Linkedlist Data Structure

All in one

~~DSA 1A~~

(Merge sort & Linkedlist at end)

1. Linkedlist self Made Notes (Implementation, Basic Interview Questions & code in Java well understood)
2. Linkedlist Implementation In Java (~~Spchedelight~~)
3. Code & Interview Questions (spaghetti.com) (Every good question already here with implementation, Approach & solution. I don't think you need to add any GFG or learn GFG.)
4. Interviewbit Learning

Not Dealing with interview questions Related to doubly linked list

⇒ In Interview don't use array or set to solve complex linkedlist questions since linkedlist are made so that we don't depend on array structures. Make use of fast pointer, slow pointer or else first approach should be fast pointer or slow pointer.

Update

⇒ Now I am able to any linkedlist interview basic questions, all I've to do is remember the approach and for reverse just make notes and start thinking and hence can be done with hit and trial.

⇒ For writing linkedlist operations just think and do code and take care of edge cases hence done.

⇒ (Fast Pointer (Slow Pointer) → Never make use of arrays to solve linkedlist interview questions. Because linkedlist

⇒ Recursion & Recursive Approach has still been difficult for me to understand and to construct some code.

⇒ Check for Recursion Notes to understand Recursion better.

⇒ In DP do everything you check with Recursion only to better understand Recursion

LinkedList

12th Jan 2023

Points to Remember

(1) For now not dealing with doubly linkedlist Interview Questions (will check in future)

(2) Only dealing with singly linkedlist Related Questions

(3) Mostly looking for approach and optimizing

(4) Never use array or set to solve linkedlist related interview problems, since linkedlist was made to remove use of arrays.

(5) You better make use of two pointer, fast Pointer or slow pointer like thing Approach

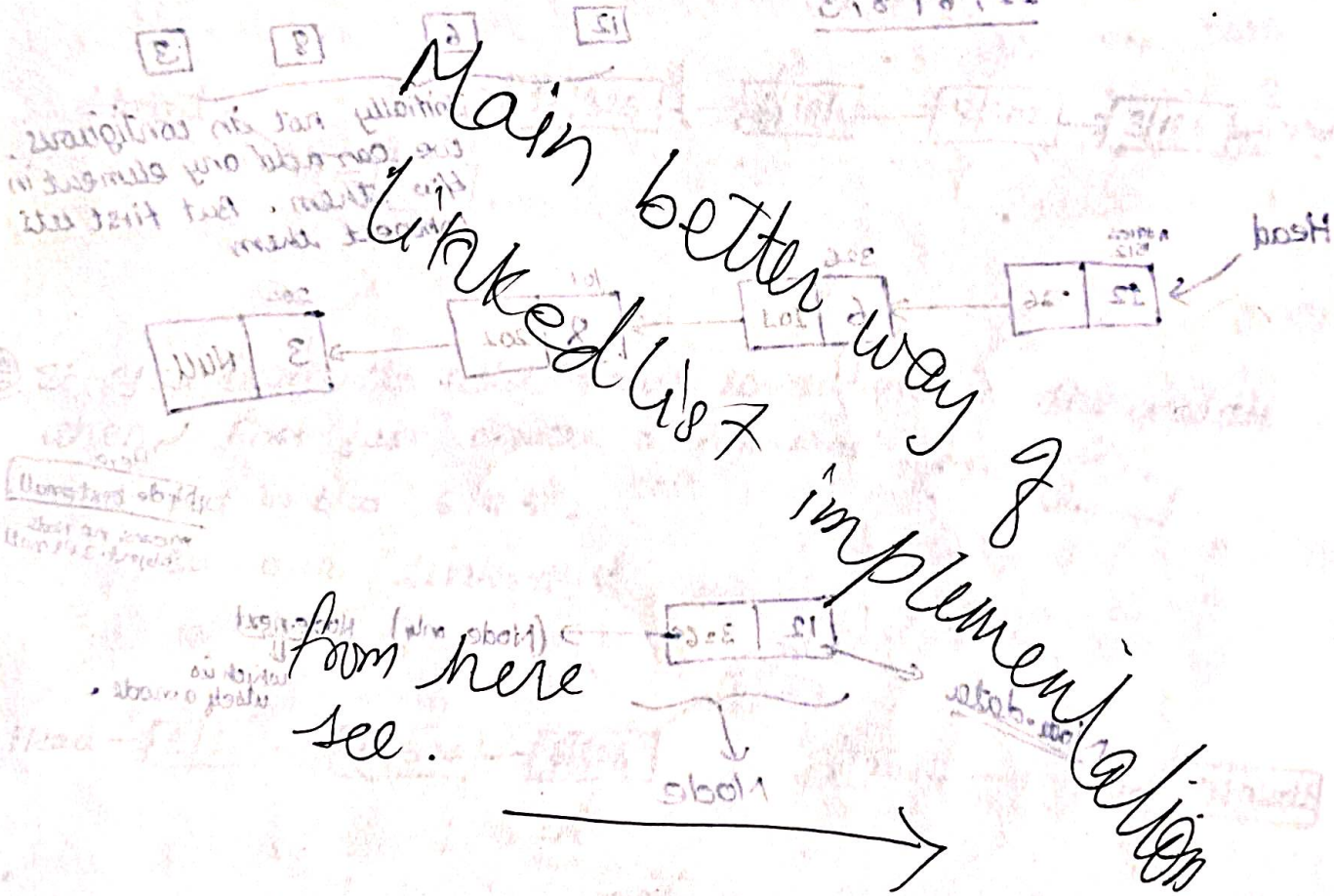
(6) You are now able to solve & write many linkedlist Questions. All you've to do is to start making list in copy with pen & pencil & hit & trial You will always come up with solution Code.

LinkedList :

⇒ Drawback of Arrays :- We cannot expand or shrink it.
But in LinkedList it is possible.

⇒ In LinkedList we will create collection of elements and link each other.

e.g. 12, 6, 8, 3



Main better way of implementation LinkedList from here see.

⇒ Initially we make just Head node. If it's null, then we connect first element of LinkedList with this Head, otherwise we will add each new data node at the end of the LinkedList.

#4 LinkedList :-

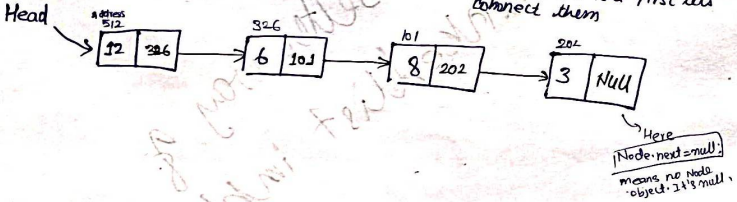
Very deep understanding from Youtube video :- #4 Introduction to Linked List / Data Structures. Similarly continue to #5 & #6 also

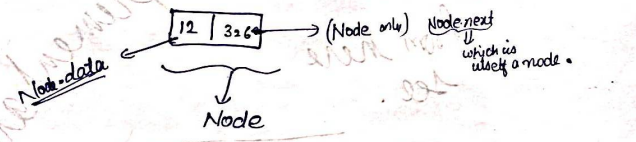
⇒ Deaowback of Anays :- You cannot expand or shrink it. But in LinkedList it is possible.

⇒ In LinkedList we will make collection of elements and link each other.

e.g. 12, 6, 8, 3

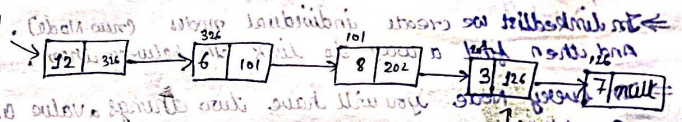
Initially not in contiguous. We can add any element in b/w them. But first lets connect them





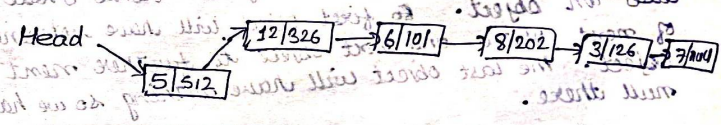
⇒ Initially we make just head Node. If it's null, then we connect first element (Node) of linkedlist with this head, otherwise we will add each new node at the end of the linkedlist.

⇒ So in case we want to add values at the end then:-



⇒ Also if you want to add Node at the start then:-

Now Head won't be pointing to 12|316 but to new Node.

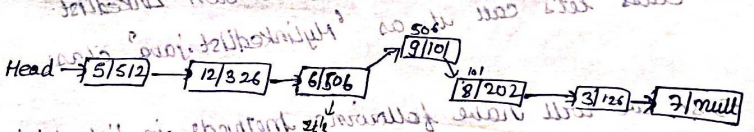


⇒ If you want to add values somewhere in the middle then. First just create a new Node:-

e.g. 506
9

and put in b/w 6 & 8.

So our linkedlist as:-



⇒ So these are singly linkedlist.

⇒ Deaowback of linkedlist ⇒ It is slow as compared to arrays because it works in sequence and has no index in its fetch value randomly. O(n) but in Array O(1).

(#) So code in Main class in PSVM as:-

```
PSVM {
    MyLinkedList list = new MyLinkedList();
    list.insert(18);
    list.insert(45);
    list.insert(12);
    list.show();
}
```

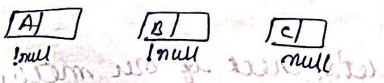
⇒ So now let's run this code:-

- we get output as:-

```
18
45
```

⇒ But we didn't get 12.
It's because it's not getting printed in our "show()" method.

→ Because we are traversing until null. But moment we get null we stop out of while loop. and for that last node whose value = null is not printed, because we've done our printing part inside the while loop.



So it won't print data for object which has null.
So in that case let's do it manually.

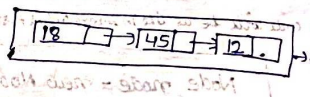
⇒ we will add (sysO(node.data);) outside the while loop for the last element.

⇒ so now again we run our code we get output as:-

```
18
45
12
```

(#) So "show()" method code as:-

```
public void show() {
    Node node = head;
    while(node.next != null) {
        Syso(node.data);
        node = node.next;
    }
    Syso(node.data);
}
```



⇒ So we will talk about other method in next video lectures:-

#) So 'insert()' method code as :- in 'MyLinkedList.java' only

```
public void insert(int data) {
    Node node = new Node();
    node.data = data;
    node.next = null;
    if (head == null) {
        head = node;
    }
    else {
        Node n = head;
        while (n.next != null) {
            n = n.next;
        }
        n.next = node;
    }
}
```

- By default also node.next = null only.
- This means that our head will be current node now (so our current node will become the head)
- Current node is 'node'
- Our node will be the one (head) inside which everything coming now so)

explanation to understand of how actually u'll you can visualize

Eg: LinkedList : 1→2→3→4
Syso(list.head.data) ⇒ prints = 1
Syso(list.head.next.data) ⇒ prints = 2

(So head also contain data since (head = node) [node.data])

If no list then make create new node and assign it head so

§) Now we will implement 'show()' method which will print all the values of our linkedList.

⇒ So to print our values of list, we've to follow the same step, i.e. we've to 'travel' through the Nodes in our linkedList. so similarly some temporary Node node, refer which will take head assigned to it and so now it travels from begining of the list to end uptill we get null. So also while travelling you need to print the data of the node. Also after everytime you print your data, after you've to shift to the next node with {node = node.next} ⇒ so this is how you shift.

so code as :-

```
public void show() {
    Node node = head;
    while (node.next != null) {
        Syso(node.data);
        node = node.next;
    }
}
```

⇒ So now let's check if our methods works fine or not.

⇒ So, to traversal we've to use a temporary node, which will hold the data. For eg: let that our temporary Node be 'n'.

So code as:-

```
else {
    Node n = head;
}
```

→ we've made this temporary node for traversing purpose through the LinkedList to reach last position.

⇒ So while traversing we've do stop the moment we got null as the next node value.

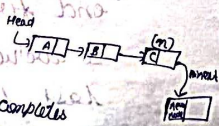
So code as:-

```
else {
    Node n = head;
    while (n.next != null) {
        n = n.next;
    }
}
```

→ so until we found last node whose value is null. this temporary Node 'n' will keep on jumping to next node.

⇒ So this while loop will stop the moment you get null. so our node n has reached to end.

so to that end node e.g. in node



⇒ So after we got n's value null after the iteration completes now we've once we go-to the null we need to change its value. {from null to new node which we've created}

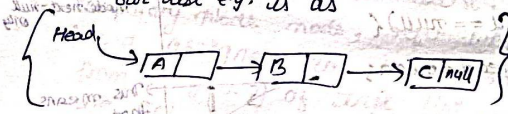
So n.next → will refer to new node which we've created.

⇒ So code as:-

```
n.next = node;
```

(#) ⇒ So how our code worked for our else part is illustrated below:-

our list e.g. is as



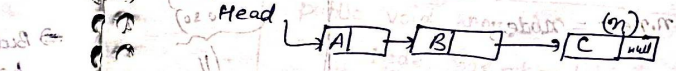
Now we need to insert a new node in it.

