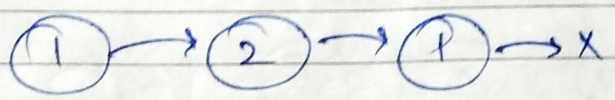
* imp

Page No.
Date

Q. Check for the palindrome in the linked list

Eg:



✓ Approach 1: we can create an array and copy the data of linked list to that array and check for palindrome as done prev.

Code:

```
Class Solution {

Private:
    bool checkPalindrome (vector<int> arr)
    {
        int n = arr.size();
        int s = 0;
        int e = n-1;

        while ( s<=e) {

        if ( arr[s] != arr[e]) {
                return false;

            }
            s++;
            e--;
        }
```

```
        return true;
    }
    public:
        bool isPalindrome(Node *head){
            vector<int> arr;
            Node *temp = head;
            while(temp != NULL){
                arr.push_back(temp->data);
                temp = temp->next;
            }
            return checkPalindrome;
        }
};
```

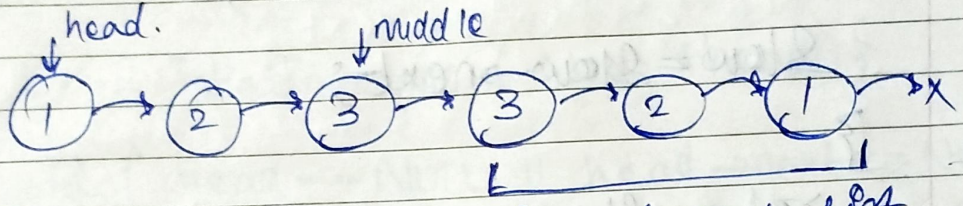
T.C = O(n)

S.C = O(n)

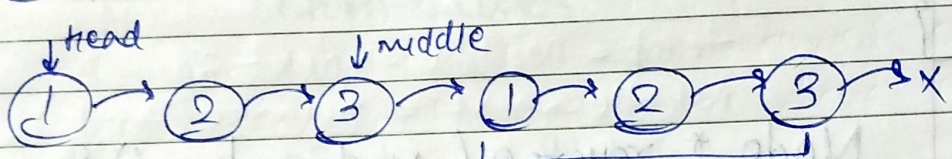
Page No.
Date

Approach 2 :

(I) Reach to the middle of the linked list

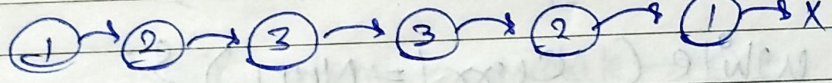


(II) Reverse the Remaining linked list



(III) Compare the linked list with other half

(IV) again Reverse to make no changes in LL



Code :-

```
Class Solution {
Private:
Node * getMid ( Node * head) {
        Node * slow = head;
        Node * fast = head -> next;

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

```
        fast = fast->next->next;
        slow = slow->next;
    }
    return slow;
}

Node * reverse(Node * head){
    Node *curr = head;
    Node * prev = NULL;
    Node * forward = NULL;
    while(curr != NULL){
        forward = curr->next;
        curr->next = prev;
        prev = curr;
        curr = forward;
    }
    return prev;
}
```

Public:

```
bool isPalindrome (Node *head){
if (head == NULL || head->next == NULL)
    {
        return false;
    }
// find Middle - Step 1
Node * middle = getMid(head);
// Reverse list after mid - Step 2
Node *temp = middle->next;
middle->next = reverse(temp);
// Step 3 - Compare the half's
Node * head1 = head;
Node *head2 = middle->next;
while (head2 != NULL){
if (head1->data != head2->data){
        return false;
    }
```

```
        // update Pointers
        head1 = head1->next;
        head2 = head2->next;

    }

        // Repeat Step 2 (optional) - Step 4

        temp = middle->next;
        middle->next = reverse(temp);

    return true;
    }
};
```

(*) T.C = O(n)

S.C = O(1)

Interviews usually.

Imp (***)

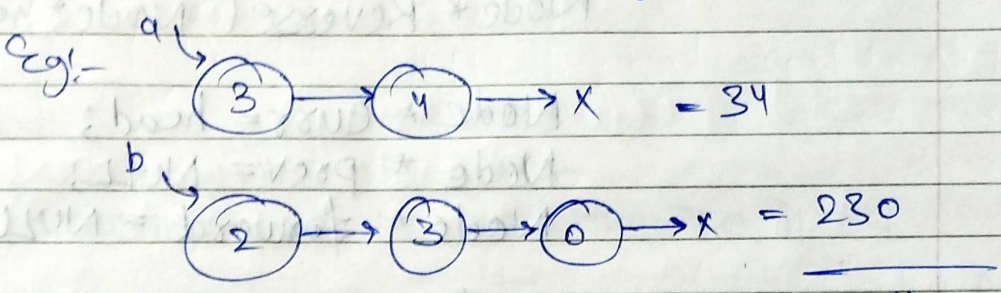
(CFG)

Page No.
Date

Q Add 2 numbers in the linked list.

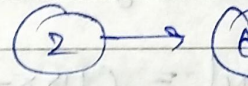
→ Add 2 no. represented by the L.L.
→ Add 1 to a no. rep by the L.L

Eg:-

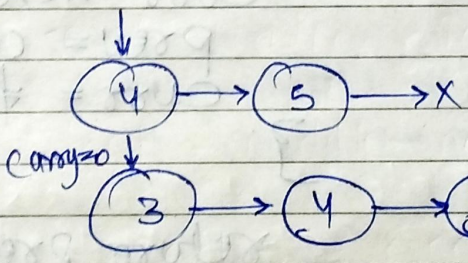


34
+230
264

o/p =



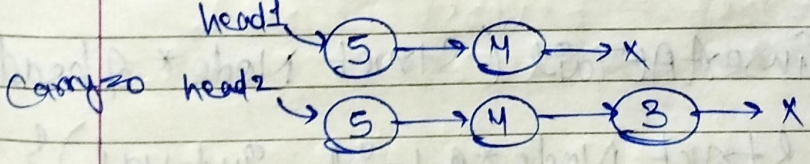
Approach :



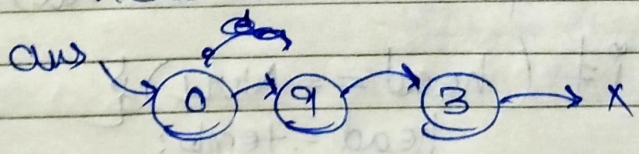
we always add like 45 + 345

345
+45
390

(I) Reverse the linked list



(II) Add and Place the carry if any to the next node.



(III) Reverse the linked list to get ans.

Code :-

