()**, which takes **head**reference as argument and this function should **reverse**the elements such that the **tail**becomes the **new head**and all pointers are pointing in the right order. You need to **return**the **new head** of the reversed list. The **printing**and **verification**is done by the **driver**code.

**Expected Time Complexity:**O(n).  
**Expected Auxiliary Space:**O(1).

**Constraints:**  
1 <= number of nodes <= 104  
0 <= value of nodes <= 104

**Approach 1:**

1. **To traversal the linklist.**
2. **Swap the node-> next to node->prev.**
3. **Finally check the node->next if NULL Then return head else return node->prev;**

**Edge case:**

1. **If node is NULL. Return head.**
2. **If number of node is one then return head.**

**Code:**

**Node\* reverseDLL(Node \* head)**

**{**

**//Your code here**

**if(head==NULL)**

**return head;**

**Node \*temp=head,\*curr=NULL;**

**while(temp!=NULL)**

**{**

**curr=temp->prev;**

**temp->prev=temp->next;**

**temp->next=curr;**

**temp=temp->prev;**

**}**

**return curr!=NULL?curr->prev:head;**

**}**

**Time Complexity: O (N)**

**Space Complexity: O(1)**

**========================================================**

### Intersection of two sorted Linked lists

Given **two linked lists**sorted in **increasing order**, create a new linked list representing the **intersection**of the two linked lists. The new linked list should be made with without changing the original lists.

**Note:** The elements of the linked list are not necessarily distinct.

**Example 1:**

**Input:**

LinkedList1 = 1->2->3->4->6

LinkedList2 = 2->4->6->8

**Output:** 2 4 6

**Explanation:** For the given two

linked list, 2, 4 and 6 are the elements

in the intersection.

**Example 2:**

**Input:**

LinkedList1 = 10->20->40->50

LinkedList2 = 15->40

**Output:** 40

**Your Task:**  
You don't have to take any input of print anything. Your task is to complete the function **findIntersection**(), which will take **head**of both of the linked lists as input and should find the**intersection**of two linked list and add all the elements in **intersection**to the **third linked list**and **return the head**of the third linked list.

**Expected Time Complexity** : O(n+m)  
**Expected Auxilliary Space** : O(n+m)  
**Note:** n, m are the size of the respective linked lists.

**Constraints:**  
1 <= size of linked lists <= 5000  
1 <= Data in linked list nodes <= 104

### Approach 1:

### Edge case:

### Any of the head node is NULL return NULL. (NO intersection node)

### Each time create a new node and update the connected node.

### It may be not any common node then return NULL.

### Solution:

### Each time create a new node.

### Create a node the provide the connectivity between newly created node and previous node.

### Statement 2 is valid for both head having same data.

### If head2 data is greater than head1 data move next node of head1.

### If head1 data is greater than head2 data move next node of head2.

### Retuen head or dummy next.

### Code:

### Node\* findIntersection(Node\* head1, Node\* head2)

### {

### // code goes here.

### if(head1==NULL||head2==NULL)return NULL;

### Node\* dummy=new Node(0);

### Node \*prev=dummy;

### int count=0; // may not common node.

### while(head1!=NULL&&head2!=NULL)

### {

### if(head1->data==head2->data)

### {

### Node \*temp=new Node(head1->data);

### prev->next=temp;

### prev=temp;

### head1=head1->next;

### head2=head2->next;

### count=1;

### }

### else if(head1->data > head2->data)

### {

### head2=head2->next;

### }

### else

### {

### head1=head1->next;

### }

### }

### return count==0?NULL:dummy->next;

### }

### **Time Complexity** : O(n+m)

### **Auxilliary Space** : O(n+m)

### ========================================================