So we created a temporary node 'n' which will go upto last until null value is found.

So our node n maps with C

(2)

So link in LinkedList.



Now to this Node n we a give a new reference node which we've created

so

```
n.next = node;
```

→ initially it was n.next = null

So here our while loop stopped so.

(3) So this method for insert() to insert node in our linked list.

In MyLinkedList.java :-

↳ This class will implement all our LinkedList methods:-

⇒ But how would you know that this is our first Node so we use concept of head Node.

So the first Node will be refereed as head Node.

⇒ So in the list itself first we define the head Node:

⇒ So code as :-

```
public class LinkedList {
    Node head;
}
```

By default the value of head is null.

So this will refere to the first Node (which is null)

It will take param 1st first node after creating & assigning it in insert method. 'head=node' ✓

⇒ Now let's implement our methods :-

→ First we will do "insert()" method :-

```
public void insert(int data){
}
```

→ So first we need to assign this 'data' in our Node.

→ So for that first we've to create a Node.

```
Node node = new Node();
```

→ so you will get this node in memory

By default it's value = null.

→ Next we've to mention its two of the values :-

```
node.data = data;
```

→ data from user is assigned.

So we got our first Node.

⇒ But it's possible 'insert()' method add node at the end. But what if there's no linkedlist already there

⇒ So we don't have list itself and hence its our first node. In that scenario we've to make check: so by default the value of our 'head' Node is still null, since it aint pointing to any next Node so,

⇒ So if we are adding our first Object of Node then we do check as :-

{ if our (head == null), then let our first node itself be a head. so our first node itself is a head. }

So code as :-

```
Node node = new Node();
node.data = data;
node.next = null;
if (head == null) {
    head = node;
}
```

in here

→ By default also it's null only. You could have tomit this line. But to make it bit more readable :-

→ head is null means we don't have any nodes in our list.

→ Also by default we already knew that node.next = null. But you can still write it also.

⇒ But if we already have our list already present. Means head is not anymore null then in that case we will go for the else part.

So in this else part we have to add our Node at the last.

But how to reach at the end of LinkedList's node and insert our node there.

For that first we've to iterate till the end first.

So we've to travel from first location to last location.

⇒ So first we go to head and using this head we go to jump to next node & so on until we find null

#5 Linkedlist Implementation in Java Part 1 | Data Structures

Linkedlist Implementation in Java :-

⇒ In Linkedlist we create individual nodes (new Node). And then find a way to link in between them.

⇒ In every Node you will have two things a value and a address of next Node.

⇒ In C++ programming we use pointers but in java we create an reference of next object.

(you know the concept this is)

⇒ So Java is all about Objects. Hence Node is also an object. So first object will have reference of next object and next object to further next object. The last object will have nothing so we have null there.

⇒ So first we want 'Node' object. so create 'Node' class which contains { int data & Node next }

and also after this we need our own Linkedlist class let's call it as 'MyLinkedlist.java' class.

⇒ so we will have following methods in Linkedlist.

① Insert(5) → It will add a new Node with data=5 at the end of our linkedlist.

② insertAt (3, 12) → It will add a new Node with data=12 at location 3 of the linkedlist.
(position, value)

③ insertAtStart(18) → It will a new Node with data=18 at the

④ delete(2) → It will delete the element (Node) at index value 2.
(index)

⑤ show(); → It display all the elements in our linkedlist

so in total we need three classes to work with. For Now.

① Node.java (For Node class.) ② MyLinkedlist.java (For defining and creating our methods implementation) ③ Main.java (for creating)

In Node.java :-

```
public class Node
{
    int data;      → You could also choose long, double, char data types also.
    Node next;     → this Node is a reference for next Node.
}
```

Other methods Implementing :-

From Youtube Videos
#6 LinkedList Implementation in Java part 2 | DataStructures

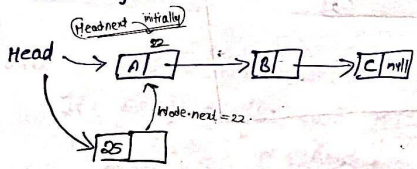
So now we will implement method 'insertAtStart(int data)'. And see how we will insert element at start in our linkedlist. We've already done how to add element in the end.

So our code as :-

```
public void insertAtStart(int data){
    |
}
```

value that we are going to put.

⇒ Now we want to add element at the start. which simply means you just have to change the head location from the current object to the new object.



So our code will be as like in Insert only first part.

```
Node node = new Node();
node.data = data;
node.next = null;

head = node;
```

So our current node will become head node.

But before assigning our node as head first we need to mention the next node of our new node which must point to the initial / previous first node. which was head then.

⇒ So before assigning node to head we want {node.next = head} ; so whatever value was previously having with the head, will be the next for our new node which we've inserted. so our code as :-

For creating a new node code.

```
Node node = new Node();
node.data = data;
node.next = null;
node.next = head;
head = node;
```

And our new current head node will become head node.

So here we are saying if node's next element will be previous head node.

So this is how we insert at start :-

Code for 'insertAtStart()' method :-

```
public void insertAtStart(int data){
    Node node = new Node();
    node.data = data;
    node.next = null;
    node.next = head;
    head = node;
}
```

Now our newly created node will point to head ; so our head is before head now so our node is now like head.

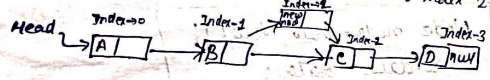
since now it's head, inside which now everything present. you know

explanation / visualisation / formatted page you know

Next method is to insert at Any position :-

⇒ method ⇒ `insertAt(int index, int data)` .

⇒ Index numbers normally starts with Zero. e.g. if index=2 & data=55.
means we want do add node with value 2 at index 2nd.



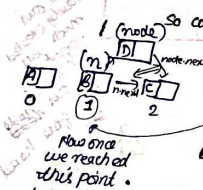
for so Node at index-1 will change address to our new node. And new node which we've created will refere to the next object which initially 'B' was pointing to.

⇒ so first step is just create a new Node :-

code as :-

```
Node node = new Node();
node.data = data;
node.next = null;
```

⇒ Now in this case we've to traverse because we've to specify the index value. so we've to traverse from head to the index value. so to travel again start with that temporary Node n which will be head.
So now the index value is given we can travel by using loop. so with this trace we've to go to that index point.
So if Index value = 2, we've to reach till 1.
Because in 1 we've to make the changes.
So we will use for loop :-



So code as :-

```
for (int i=0 ; i<index-1 ; i++) {
    n = n.next;
}
```

because here we are taken n.next

Now once we reached this point.

we just have to change the address of this node with new node. But before we will have {node.next = n.next} so whatever value in n.next will go to

⇒ Now we've to update n's next i.e. it point to our new node so our job is done.

so our overall code is as :-

```
public void InsertAt(int index, int data) {
    Node node = new Node();
    node.data = data;
    node.next = null;
    Node n = head;
    for (int i=0 ; i<index-1 ; i++) {
        n = n.next;
    }
    node.next = n.next;      // whatever is the next node after n give to our node
    n.next = node;           // And then assign our created node in that position (replacing)
}
```

⇒ But still there is one twist in our code :- for eg. if we add at 0th location ie Index=0 ; the result will be bit different.

so if we do code as :- [list.insertAt(0, 55)] in LinkedList as {25, 18, 45, 12}

it will add 55 at the first location.
i.e. we got output as :- {55, 25, 18, 45, 12} → so it's not working.

⇒ so for zero we've to put a special cod condition.

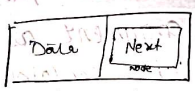
(#) For Now we will be only dealing with single linked list manipulation and understand how things are working.
∴ Singly linked list

⇒ we will also try and look for very optimized way to implement various types of manipulation in linkedlist.

Singly LinkedList

⇒ To understand backstage paradigm of LinkedList.

```
static class Node {
    int data;
    Node next;
}
```

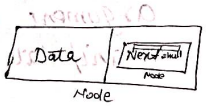


Node class

→ This class can be inner class of MyLinkedList or can be separately defined. It's no big problem.

It most often can also have constructor

```
Node(int data) {
    data = data
    next = null;
}
```



But we generally take default constructor only which will assign data=0 & Next=null by itself rather than assigning ourself.

⇒ Our MyLinkedList class will be as:-

```
class MyLinkedList {
    Node head;
    //various basic functions:
}
```

→ this head alone contains all its node in it.

⇒ So it's an overview of singly linked list now we move forward do some of most often asked PSS or interview questions solved here and understanding it's algorithm.

Linked List Coding Stuffs

#1 very Basic Linked Ust Implementation You will find while solving pss :-

⇒ Owing competitive programming questions related to linkedUst (whether be doubly, singly or circular) there's a backstage LinkedUst implemented already. so they will mostly only give question such as function base competitive programming.

⇒ In that function they will either give argument as 'Head' → Head node (start node) of the linkedUst which contains all the nodes in it.
or they can give argument as List LinkedUst object only where you've to manipulate as per the question asked.

⇒ It's mostly 'head' they will give in argument and that's enough for us to manipulate it.

⇒ The very basic implementation of LinkedUst like "insert", "deleteAt", "insertAtBegin", "display", "delete()", "insertAtPosition" are already done in competitive programming copy. You can refer their and understand how it work!

(5) so just do `Syso("deleted node is", n1.data);`.

⇒ Also if you want to remove this node's n1's to garbage rather than pinted do code as `n1 = null` to nullify it. and eligible for garbage collection.
Because this node object n1 is still there only removed from the list.

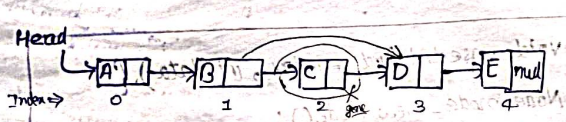
⇒ so now run the code & everything is working fine.

(#) Code for "deleteAt (int index)" method.

```
public void deleteAt (int index) {
        if (index == 0) {
            head = head.next;
        }
        else {
            Node n = head;
            Node n1 = null;
            for (int i = 0; i < index-1; i++) {
                n = n.next;
            }
            n1 = n.next;
            n.next = n1.next;
            Syso("Deleted node is.." + n1.data);
        }
}
```

You could also instead in above code pointing the n1.data, make it null only as (n1 = null) → and of course nothing to point for n1. And it will go for garbage collection.

Now we need to perform delete operation method:- for e.g. Given linkedList as:



So here we want to delete [C] node with index number = 2.

So for that we only need to change the address of [B] node's to [D] instead.

Then our job will be do.

⇒ So let's execute this method :- `deleteAt(int index)` 'deleteAt' method.

⇒ Also we'll have a special scenario here, i.e. what if we want to delete the first element: e.g. node [A], then in that case you've to change the head's location with [B]'s node.

so to achieve that our code as:-

```
if (index == 0) {
    head = head.next;
}
```

→ so if index=0; we change our head's location, i.e to next node's instead of 'A'. which is referred as head.next.

So this is the case for when we want to delete the first element.

⇒ Now we see for other index, rather than 0.

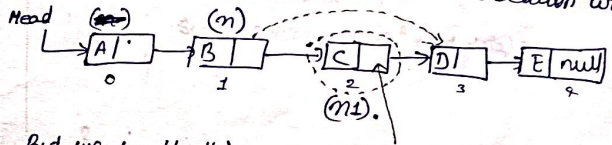
So we will go for else part.

⇒ so in else part we've to traverse with index value once as:-

```
Node n = head;
for (int i=0; i<index-1; i++){
    n = n.next;
}
```

So from the head node we are traversing to that location.

⇒ Now once you reached to that location. so for eg. if we put index=2, then above loop will run only once. So the current n' value after iteration will be



But we don't this value address, so to that we can assign c's address to n (i.e. to B's new address).

⇒ so to do that we need to put c's node to some temporary variables. let c's node be "n1".

so we make node n1. `Node n1 = null;` ← creating a temporary node n1.

and assign n1 as `n1 = n.next;`

n.next → which is eaval to c's node.

⇒ So now after we got node n1, the only thing we've to do now is:-

that n's next need to be replaced by n1's next

so code as:- `n.next = n1.next;`.

So once we delete the value you can also print & check which value

So we will check if index == 0, we will simply call that insertAtStart(data) method.

So our code as:- before Node n=head; as

```
if(index == 0){
    insertAtStart(data);
}
```

→ after this so put other part of the code in else part.

⇒ So this is our overall code to add a node at any location.

⇒ In the next video we will see how to delete an element.

So our overall code for "insertAt(int index, int data)" method:

```
Public void insertAt(int index, int data){
    Node node = new Node();
    node.data = data;
    node.next = null;
    if(index == 0){
        insertAtStart(data);
    }
    else{
        Node n = head;
        for(int i=0; i<index-1; i++){
            n = n.next;
        }
        node.next = n.next;
        n.next = node;
    }
}
```

Other method to detect loop in linked list :-

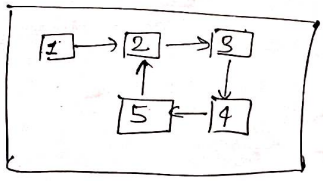
```
int detectLoop (Node head) {
    Node slowP = head;
    Node fastP = head;
    while (slowP != null && fastP != null && fastP.next != null) {
        slowP = slowP.next;
        fastP = fastP.next.next;
        if (slowP == fastP) {
            Sysoc "found Loop");
            return 1;
        }
    }
    return 0;
}
```

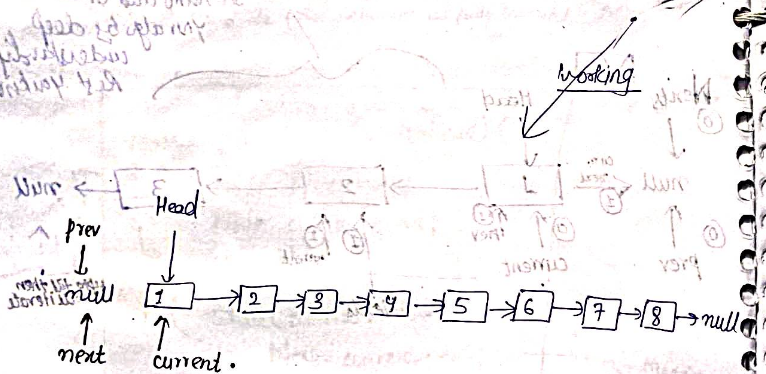
(#) Detect loop in a LinkedList :-

```
//Returns true if there is a loop in Linked List
//else return false.
static boolean detectLoop(Node h) {
    HashSet<Node> s = new HashSet<Node>();
    while (h != null) {
        //If we have already has this node
        //in hashmap it means their is a cycle
        //Because you are encountering the node second time
        if (s.contains(h)) {
            return true;
        }
        //If we are seeing the node for the first time, insert it in hash.
        s.add(h);
        h = h.next;
    }
    return false;
}
```

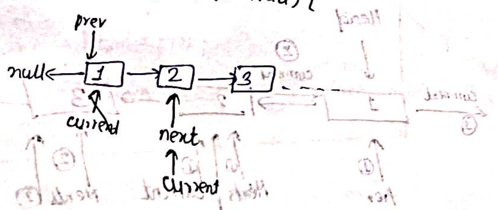
⇒ We are just finding if any node is appeared again while traversing.

⇒ If it appears means the LinkedList is in loop.





⇒ while (count<k && current!=null){



It's working

so now it's working just like reverse only. It's just you've to add extra check as count<k.
and every time new head (next) is passed recursively so fully understand.

Reverse a linked list in groups of given size.

Input ⇒ 1 2 3 4 5 6 7 8

Output ⇒ 3 2 1 6 5 4 8 7

rev rev rev

K=3

code:-
O(n)

```
Node reverse(Node head, int k){
    Node current = head;
    Node next = null;
    Node prev = null;

    int count = 0;

    /*Reverse first k nodes of linked list*/
    while(count<k && current!=null){
        next = current.next;
        current.next = prev;
        prev = current;
        current = next;
        count++;
    }

    /*next is now pointer to (k+1)th node
      Recursively call for the list starting from current.
      And make rest of the list as next of first node*/
    if(next != null){
        head.next = reverse(next, k);
    }

    return prev;
}
```

→ Here it's just normal reverse process you know from previous page

→ to just keep on follow k thing recursive until you reach tail.

② Reverse the linkedlist :-

little hard to understand -

⇒ difficult to understand how the sequence is fully connected in prev.

```
Node reverseList(Node head){
    Node prev=null;
    Node current=head;
    Node Nexts=null;
    while(current!=null){
        Nexts=current.next;
        current.next=prev;
        prev=current;
        current=Nexts;
    }
    head=prev;
    return head;
}
```

3 pointers required

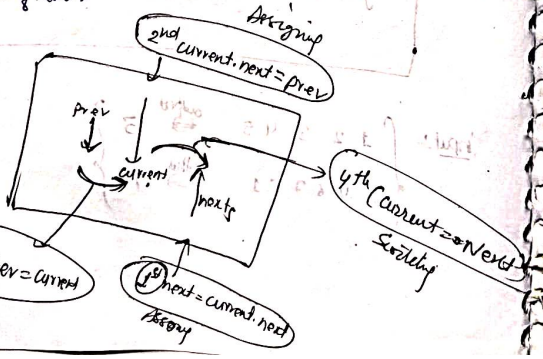
Firstly shift Nexts to next of current so that later we can shift our current to next element

Secondly define the current's next as prev.

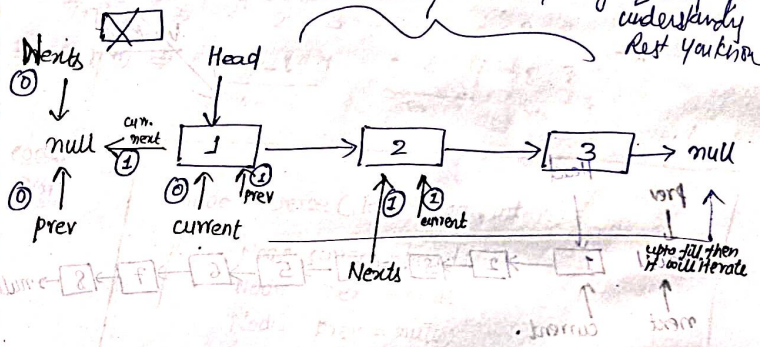
Thirdly move the prev to current's position

And then lastly update current to firstly stored value pointer of Nexts.

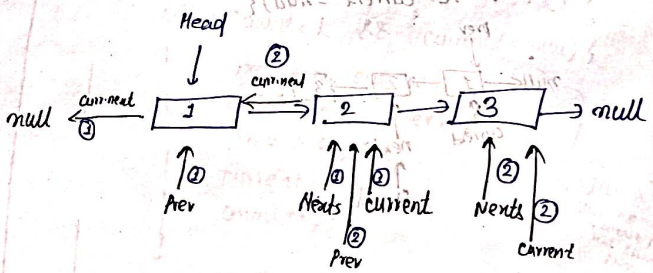
or just "return prev"



Just make this scenario and create your algo by deep understanding Rest you'll see



(while current!=null)



At after ending loop :-

head=prev

return head.

Q① Return the middle element in Linked list :-

⇒ optimized way :- Take two pointer to head.
Slow → one moves by one step both
Fast → other by two step. until fast pointer or its next becomes null.

```
int getMiddle(Node head) {
    Node slow_ptr = head;
    Node fast_ptr = head;
    if (head != null) {
        while (fast_ptr != null && fast_ptr.next != null) {
            fast_ptr = fast_ptr.next.next;
            slow_ptr = slow_ptr.next;
        }
    }
    return slow_ptr.data;
}
```

Write this in first & then next otherwise it was giving null pointer Exception.

Both fast_ptr's only

input :- { 1 2 3 4 5 output ⇒ 3 }
{ 2 4 6 7 5 1 output ⇒ 7 }

#Fully Linkedlist Implementation Code In Java

#① In Node.java

```
public class Node {
    int data;
    Node next;
}
```

#② In MyLinkedlist.java :-

```
public class MyLinkedList {
    Node head;
```

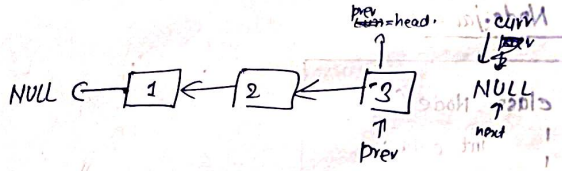
A LinkedList will always have head as its reference

(You can make this class Node here also) like array needed in Stack / Queue this also needed here

```
    //Insert Method
    public void insert(int data) {
        Node node = new Node();
        node.data = data
        node.next = null;
        if(head == null) {
            head = node;
        }
        else {
            Node n = head;
            while(n.next != null) {
                n = n.next;
            }
            n.next = node;
        }
    }
```

In else we can't do head.data = data than just do (head.next = node) in here

After the last iteration the LinkedList will be reversed as :-



and then just assign

head as prev

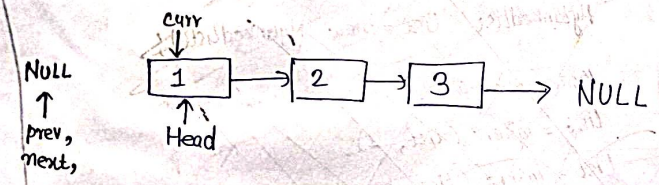
```
while (current!=NULL) {
    next = current->next;
    current->next=prev;
    prev = current;
    current=next;
}
head=prev;
```

working of reverse a LinkedList

Initialise

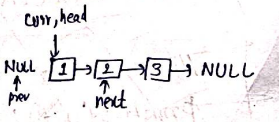
prev ⇒ null , curr ⇒ head , next ⇒ null ✓

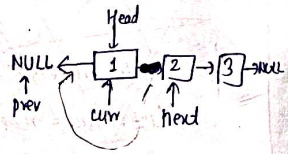
~~next ⇒ curr → next~~

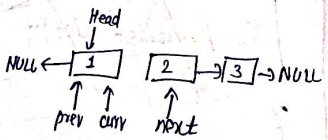


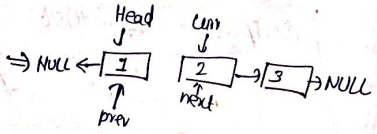
1st iteration

```
while (current != NULL) {
    next = current -> next;
    current -> next = prev;
    prev = current;
    current = next;
}
```

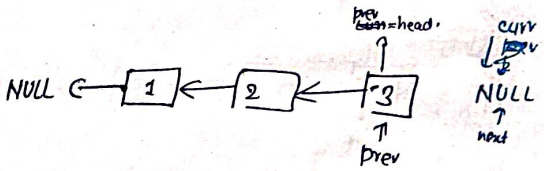








After the last iteration the linkedlist will be reversed as :



and then just assign head as prev

```
while (current!=NULL) {
    next = current->next;
    current->next = prev;
    prev = current;
    current=next;
}
head=prev;
```

(5) code to reverse the LinkedList

```
static Node reverse (Node head) {
    Node prev = null;
    Node current = head;
    Node next = null;
    while (current != null) {
        next = current.next;
        current.next = prev;
        prev = current;
        current = next;
    }
    head = prev;
    return head;
}
```

(5) In PSVM AGAIN

```
MyLinkedList list = new MyLinkedList();
// Insert value in LinkedList in the head modification code node
list = insert (list, 1);
list = insert (list, 2);
list = insert (list, 3);
list = insert (list, 4);
// Print the list
printList(list); // or show();
// After reverse:
list.head = reverse(list.head);
printList(list); // or show();
}
```

Output:- LinkedList: 1 → 2 → 3 → 4 → Null

LinkedList: 4 → 3 → 2 → 1 → Null

Source code "Helene Linux / Idea projects / LinkedList 2" in github.

```
//deleteAt method
public void deleteAt(int index){
    if(index==0){
        head=head.next;
    }else{
        Node n=head;
        Node n1=null;
        for(int i=0; i<index-1; i++){
            n=n.next;
        }
        n1=n.next;
        n.next=n1.next;
        Sysо("Deleted node is"+n1.data);
    }
}
```

(#3) In Main :-

```
P S V M {
    MyLinkedlist list = new MyLinkedList();
    list.insert(18);
    list.insert(45);
    list.insert(12);
    list.show();
}
```

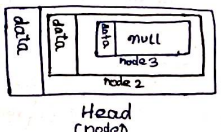
① Head initialised. (MyLinke..new...();) — null (Head)

② list.insert(18) — 18 | null (Head (node))

③ list.insert(45) — 18 | → | 45 | null (Head (node)) (node)

④ list.insert(12) — 18 | → | 45 | → | 12 | null (Head) node1, node2, node3

Actual visualisation.



```
//display/show method
public void show() {
    Node node = head;
    while (node.next != null) {
        Syso (node.data);
        node = node.next;
    }
    Syso (node.data);
}

//insertAtStart method
public void insertAtStart (int data) {
    Node node = new Node();
    node.data = data;
    node.next = null;

    node.next = head;
    head = node;
}
```

```
//insertAt method (In any position/index)
public void insertAt (int index, int data) {
    Node node = new Node();
    node.data = data;
    node.next = null;
    if (index == 0) {
        insertAtStart(data);
    }
    else {
        Node n = head;
        for (int i=0; i<index-1; i++) {
            n = n.next;
        }
        node.next = n.next;
        n.next = node;
    }
}
```

Linked List Implementation in Java

We know that the `LinkedList` class in Java is a doubly-linked list implementation of the `List` interface. This post provides an overview of common techniques to implement a linked list in Java programming language.

We know that each node of a linked list contains a single data field and a reference to the next node in the list. The nodes of the linked list are allocated in the heap memory during runtime by the JVM. We can use the constructor of the `Node` class to initialize the `data` field and the `next` pointer.

```
1  // A Linked List Node
2  class Node
3  {
4      int data;
5      Node next;
6
7      // constructor
8      Node(int data, Node next)
9      {
10         this.data = data;
11         this.next = next;
12     }
13 }
```

Practice this problem

There are several methods to construct a singly linked list in Java:

A1

(LinkedList Implementation in Java)

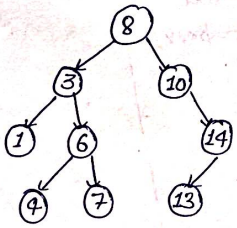
```java
package com.company;
import java.util.*;
import java.lang.*;
import java.io.*;

public  class Main{

    public static void main(String[] args) {

        /** Now We can as much possible linkedlist from it */

        MyLinkedList list = new MyLinkedList();
        list.display();
        list.add(1);
        list.display();

        list.add(2);
        list.add(3);
        list.add(4);
        list.add(5);
        list.display();

        MyLinkedList list2= new MyLinkedList();
        list2.display();
        list2.add(10);
        list2.add(11);
        list2.display();


    }
}
```

```
            System.out.println();
        }
    }
}
```

28 Binary Search Tree Implemented

1) In Node.java :-

```
class Node {
    int data;
    Node left;
    Node right;
}
```



our resultant tree after insert

⇒ BST for PSS

→ Inorder Traversal of BST gives sorted manner.

File: /home/vicky/IdeaProjects/Amaz.../com/company/MyLinkedList.javaPage 1 of 2

This is to show you how head is used and what is node in linkedlist.

```
package com.company;

public class MyLinkedList {

    /**  These are the members we required in building a linked list */

    public Node head;

    /**  Node class is part of linkedlist member like array in stack and queue too
Node is like data member)*/

    public class Node{
        int data;
        Node next;
        public Node(){
            this.data=0;
            this.next=null;
        }
    }

    /**  Below are some methods in linked list */

Method 1 /**    Adding a new node in linkedlist   */

    public Node add(int data){

        /** first create a new node which you want to add */

        Node newNode= new Node();
        newNode.data=data;
        newNode.next=null;

        /** Now check if that linkedlist or head is empty or not. If it's emmpty
then that node will become head of our linkedlist*/

        if(head==null){
            head=newNode;
        }
        else{  /** else take a pointer and move to end of the head and add your
new node there */
            Node pointerUntilEnd=head;
            while (pointerUntilEnd.next!=null){
                pointerUntilEnd=pointerUntilEnd.next;
            }

            /** Now we haave reached that pointer to end of our linkedlist or
head. so now add our new node over there */
            pointerUntilEnd.next=newNode;

        }
        return head;
    }

Method2 /**  Method to display our linkedlist */

    public void display(){
        if(head==null){
            System.out.println("Linkedlist is empty");
        }
        else{
            Node tempPointer=head;
            while (tempPointer!=null){
                System.out.print(tempPointer.data+ "->");
                tempPointer=tempPointer.next;
            }
```

A simple solution would be to allocate memory for all individual nodes of the linked list, set their data, and rearrange their references to build the complete list.

Construct three linked list nodes

Rearrange the pointers to construct a list

```
// A Linked List Node
class Node
{
    int data;
    Node next;

    Node(int data)
    {
        this.data = data;
        this.next = null;
    }
}

class Main
{
    // Helper function to print a given linked list
    public static void printList(Node head)
    {
        Node ptr = head;
        while (ptr != null)
        {
            System.out.print(ptr.data + " -> ");
            ptr = ptr.next;
        }

        System.out.println("null");
    }

    // Naive function for linked list implementation containing three
    public static Node constructList()
    {
        // construct linked list nodes
        Node first = new Node(1);
        Node second = new Node(2);
        Node third = new Node(3);
        Node fourth = new Node(4);

        // rearrange the references to construct a list
```

```
41          second.next = third;
42          third.next = fourth;
43
44          // return reference to the first node in the list
45          return head;
46      }
47
48      public static void main(String[] args)
49      {
50          // `head` points to the head node of the linked list
51          Node head = constructList();
52
53          // print linked list
54          printList(head);
55      }
56  }
```

Download Run Code

We can write the above code in a single line by passing the next node as an argument to the Node constructor:

```
1   // A Linked List Node
2   class Node
3   {
4       int data;
5       Node next;
6
7       Node(int data, Node next_node)
8       {
9           // Set data
10          this.data = data;
11
12          // set the next field to point to a given node of the list
13          this.next = next_node;
14      }
15  }
16
17  class Main
18  {
19      // Helper function to print a given linked list
20      public static void printList(Node head)
21      {
22          Node ptr = head;
23          while (ptr != null)
24          {
25              System.out.print(ptr.data + " -> ");
26              ptr = ptr.next;
27          }
28
29          System.out.println("null");
30      }
```

Java Linked List Interview Questions

Linked List

Java Linked List Interview Questions, Programs and Algorithms

A2

Code spaghetti.com

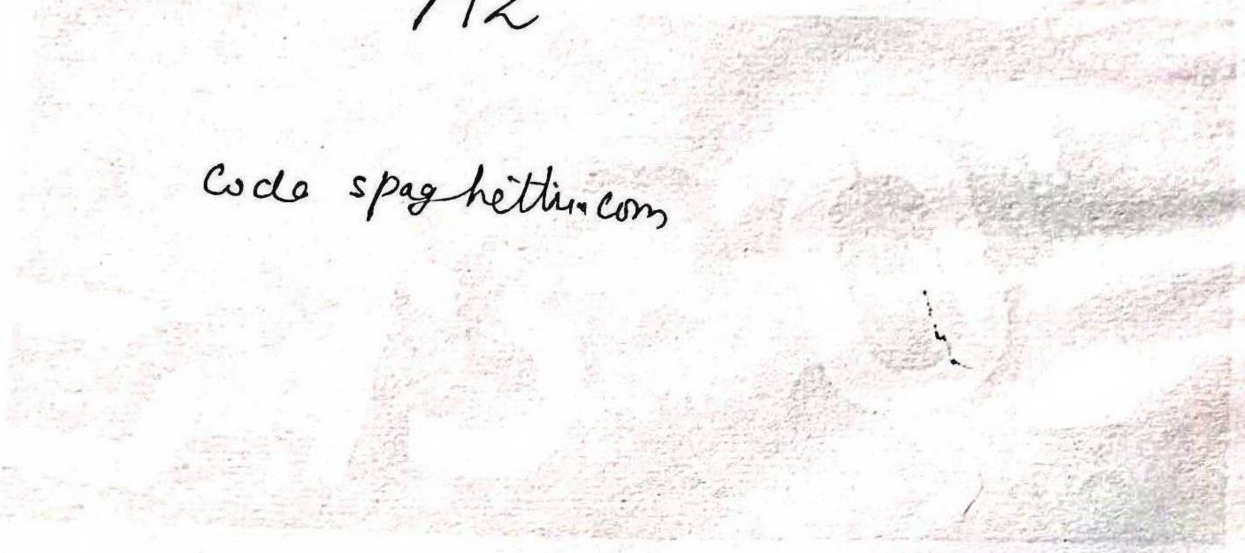


Table of Contents

[illegible] take the time to master a simple programming concept. You develop skills which make you successful in the technical interviews.

It is especially true for data structures and particularly for linked lists.

Usually, a lot of candidates can handle questions about arrays. But they get trapped in linked list interview questions.

What if we can pinpoint the questions that are proven to be the most frequent questions asked in programming interviews?

Then, of course, you can prepare and answer those questions, while others have failed.

In this way, you will immediately separate yourself from the crowd of ordinary candidates.

```
53      {
54          // input keys
55          int[] keys = { 1, 2, 3, 4 };
56
57          // points to the head node of the linked list
58          constructList(keys);
59
60          // print linked list
61          printList(head);
62      }
63  }
```

Download Run Code

Continue Reading:

Linked List – Insertion at Tail | C, Java, and Python Implementation

Also See:

Linked List Implementation in C

Linked List Implementation in C++

Linked List Implementation in Python

References: http://cslibrary.stanford.edu/103/LinkedListBasics.pdf

3. Make head reference global

We can construct a linked list by making the head reference global, but this approach is not recommended since global variables are usually considered bad practice.

```
1  // A Linked List Node
2  class Node
3  {
4      int data;
5      Node next;
6  }
7
8  class Main
9  {
10     // Helper function to print a given linked list
11     public static void printList(Node head)
12     {
13         Node ptr = head;
14         while (ptr != null)
15         {
16             System.out.print(ptr.data + " -> ");
17             ptr = ptr.next;
18         }
19         System.out.println("null");
20     }
21
22     // global head
23     public static Node head;
24
25     // Takes a list and a data value, creates a new link with the give
26     // data and pushes it onto the list's front.
27     public static Node push(int data)
28     {
29         // allocate a new node in a heap and set its data
30         Node newNode = new Node();
31         newNode.data = data;
32
33         // set the next field of the new node to point to the current
34         // head node of the list.
35         newNode.next = head;
36
37         // change the head to point to the new node, so it is
38         // now the first node in the list.
39
40         return newNode;
41     }
42
43     // Function for linked list implementation from the given set of k
44     public static void constructList(int[] keys)
45     {
46         // start from the end of the array
47         for (int i = keys.length - 1; i >= 0; i--) {
48             head = push(keys[i]);
49         }
50     }
```

```
{
    // Helper function to print a given linked list
    public static void printList(Node head)
    {
        Node ptr = head;
        while (ptr != null)
        {
            System.out.print(ptr.data + " –> ");
            ptr = ptr.next;
        }
        System.out.println("null");
    }

    public static Node push(Node head, int data)
    {
        // allocate a new node in a heap and set its data
        Node newNode = new Node();
        newNode.data = data;

        // set the next field of the new node to point to the current
        // first node of the list.

        newNode.next = head;

        // change the head to point to the new node, so it is
        // now the first node in the list.

        return newNode;
    }

    // Function for linked list implementation from the given set of k
    public static Node constructList(int[] keys)
    {
        Node head = null;

        // start from the end of the array
        for (int i = keys.length - 1; i >= 0; i--) {
            head = push(head, keys[i]);
        }

        return head;
    }

    public static void main(String[] args)
    {
        // input keys
        int[] keys = { 1, 2, 3, 4 };

        // points to the head node of the linked list
        Node head = constructList(keys);

        // print linked list
        printList(head);
    }
}
```

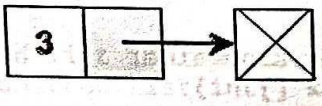
Download Run Code

```
34      {
35          Node head = new Node(1, new Node(2, new Node(3, null)));
36          return head;
37      }
38
39      public static void main(String[] args)
40      {
41          // `head` points to the head node of the linked list
42          Node head = constructList();
43
44          // print linked list
45          printList(head);
46      }
47  }
```

Download Run Code

2. Return Head Node

The standard solution adds a single node to the head end of any list. This function is called `push()` since we are adding the link to the head end, making a list look a bit like a stack.



This is demonstrated below, where we return the head node from the `push()` function and update the head in the caller.

```
1   // A Linked List Node
2   class Node
3   {
4       int data;
5       Node next;
6   }
```

Java Linked List Interview Questions

codespaghetti.com/linked-list-interview/

Linked List

Java LinkedList Interview Questions, Programs and Algorithms.

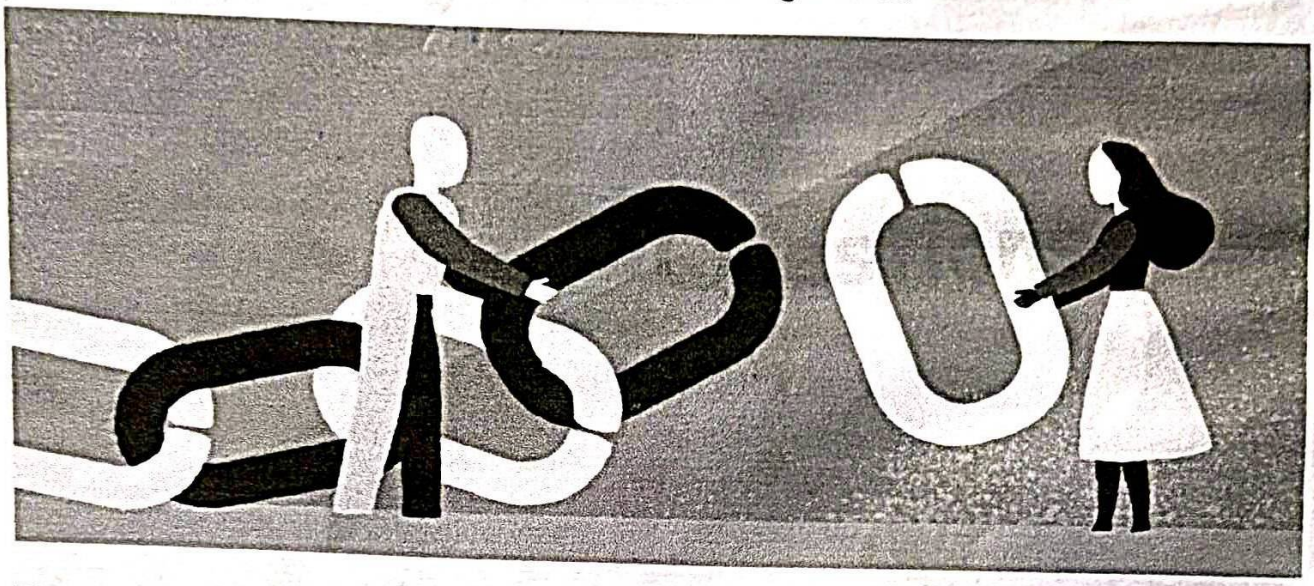


Table of Contents:

When you take the time to master a simple programming concept, You develop skills which can make you successful in the technical interviews.

It is especially true for data structures and particularly for linked lists.

Usually, a lot of candidates can handle questions about arrays. But they get trapped in linked list interview questions.

What if we can pinpoint the questions that are proven to be the most frequent questions asked in programming interviews?

Then, of course, you can prepare and answer those questions, where others have failed.

In this way, you will immediately separate yourself from the crowd of ordinary candidates.

In this tutorial, I'm going to show you exactly how to prepare, answer, and solve linked list questions and algorithms in programming interviews.

But before we continue, Let's have a look at the top five linked list questions asked in interviews.

Top five Java Linked List Questions Asked in Interviews?

Question: How to describe Java Linked List in interview?

- Java LinkedList is the Doubly-linked list implementation of the list interface
- LinkedList can be iterated in reverse direction using descendingIterator()
- LinkedList can contain duplicate elements.
- LinkedList maintains insertion order.
- LinkedList is not synchronized.
- LinkedList manipulation is fast because no shifting needs to be occurred.
- LinkedList can be used as list, stack or queue.

Question: What is the performance of different linked list operations in terms of big 'O' notation.

Operations	Access	Search	Insertion	Deletion
Single Linked list	O(n) Slow	O(n) Slow	O(1) Fast	O(1) Fast
Doubly Linked list	O(n) Slow	O(n) Slow	O(1) Fast	O(1) Fast

how deletion is O(1) in case we want to deleted any random element

The Solution:

In order to find intersection of two linked lists, we will create a new linked lists and copy all the elements which are same in both linked lists. Order of elements in new linked list does not matter.

And how exactly we will achieve that ?

Well there are two different ways

Algorithm 1

Create a new linked list called result and Initialize as NULL. Traverse list1 and look for its each element in list2, if the element is present in list2, then add the element to result. **Time complexity:** Time Complexity for this algorithm is **O(mn) m is the number of elements in first list n is the number of elements in second list**

Algorithm 2:

The third algorithm for the implementation of this problem is very simple, here are steps

1. Create a new **HashMap.**
2. Traverse **second linked list** and store data of each node in a map as a key and false (Boolean) as a value.
3. Traverse **first linked list,** check if data (data stored in the node) is present in a map (as a key) and its value is false. if it is present then we create a new node with that data, add it to the new linked list and update its value to true in map. Otherwise continue.

Algorithm3:

In this method we will use merge sort.

Following are the steps to be followed to get intersection of lists.

1) Sort the first Linked List using merge sort. This step takes O(mLogm) time.2) Sort the second Linked List using merge sort. This step takes O(nLogn) time. 3) Linearly scan both sorted lists to get the union and intersection. This step takes O(m + n) time.

Time complexity:

Time complexity of this method is O(mLogm + nLogn) which is better than method 1's time complexity.

Code Implementation Algorithm 1:

```
// Java program to find union and intersection of two unsorted linked lists
class LinkedList
```

SOBDD.

Remember! When solving linkedlist Questions or in ₹ interview also, never use 'Arrays' to solve linkedlist problems, Because we are using linkedlist because we're taken consideration that linkedlist is better than array. So don't use Array now.

Performance		Structure
Performance of ArrayList and LinkedList depends on the type of operation		ArrayList is the resizable array implementation of list interface
Get:	ArrayList: O(1) LinkedList: O(n)	While LinkedList is the Doubly-linked list implementation of the list interface.
Add:	ArrayList: O(n) LinkedList: O(1)	
Remove:	ArrayList: O(n) LinkedList: O(1)	

Memory overhead

Memory overhead in LinkedList is more as compared to ArrayList as node in LinkedList needs to maintain the addresses of next and previous node.

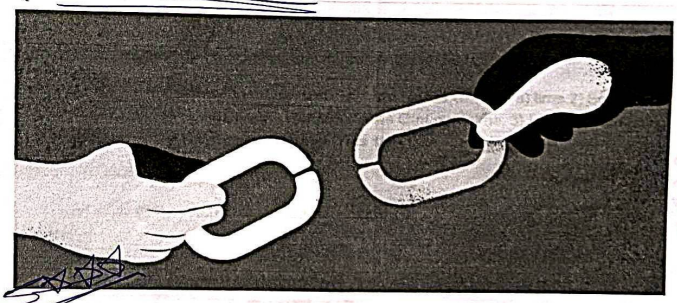
While in ArrayList each index only holdsthe actual object.

Iteration

LinkedList can be iterated in reverse direction using descendingIterator().

ArrayList has no descending Iterator(),So we need to write our own code to iterate over the ArrayList in reverse

Java Linked List Algorithm Questions



Question: Find union and Intersection of two Linked Lists in Java? [Microsoft]

Union means like union in Venn Diagram → Return all value. Duplicate only return once.

Intersection means only those which are

maybe we can use set

java.util.AbstractSequentialList

Java Linked List Vs Other Data Structures

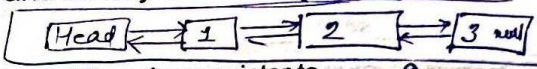
Java Linked List Vs Array

Answer: If we maintain two pointers, and we increment one pointer after processing two nodes and other after processing every node.

We are likely to find a situation where both the pointers will be pointing to same node. This will only happen if linked list has loop.

Question: What is difference between Singly Linked List and Doubly Linked List in Java?

The Main difference between singly linked list and doubly linked list is ability to traverse.



doubly linked list

In a single linked list, node only points towards next node, and there is no pointer to previous node. Which means you can not traverse back on a singly linked list.

On the other hand doubly linked list maintains two pointers, towards next and previous node, which allows you to navigate in both direction in any linked list.

Question: How to find 3rd element from end in a linked list in one pass?

If we apply a trick of maintaining two pointers and increment other pointer, when first has moved up to 3rd element.

find pointer restart from 3rd node and keep traversing

Than when first pointer reaches to the end of linked list, second pointer will be pointing to the 3rd element from last in a linked list.

Question: Which interfaces are implemented by Linked List in Java?

Following interfaces are implemented by Java Linked lists

```
Serializable, Queue, List, Cloneable, Collection, Deque, Iterable
```

Question: What is the package name for Linked List class in Java?

```
java.util
```

Question: What class is the Parent Class of LinkedList class?

Question: What are advantages and disadvantages of linked lists?

Advantages of linked lists in Java:

- Linked lists have constant-time insertion and deletion operation in any position. Arrays require O(n)O(n) time to do the same thing, because you'd have to "shift" all the subsequent items over 1 index.
- Linked lists can continue to expand as long as there is space on the machine. Arrays (in low-level languages) must have their size specified ahead of time.Even if array is dynamic like array list that automatically resize themselves when they run out of space. the operation to resize a dynamic array has a large cost which can make a single insertion unexpectedly expensive.

Disadvantages of linked lists in Java:

To access or edit an item in a linked list, you have to take O(i)O(i) time to walk from the head of the list to the iith item (unless of course you already have a pointer directly to that item). Arrays have constant-time lookups and edits to the iith item.

Question: How many types of Linked lists exists?

Java Linked lists can be categorized into three types:

Single linked: lists have a single pointer pointing to the next element in the list. The last pointer is empty or points to null, meaning the end of the list.

Doubly linked: lists have two pointers, one pointing to the next element and one pointing to the previous element.

The head node's previous pointer points to null and the tail node's next pointer points to null to signal the end of the list.

Circular linked: lists usually represent buffers. They have no head or tail, and the primary issue is avoiding infinite traversals because of the cycle.

Arrays are oftentimes a better substitute for a circular linked list, using the modulus operator to wrap around.

Question: How to find if linked list has a loop?

```
public class Main {

   public static void main(String[] args) {
     CreateLinkedList createLinkedList = new CreateLinkedList();
     String[] firstListData = {"a","c","c","d"};
     Node firstList = createLinkedList.createList(firstListData);
     String[] secondListData = {"c","d","e","a"};
     Node secondList = createLinkedList.createList(secondListData);

     IntersectionLinkedList intersectionLinkedList = new IntersectionLinkedList();
     Node intersectionList = intersectionLinkedList.intersection(firstList,
secondList);

     while(intersectionList != null){
       System.out.println(intersectionList.getData());
       intersectionList = intersectionList.getNext();
     }
   }
}
```

Result:

```
a
c
d
```

Find Union of two linkedLists

You have given two linked list, create a linked list containing all the elements of the given two linked list excluding duplicates. i.e. create union of two given linked list. Order of elements in new linked list does not matter.

Algorithm 1:

Create a new linked list called result and Initialize as NULL. Traverse list1 and add all of its elements to the result.

Traverse list2. If an element of list2 is already present in result then do not insert it to result, otherwise insert.

Time complexity:

Time Complexity for this algorithm is **O(mn) m is the number of elements in first listn is the number of elements in second list. Algorithm 2:**
The algorithm for the implementation of this problem is very simple, here are steps :-

1. Create a new **HashSet.**
2. Traverse **first linked list.**
3. Check data (data stored in the node) is present in a set or not, if it is not present then we create a new node with that data, add it to the new linked list and to the set. Otherwise continue.
4. Repeat step 2 and step 3 for**second linked list.**

```
import java.util.HashMap;
import java.util.Map;

public class IntersectionLinkedList {

   private Node intersectionListHeadNode;

   private Map<String,Boolean> getAllDataOfList(Node list){
     Map<String,Boolean> uniqueMap = new HashMap<String,Boolean>();
     while(list != null){
       uniqueMap.put(list.getData(),false);
       list = list.getNext();
     }
     return uniqueMap;
   }

   public Node intersection(Node firstList, Node secondList){
     Map<String,Boolean> uniqueMap = getAllDataOfList(secondList);
     addList(firstList,uniqueMap);
     return this.intersectionListHeadNode;
   }

   private void addList(Node sourceList,Map<String,Boolean> uniqueMap){
     Node tempNode = null;
     Node tempList = null;
     while(sourceList != null){
       if(uniqueMap.containsKey(sourceList.getData()) &&
!uniqueMap.get(sourceList.getData())){
         uniqueMap.put(sourceList.getData(), true);
         tempNode = createNode(sourceList.getData());
         if(tempList == null){
           tempList = tempNode;
           this.intersectionListHeadNode = tempList;
         }else{
           tempList.setNext(tempNode);
           tempList = tempNode;
         }
       }
       sourceList = sourceList.getNext();
     }
   }

   private Node createNode(String data){
     Node node = new Node();
     node.setData(data);
     return node;
   }
}
```

Approach 2

4) Create a class Main, this class will be our demo class.

```
public class Node {

 private String data;
 private Node next;

 public String getData() {
  return data;
 }

 public void setData(String data) {
  this.data = data;
 }

 public Node getNext() {
  return next;
 }

 public void setNext(Node next) {
  this.next = next;
 }

}
```

2) Create a class CreateLinkedList, this class has a method createList which takes string array as an argument and return linked list.

```
public class CreateLinkedList {

 public Node createList(String[] dataArr){
  Node listHead = null;
  Node node = null;
  Node tempList = null;
  for(String data : dataArr){
   node = new Node();
   node.setData(data);
   if(listHead == null){
    listHead = node;
    tempList = node;
   }else{
    tempList.setNext(node);
    tempList = node;
   }
  }
  return listHead;
 }

}
```

3) Create a class IntersectionLinkedList, this class has a method intersection which takes two linked list as an argument and return intersection linked list.

```
    /* A utilty function that returns true if data is present
       in linked list  else return false */
    boolean isPresent (Node head, int data)
    {
        Node t = head;
        while (t != null)
        {
            if (t.data == data)
                return true;
            t = t.next;
        }
        return false;
    }

    /* Driver program to test above functions */
    public static void main(String args[])
    {
        LinkedList llist1 = new LinkedList();
        LinkedList llist2 = new LinkedList();
        LinkedList unin = new LinkedList();
        LinkedList intersecn = new LinkedList();

        /*create a linked lits 10->15->5->20 */
        llist1.push(20);
        llist1.push(4);
        llist1.push(15);
        llist1.push(10);

        /*create a linked lits 8->4->2->10 */
        llist2.push(10);
        llist2.push(2);
        llist2.push(4);
        llist2.push(8);

        intersecn.getIntersection(llist1.head, llist2.head);
        unin.getUnion(llist1.head, llist2.head);

        System.out.println("First List is");
        llist1.printList();

        System.out.println("Second List is");
        llist2.printList();

        System.out.println("Intersection List is");
        intersecn.printList();
    }
}
```

Code Implementation Algorithm 2:

1) Create a class **Node**, this class will be a node of our linked list.

```
{
    Node head;  // head of list

    /* Linked list Node*/
    class Node
    {
        int data;
        Node next;
        Node(int d)
        {
            data = d;
            next = null;
        }
    }

    void getIntersection(Node head1, Node head2)
    {
        Node result = null;
        Node t1 = head1;

        // Traverse list1 and search each element of it in list2.
        // If the element is present in list 2, then insert the
        // element to result
        while (t1 != null)
        {
            if (isPresent(head2, t1.data))
                push(t1.data);
            t1 = t1.next;
        }
    }

    /* Utility function to print list */
    void printList()
    {
        Node temp = head;
        while(temp != null)
        {
            System.out.print(temp.data+" ");
            temp = temp.next;
        }
        System.out.println();
    }

    /*  Inserts a node at start of linked list */
    void push(int new_data)
    {
        /* 1 & 2: Allocate the Node &
                  Put in the data*/
        Node new_node = new Node(new_data);

        /* 3. Make next of new Node as head */
        new_node.next = head;

        /* 4. Move the head to point to new Node */
        head = new_node;
    }
```

Code Implementation Algorithm 1:

Rest of the code is same as algorithm implementation 1 for intersection of linked lists

```
void getUnion(Node head1, Node head2)
 {
     Node t1 = head1, t2 = head2;

     //insert all elements of list1 in the result
     while (t1 != null)
     {
         push(t1.data);
         t1 = t1.next;
     }

     // insert those elements of list2 that are not present
     while (t2 != null)
     {
         if (!isPresent(head, t2.data))
             push(t2.data);
         t2 = t2.next;
     }
 }
```

Code Implementation Algorithm 2:

Rest of the code is same as above implementation for algorithm 2 in intersection of linked lists

Algorithm 3

Not Specified

↳ Since merge sort in LinkedList is bit difficult problem since separately LinkedList and then merging them.

```
                    dup = ptr2.next;
                    ptr2.next = ptr2.next.next;
                    System.gc();
                } else /* This is tricky */ {
                    ptr2 = ptr2.next;
                }
            }
            ptr1 = ptr1.next;
        }
    }

    void printList(Node node) {
        while (node != null) {
            System.out.print(node.data + " ");
            node = node.next;
        }
    }

    public static void main(String[] args) {
        LinkedList list = new LinkedList();
        list.head = new Node(10);
        list.head.next = new Node(12);
        list.head.next.next = new Node(11);
        list.head.next.next.next = new Node(11);
        list.head.next.next.next.next = new Node(12);
        list.head.next.next.next.next.next = new Node(11);
        list.head.next.next.next.next.next.next = new Node(10);

        System.out.println("Linked List before removing duplicates ");
        list.printList(head);

        list.remove_duplicates();
        System.out.println("");
        System.out.println("Linked List after removing duplicates");
        list.printList(head);

    }
}
```

Code implementation of Algorithm 2:

```
import java.util.HashMap;

public class RemoveDuplicates {

    public Node removeDup(Node head){
        HashMap<Integer, Integer> ht = new HashMap<Integer, Integer>();
        if(head==null){
            return null;
        }
        Node currNode = head.next;
        Node prevNode = head;
        Node temp; //keeping it so that last node would be eligible for garbage
collection
        ht.put(head.data, 1);
```

Algorithm 2:

- Create a Hash Table
- Take two pointers, prevNode and CurrNode.
- PrevNode will point to the head of the linked list and currNode will point to the head.next.
- Now navigate through the linked list.
- Check every node data is present in the HashTable.
- if yes then delete that node using prevNode and currNode.
- If No, then insert that node data into the linked list
- Return the head of the list

Performance:

Time Complexity of this solution in terms of big'0 notation is **O(n)**

Code implementation of Algorithm 1:

```
// Java program to remove duplicates from unsorted linked list

class LinkedList {

    static Node head;

    static class Node {

        int data;
        Node next;

        Node(int d) {
            data = d;
            next = null;
        }
    }

    /* Function to remove duplicates from a unsorted linked list */
    void remove_duplicates() {
        Node ptr1 = null, ptr2 = null, dup = null;
        ptr1 = head;

        /* Pick elements one by one */
        while (ptr1 != null && ptr1.next != null) {
            ptr2 = ptr1;

            /* Compare the picked element with rest of the elements */
            while (ptr2.next != null) {

                /* If duplicate then delete it */
                if (ptr1.data == ptr2.next.data) {

                    /* sequence of steps is important here */
```

```
public class MiddleOfList {
public String findMiddleOfList() {
LinkedListNode tail = new LinkedListNode("data5", null);
LinkedListNode node4 = new LinkedListNode("data4", tail);
LinkedListNode node3 = new LinkedListNode("data3", node4);
LinkedListNode node2 = new LinkedListNode("data2", node3);
LinkedListNode head = new LinkedListNode("data1", node2);
return findMiddle(head);
}
private String findMiddle(LinkedListNode head) {
LinkedListNode current = head;
int length = 0;
LinkedListNode middle = head;
while(current.nextNode != null){
length++;
if(length % 2 == 0) {
middle = middle.nextNode;
}
current = current.nextNode;
}
return middle.data;
}
private class LinkedListNode {
public String data = null;
public LinkedListNode nextNode = null;
public LinkedListNode(String data, LinkedListNode nextNode) {
this.data = data;
this.nextNode = nextNode;
}
}
}
```

Question: How to remove Duplicates from a Linked List in Java? [Telephonic]

The Solution

Write a program which can go through a linked list and remove all the duplicate values, For example if the linked list is 12->11->12->21->41->43->21 then our program should convert the list to 12->11->21->41->43.

There are two approaches to achieve this

Algorithm 1:

Simplest way to achieve this is by using two loops. **Outer loop** is used to pick the elements one by one and **inner loop** compares the picked element with rest of the elements.

Performance:

Time Complexity of this solution in terms of big'0 notation is **O(n^2)**

Middle element of a linked list can be found by using two pointers, which will move like this

- **Pointer 1:** Incrementing one at each iteration
- **Pointer 2:** Incrementing at every second iteration

When first pointer will point at end of Linked List, second pointer will be pointing at middle node of Linked List.

Pseudo Code

```
Node current = LinkedListHead;
int length = 0;
Node middle = LinkedListHead;
while(current.next() != null){
length++;
if(length % 2 == 0) {
middle = middle.next();
}
current = current.next();
}
return middle;
```

Complete Solution

```
public class UnionLinkedList {

  private Node unionListHeadNode;

  public Node union(Node firstList, Node secondList){
    Set uniqueSet = new HashSet();
    Node unionListLastNode = addList(firstList,null,uniqueSet);
    unionListLastNode = addList(secondList,unionListLastNode,uniqueSet);
    return this.unionListHeadNode;
  }

  private Node addList(Node sourceList,Node unionListLastNode,Set uniqueSet){
    Node tempNode = null;
    Node tempList = unionListLastNode;
    while(sourceList != null){
      if(!uniqueSet.contains(sourceList.getData())){
        uniqueSet.add(sourceList.getData());
        tempNode = createNode(sourceList.getData());
        if(tempList == null){
          tempList = tempNode;
          this.unionListHeadNode = tempList;
        }else{
          tempList.setNext(tempNode);
          tempList = tempNode;
        }
      }
      sourceList = sourceList.getNext();
    }
    return tempList;
  }

  private Node createNode(String data){
    Node node = new Node();
    node.setData(data);
    return node;
  }

}
```

Question: How to find middle element of linked list in one pass? [Google]

The Problem

Question: How would you find the middle element of a linked list in one pass/loop?

This is one of the starter question in most of technical interviews. so make sure you understand the concept and implementation correctly.

The Solution

```
if(ptrTwo!=null){
  if(ptrTwo.next!=null){
    Node t1 = ptrTwo;
    Node t2 = ptrTwo.next;
    ptrOne.next = prevNode;
    prevNode = ptrOne;
    reverseRecursion(t1,t2, prevNode);
   }
   else{
    ptrTwo.next = ptrOne;
    ptrOne.next = prevNode;
    System.out.println("n Reverse Through Recursion");
    display(ptrTwo);
   }
 }
 else if(ptrOne!=null){
  System.out.println("n Reverse Through Recursion");
  display(ptrOne);
 }
}

public void display(Node head){
 //
 Node currNode = head;
 while(currNode!=null){
  System.out.print("->" + currNode.data);
  currNode=currNode.next;
 }
}
}
```

Output:

```
->30->25->20->15->10->5
Reverse Through Iteration
->5->10->15->20->25->30
```

```
Original Link List 2 : ->36->35->34->33->32->31
Reverse Through Recursion
->31->32->33->34->35->36
```

Question: Move last node to front in Java linked list. [Telephonic Interviews]

Analysis:

Traverse the list till last node. Use two pointers: one to store the address of last node and other for address of second last node. After the end of loop do following operations.

i) Make second last as last (secLast->next = NULL).

ii) Set next of last as head (last->next = *head_ref).

iii) Make last as head (*head_ref = last)

```
 head=null;
}

public void addAtBegin(int data){
 Node n = new Node(data);
 n.next = head;
 head = n;
}
public void reverseIterative(Node head){
 Node currNode = head;
 Node nextNode = null;
 Node prevNode = null;

 while(currNode!=null){
  nextNode = currNode.next;
  currNode.next = prevNode;
  prevNode = currNode;
  currNode = nextNode;
 }
 head = prevNode;
 System.out.println("n Reverse Through Iteration");
 display(head);
}

public void reverseRecursion(Node ptrOne,Node ptrTwo, Node prevNode){
 if(ptrTwo!=null){
   if(ptrTwo.next!=null){
    Node t1 = ptrTwo;
    Node t2 = ptrTwo.next;
    ptrOne.next = prevNode;
    prevNode = ptrOne;
    reverseRecursion(t1,t2, prevNode);
   }
   else{
    ptrTwo.next = ptrOne;
    ptrOne.next = prevNode;
    System.out.println("n Reverse Through Recursion");
    display(ptrTwo);
   }
 }
 else if(ptrOne!=null){
  System.out.println("n Reverse Through Recursion");
  display(ptrOne);
 }
}

 public void display(Node head){
  //
  Node currNode = head;
  while(currNode!=null){
   System.out.print("->" + currNode.data);
   currNode=currNode.next;
  }
 }
}
 public void reverseRecursion(Node ptrOne,Node ptrTwo, Node prevNode){
```

```
    public Node next;
    public Node(int data){
     this.data = data;
     this.next = null;
    }
   }
   class LinkedListT{
    public Node head;
    public LinkedListT(){
     head=null;
    }

    public void addAtBegin(int data){
     Node n = new Node(data);
     n.next = head;
     head = n;
    }
  public class ReverseLinkedList {
   public static void main (String[] args) throws java.lang.Exception
   {
    LinkedListT a = new LinkedListT();
    a.addAtBegin(5);
    a.addAtBegin(10);
    a.addAtBegin(15);
    a.addAtBegin(20);
    a.addAtBegin(25);
    a.addAtBegin(30);
  //  System.out.print("Original Link List 1 : ");
    a.display(a.head);
    a.reverseIterative(a.head);
    LinkedListT b = new LinkedListT();
    b.addAtBegin(31);
    b.addAtBegin(32);
    b.addAtBegin(33);
    b.addAtBegin(34);
    b.addAtBegin(35);
    b.addAtBegin(36);
    System.out.println("");
    System.out.println("____________________");
    System.out.print("Original Link List 2 : ");
    b.display(b.head);
    b.reverseRecursion(b.head,b.head.next,null);
    System.out.println("");
    //b.display(x);
   }
  }
  class Node{
   public int data;
   public Node next;
   public Node(int data){
    this.data = data;
    this.next = null;
   }
  }
 class LinkedListT{
  public Node head;
  public LinkedListT(){
```

- Create 3 nodes, currNode, PrevNode and nextNode.
- Initialize them as currNode = head; nextNode = **null**;prevNode = **null**;
- Now keep reversing the pointers one by one till currNode!=null.

```
while(currNode!=null){
  nextNode = currNode.next;
  currNode.next = prevNode;
  prevNode = currNode;
  currNode = nextNode;
}
```

Algorithm 2 : by using recursion

- Take 3 nodes as Node ptrOne,Node ptrTwo, Node prevNode
- Initialize them as ptrOne = head; ptrTwo=head.next, prevNode = null.
- Call reverseRecursion(head,head.next,null)
- Reverse the ptrOne and ptrTwo
- Make a recursive call for **reverseRecursion(ptrOne.next,ptrTwo.next,null)**

Full implementation:

```
public class ReverseLinkedList {
 public static void main (String[] args) throws java.lang.Exception
 {
  LinkedListT a = new LinkedListT();
  a.addAtBegin(5);
  a.addAtBegin(10);
  a.addAtBegin(15);
  a.addAtBegin(20);
  a.addAtBegin(25);
  a.addAtBegin(30);
//  System.out.print("Original Link List 1 : ");
  a.display(a.head);
  a.reverseIterative(a.head);
  LinkedListT b = new LinkedListT();
  b.addAtBegin(31);
  b.addAtBegin(32);
  b.addAtBegin(33);
  b.addAtBegin(34);
  b.addAtBegin(35);
  b.addAtBegin(36);
  System.out.println("");
  System.out.println("____________________");
  System.out.print("Original Link List 2 : ");
  b.display(b.head);
  b.reverseRecursion(b.head,b.head.next,null);
  System.out.println("");
  //b.display(x);
 }
}
class Node{
 public int data;
```

```
        while(currNode!=null){
            int data = currNode.data;
            if(ht.containsKey(data)){
                prevNode.next = currNode.next;
                temp= currNode;
                currNode = currNode.next;
                temp.next = null;
            }else{
                ht.put(data, 1);
                prevNode = currNode;
                currNode = currNode.next;
            }
        } return head;
    }
    public void display(Node head){
        Node n=head;
        while(n!=null){
            System.out.print("->" + n.data);
            n=n.next;
        }
    }
    public static void main(String args[]){
        Node n = new Node(2);
        n.next = new Node(2);
        n.next.next = new Node(2);
        n.next.next.next = new Node(3);
        n.next.next.next.next = new Node(4);
        n.next.next.next.next.next = new Node(4);
        n.next.next.next.next.next.next = new Node(2);
        System.out.print("Original List : ");
        RD rm = new RD();
        rm.display(n);
        System.out.print("n Updated List: ");
        Node x =rm.removeDup(n);
        rm.display(x);
    }
}
class Node{
    int data;
    Node next;
    public Node(int data){
        this.data = data;
        next = null;
    }
}
```

Question: Reverse a Linked List using Recursion. [Accenture]

The solution:

There are two ways to reverse a given linked list

Algorithm 1 : by using iteration

Full Implementation:

```
class LinkedList
{
    Node head;  // head of list

    /* Linked list Node*/
    class Node
    {
        int data;
        Node next;
        Node(int d) {data = d; next = null; }
    }

    void moveToFront()
    {
        /* If linked list is empty or it contains only
           one node then simply return. */
           if(head == null || head.next == null)
              return;

        /* Initialize second last and last pointers */
        Node secLast = null;
        Node last = head;

        /* After this loop secLast contains address of
           second last  node and last contains address of
           last node in Linked List */
        while (last.next != null)
        {
           secLast = last;
           last = last.next;
        }

        /* Set the next of second last as null */
        secLast.next = null;

        /* Set the next of last as head */
        last.next = head;

        /* Change head to point to last node. */
        head = last;
     }


    /* Utility functions */

    /* Inserts a new Node at front of the list. */
    public void push(int new_data)
    {
        /* 1 & 2: Allocate the Node &
                  Put in the data*/
        Node new_node = new Node(new_data);
```

```
        /* 3. Make next of new Node as head */
        new_node.next = head;

        /* 4. Move the head to point to new Node */
        head = new_node;
    }

    /* Function to print linked list */
    void printList()
    {
        Node temp = head;
        while(temp != null)
        {
            System.out.print(temp.data+" ");
            temp = temp.next;
        }
        System.out.println();
    }

    /* Drier program to test above functions */
    public static void main(String args[])
    {
        LinkedList llist = new LinkedList();
        /* Constructed Linked List is 1->2->3->4->5->null */
        llist.push(5);
        llist.push(4);
        llist.push(3);
        llist.push(2);
        llist.push(1);

        System.out.println("Linked List before moving last to front ");
        llist.printList();

        llist.moveToFront();

        System.out.println("Linked List after moving last to front ");
        llist.printList();
    }
}
```

OutPut:

```
Linked list before moving last to front
1 2 3 4 5
Linked list after removing last to front
5 1 2 3
```

Performance:

Time Complexity: O(n) where n is the number of nodes in the given Linked List.

Question: Reverse a singly Linked List in Java?

The solution:

In order to add a new element we will use following algorithm

1. If Linked list is empty then make the node as head and return it.
2. If value of the node to be inserted is smaller than value of head node then insert the node at start and make it head.
3. If value is larger then the head node then Iterate through the linked list and compare value of each node with the value that you want to insert
4. if value > currentNodeValue then add the value after the current node

Full Code implementation

```
// Java Program to insert in a sorted list
class LinkedList
{
    Node head;  // head of list

    /* Linked list Node*/
    class Node
    {
        int data;
        Node next;
        Node(int d) {data = d; next = null; }
    }

    /* function to insert a new_node in a list. */
    void sortedInsert(Node new_node)
    {
         Node current;

         /* Special case for head node */
         if (head == null || head.data >= new_node.data)
         {
            new_node.next = head;
            head = new_node;
         }
         else {

            /* Locate the node before point of insertion. */
            current = head;

            while (current.next != null &&
                   current.next.data < new_node.data)
                current = current.next;

            new_node.next = current.next;
            current.next = new_node;
         }
     }

                /*Utility functions*/
```

```
while(currNode!=null){
  System.out.print("->" + currNode.data);
  currNode=currNode.next;
 }
}
}
 public void reverseRecursion(Node ptrOne,Node ptrTwo, Node prevNode){
  if(ptrTwo!=null){
    if(ptrTwo.next!=null){
     Node t1 = ptrTwo;
     Node t2 = ptrTwo.next;
     ptrOne.next = prevNode;
     prevNode = ptrOne;
     reverseRecursion(t1,t2, prevNode);
     }
     else{
      ptrTwo.next = ptrOne;
      ptrOne.next = prevNode;
      System.out.println("n Reverse Through Recursion");
      display(ptrTwo);
     }
    }
     else if(ptrOne!=null){
      System.out.println("n Reverse Through Recursion");
      display(ptrOne);
     }
    }

     public void display(Node head){
      //
      Node currNode = head;
      while(currNode!=null){
       System.out.print("->" + currNode.data);
       currNode=currNode.next;
      }
      }
      }
```

Output:

```
->30->25->20->15->10->5
Reverse Through Iteration
->5->10->15->20->25->30
```

```
Original Link List 2 : ->36->35->34->33->32->31
Reverse Through Recursion
->31->32->33->34->35->36
```

Question: Insert nodes into a Linked list in a sorted fashion in Java. [Telephonic]

How will you add a new node in sorted manner in a linked list?

compare when it's greater than next node → decide with current node. If smaller then add here only. If greater than next node Just

```
    this.data = data;
    this.next = null;
  }
}
class LinkedListT{
 public Node head;
 public LinkedListT(){
  head=null;
 }

 public void addAtBegin(int data){
  Node n = new Node(data);
  n.next = head;
  head = n;
 }
 public void reverseIterative(Node head){
  Node currNode = head;
  Node nextNode = null;
  Node prevNode = null;

  while(currNode!=null){
   nextNode = currNode.next;
   currNode.next = prevNode;
   prevNode = currNode;
   currNode = nextNode;
  }
  head = prevNode;
  System.out.println("n Reverse Through Iteration");
  display(head);
 }

 public void reverseRecursion(Node ptrOne,Node ptrTwo, Node prevNode){
  if(ptrTwo!=null){
    if(ptrTwo.next!=null){
     Node t1 = ptrTwo;
     Node t2 = ptrTwo.next;
     ptrOne.next = prevNode;
     prevNode = ptrOne;
     reverseRecursion(t1,t2, prevNode);
    }
    else{
     ptrTwo.next = ptrOne;
     ptrOne.next = prevNode;
     System.out.println("n Reverse Through Recursion");
     display(ptrTwo);
    }
  }
  else if(ptrOne!=null){
   System.out.println("n Reverse Through Recursion");
   display(ptrOne);
  }
 }

 public void display(Node head){
  //
  Node currNode = head;
```

Ok be known

```
   b.reverseRecursion(b.head,b.head.next,null);
   System.out.println("");
   //b.display(x);
  }
 }
 class Node{
  public int data;
  public Node next;
  public Node(int data){
   this.data = data;
   this.next = null;
  }
 }
 class LinkedListT{
  public Node head;
  public LinkedListT(){
   head=null;
  }

  public void addAtBegin(int data){
   Node n = new Node(data);
   n.next = head;
   head = n;
  }
 public class ReverseLinkedList {
  public static void main (String[] args) throws java.lang.Exception
  {
   LinkedListT a = new LinkedListT();
   a.addAtBegin(5);
   a.addAtBegin(10);
   a.addAtBegin(15);
   a.addAtBegin(20);
   a.addAtBegin(25);
   a.addAtBegin(30);
 //  System.out.print("Original Link List 1 : ");
   a.display(a.head);
   a.reverseIterative(a.head);
   LinkedListT b = new LinkedListT();
   b.addAtBegin(31);
   b.addAtBegin(32);
   b.addAtBegin(33);
   b.addAtBegin(34);
   b.addAtBegin(35);
   b.addAtBegin(36);
   System.out.println("");
   System.out.println("____________________");
   System.out.print("Original Link List 2 : ");
   b.display(b.head);
   b.reverseRecursion(b.head,b.head.next,null);
   System.out.println("");
   //b.display(x);
  }
 }
 class Node{
  public int data;
  public Node next;
  public Node(int data){
```

The solution:

There are two ways to reverse a given linked list

Algorithm 1 : by using iteration

- Create 3 nodes, currNode, PrevNode and nextNode.
- Initialize them as currNode = head; nextNode = **null**;prevNode = **null**;
- Now keep reversing the pointers one by one till currNode!=null.

```
   while(currNode!=null){
     nextNode = currNode.next;
     currNode.next = prevNode;
     prevNode = currNode;
     currNode = nextNode;
}
```

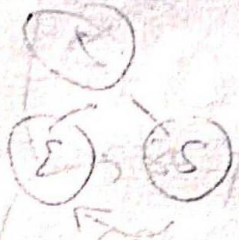
Algorithm 2 : by using recursion

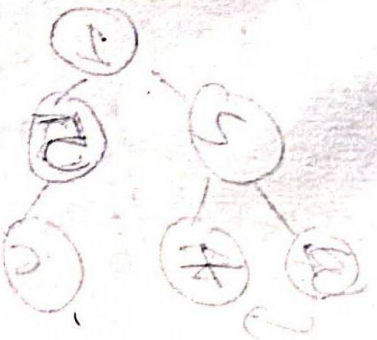
- Take 3 nodes as Node ptrOne,Node ptrTwo, Node prevNode
- Initialize them as ptrOne = head; ptrTwo=head.next, prevNode = null.
- Call reverseRecursion(head,head.next,null)
- Reverse the ptrOne and ptrTwo
- Make a recursive call for **reverseRecursion(ptrOne.next,ptrTwo.next,null)**

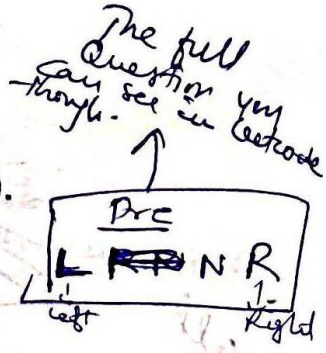
Full implementation:

```
public class ReverseLinkedList {
 public static void main (String[] args) throws java.lang.Exception
 {
  LinkedListT a = new LinkedListT();
  a.addAtBegin(5);
  a.addAtBegin(10);
  a.addAtBegin(15);
  a.addAtBegin(20);
  a.addAtBegin(25);
  a.addAtBegin(30);
 //  System.out.print("Original Link List 1 : ");
  a.display(a.head);
  a.reverseIterative(a.head);
  LinkedListT b = new LinkedListT();
  b.addAtBegin(31);
  b.addAtBegin(32);
  b.addAtBegin(33);
  b.addAtBegin(34);
  b.addAtBegin(35);
  b.addAtBegin(36);
  System.out.println("");
  System.out.println("____________________");
  System.out.print("Original Link List 2 : ");
  b.display(b.head);
```

```
/* Function to create a node */
Node newNode(int data)
{
    Node x = new Node(data);
    return x;
}

/* Function to print linked list */
void printList()
{
    Node temp = head;
    while (temp != null)
    {
        System.out.print(temp.data+" ");
        temp = temp.next;
    }
}

/* Drier function to test above methods */
public static void main(String args[])
{
    LinkedList llist = new LinkedList();
    Node new_node;
    new_node = llist.newNode(5);
    llist.sortedInsert(new_node);
    new_node = llist.newNode(10);
    llist.sortedInsert(new_node);
    new_node = llist.newNode(7);
    llist.sortedInsert(new_node);
    new_node = llist.newNode(3);
    llist.sortedInsert(new_node);
    new_node = llist.newNode(1);
    llist.sortedInsert(new_node);
    new_node = llist.newNode(9);
    llist.sortedInsert(new_node);
    System.out.println("Created Linked List");
    llist.printList();
}
}
```







Question: Flatten A Binary Tree to Linked List (In-place). [Google]

How will you flat a binary tree to linked list?

The Solution:

This is a tricky question often asked in google phone interview, here is the solution

For example we have a following binary tree

A3

Interview Set

Linked List.

Linkedlist Interview Questions PDF

About The Author:

References:

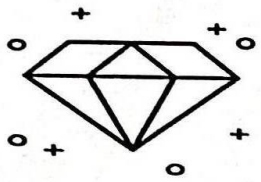
- https://en.wikipedia.org/wiki/Linked_list
- https://www.amazon.com/Cracking-Coding-Interview-Programming-Questions/dp/098478280X
- https://www.cs.cmu.edu/~adamchik/15-121/lectures/Linked%20Lists/linked%20lists.html
- https://www.quora.com/
- http://cslibrary.stanford.edu/103/LinkedListBasics.pdf

INTRODUCTION TO
LINKED LISTS

TOP 5 LINKED LIST QUESTIONS

1- Reverse a linked list
2- Cycle detection
3- Compare one with other DS
4- Remove duplicates
5- Advantage & Disadvantages

STRUCTURE



Java LinkedList is the Doubly-linked list implementation of the list interface

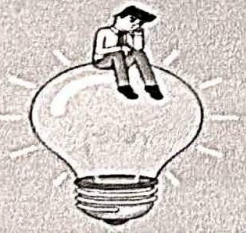
PERFORMANCE OF LINKED LISTS

	Single Linked list	Doubly Linked list
Access : a data	O(n) Slow	O(n) Slow
Search : a data	O(n) Slow	O(n) Slow
Insertion: a data	O(1) Fast	O(1) Fast
Deletion: a data	O(1) Fast	O(1) Fast

ADVANTAGES

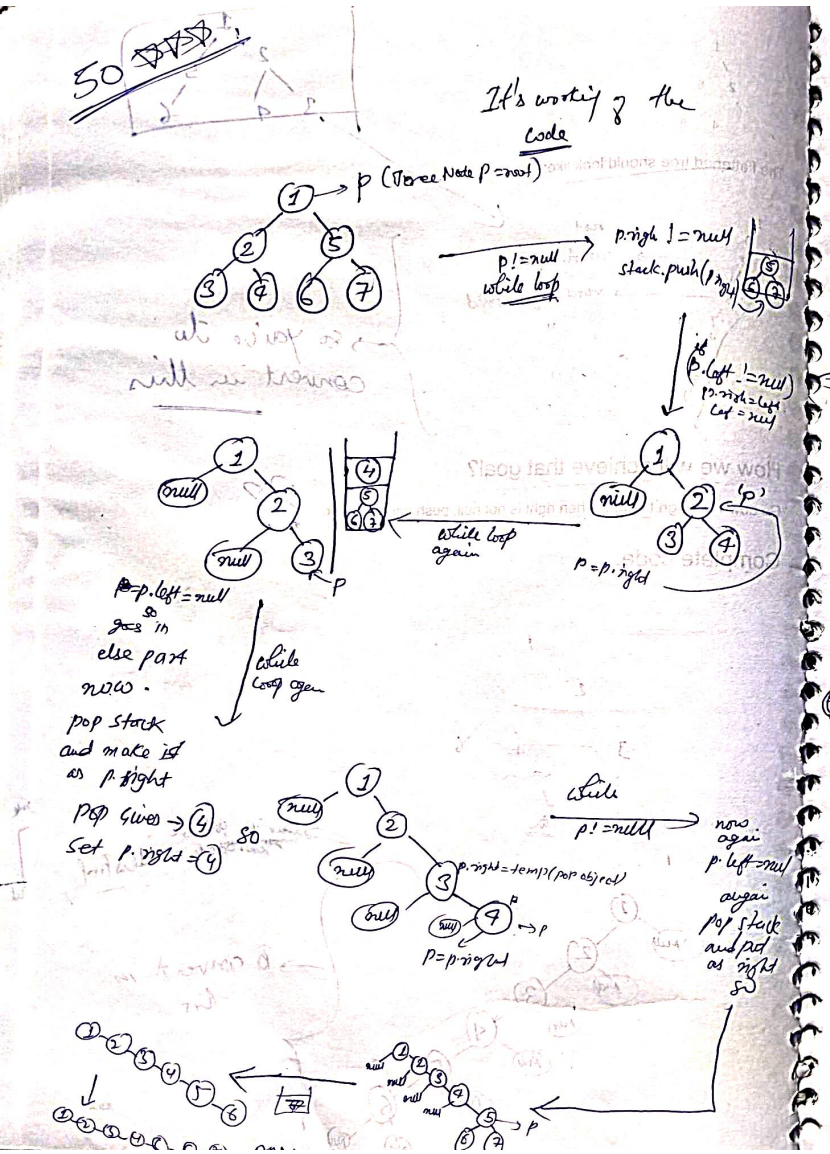
- Linked lists have constant-time insertion and deletion operation in any position
- Linked lists can continue to expand as long as there is space on the machine

DISADVANTAGES

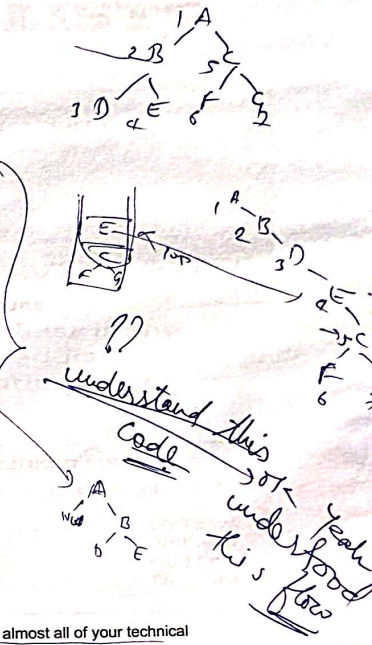


To access or edit an item in a linked list, you have to take O(n) time to walk from the head of the list to the ith item (unless of course you already have a pointer directly to that item). Arrays have constant time lookups and edits to the ith item.

http://www.codespaghetti.com




```
/**
 * Definition for binary tree
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    public void flatten(TreeNode root) {
        Stack stack = new Stack();
        TreeNode p = root;

        while(p != null || !stack.empty()){

            if(p.right != null){
                stack.push(p.right);
            }

            if(p.left != null){
                p.right = p.left;
                p.left = null;
            }else if(!stack.empty()){
                TreeNode temp = stack.pop();
                p.right=temp;
            }

            p = p.right;
        }
    }
}
```



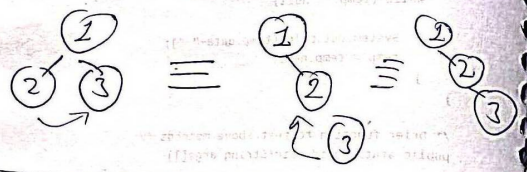
Keys to Interview Success:

You will be asked questions about the linked lists in almost all of your technical interviews. Google is particularly notorious in asking linked lists questions in their telephonic interviews.

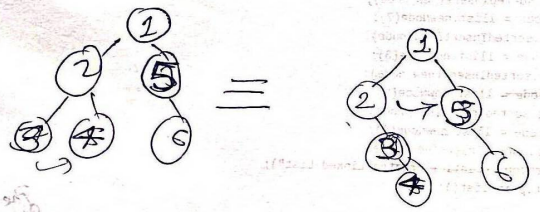
Part of this has to do is, with the underlying nature of the linked lists structure. It offers a great flexibility for the developer to modify the linked lists basic functionality based on your requirements.

Few examples are Queues and stack. Make sure you have gone through all the basic concepts and most important programs. Knowing linked lists thoroughly will definitely increase your chances of success in interviews.

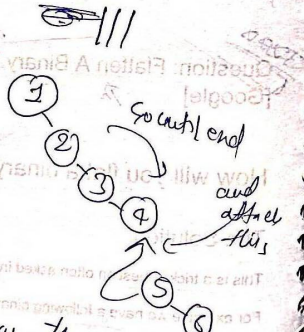
e.g

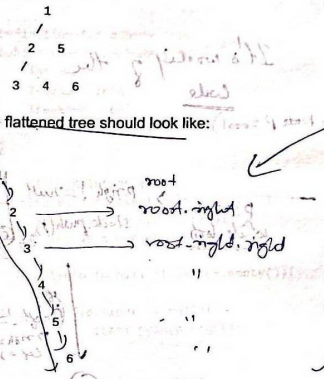


But for more than 3 nodes.

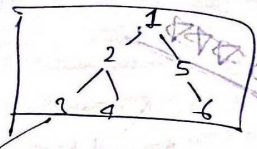


Call this function recursively





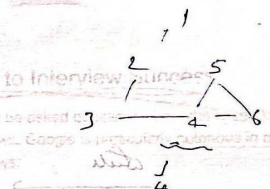
The flattened tree should look like:

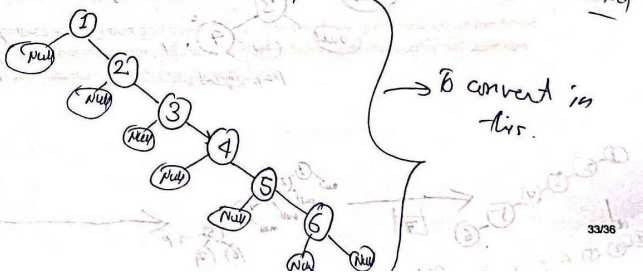


How we will achieve that goal?

Go down through the left, when right is not null, push right to stack.

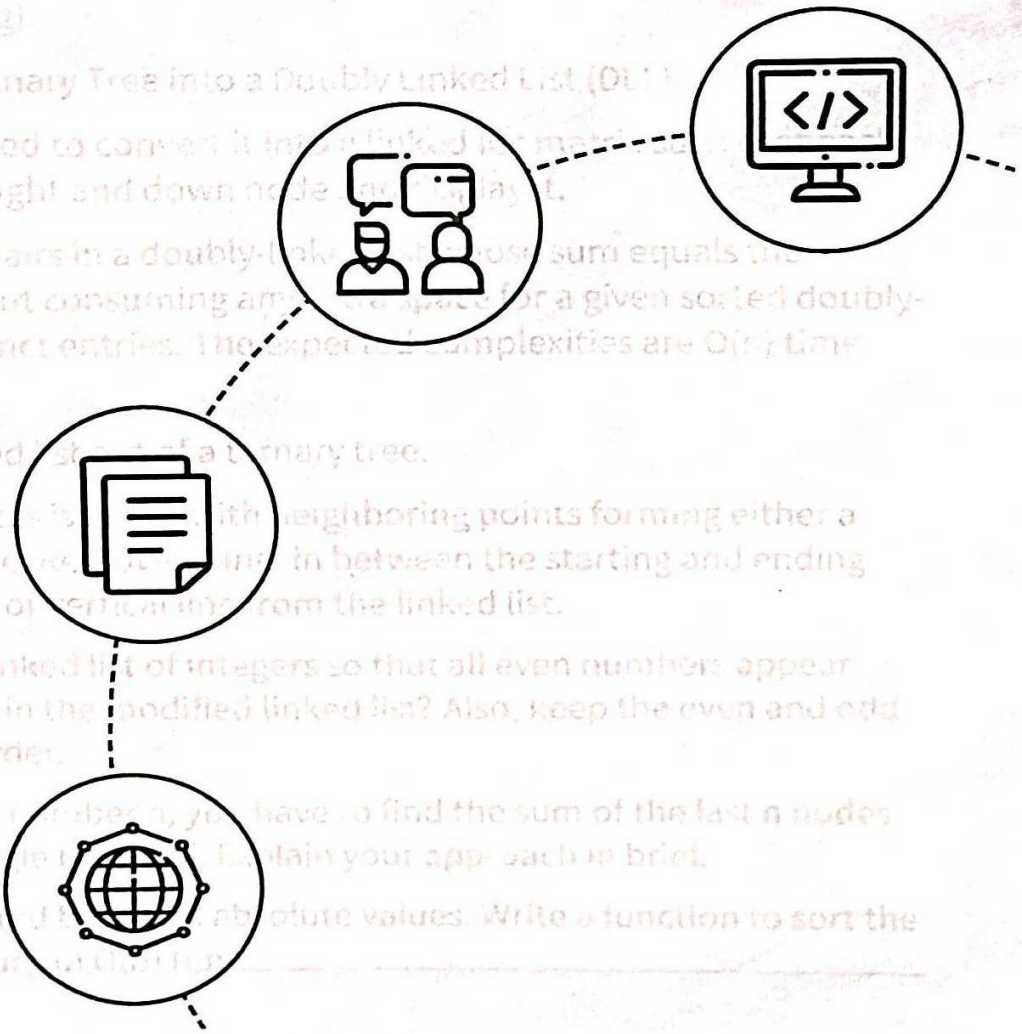
Complete code:





InterviewBit

Linked List Interview Questions



To view the live version of the page, click here.