```
class Solution {
    private:
    // method used to reverse the L.L
    Node * Reverse( Node *head){

        Node * curr= head;
        Node * prev= NULL;
        Node * forward = NULL;

        while( curr != NULL){

            forward = curr->next;
            curr->next = prev;
            prev= curr;
            curr = forward;
        }

        return prev;
    }
// method used to insert the node in ans
void insertAtTail ( struct Node * &head,
        struct Node *&tail , int val){
Node * temp = new Node (val);

    if ( head == NULL){
        head =temp;
        tail= temp;
        return; }
```

```
else{
        tail->next=temp;
        tail=temp;
    }
}
```

// method use to add the no in L.L

```
Struct Node * add( struct Node *first,
                   struct Node *second)
{   Node * ansHead=NULL;
{   Node* ansTail =NULL;
    int carry=0;
while ( first !=NULL || second !=NULL
        || carry !=0)
{
    int val1=0;
    if (first !=NULL)
        val1 = first->data;

    if (second !=NULL)
        val2 = second->data;

    int Sum = carry+val1+val2;

    int digit = Sum %10;
InsertAtTail (ansHead,ansTail,
              digit);
```

```
carry = sum/10;
if (first != NULL)
        first = first->next;

if (second != NULL)
        second = second->next;

}
return ansHead;
}
```

Public:

```
Struct Node* addTwoLists (struct Node* first,
        struct Node* second)
{
    // Step 1 - Reverse the L.L
    first = Reverse (first);
    second = Reverse (second);

    // Step 2 - add the 2 L.L
    Node *ans = add (first, second);

    // Step 3 - Reverse ans L.L
    ans = Reverse (ans);
```

return ans;

}

};

(*) T.C = O(M+N)

M → length of first L.L

N → length of second L.L

(*) S.C = O(Max(M,N))

Dry Run :-

first → (4) → (5) → X

second → (3) → (4) → (5) → X

(I) Reverse the L.L

first → (5) → (4) → X

second → (5) → (4) → (3) → X

(II) Add the L.L

first → (5) → (4) → X

int carry = 0

second → (5) → (4) → (3) → X

if (first != NULL)
then add.
if (second != NULL)
then add.

first → 5 → 4 → x

int carry = 0 → 1

second → 5 → 4 → 3 → x

Eg'

---

ans = 10 → 9 → 3 → x
Sum = 10

digit = 0
carry = 1

do this step until the first, second becomes NULL

(*)

ans = 0 → 9 → 3 → x

again Reverse the ans to get final ans

ans = 3 → 9 → 0 → x

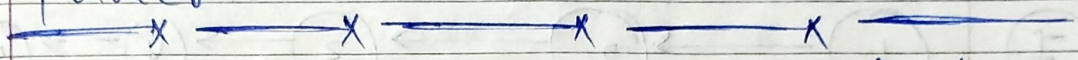
return ans;

→ GFG (Hard)

→ Imp (***) // important.

Page No.
Date

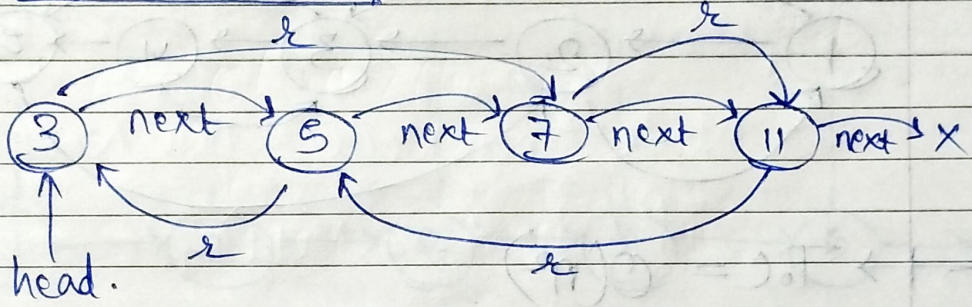
Q Clone a linked list with next and random pointer.



```
int data;
Node *next;
Node *random;
```

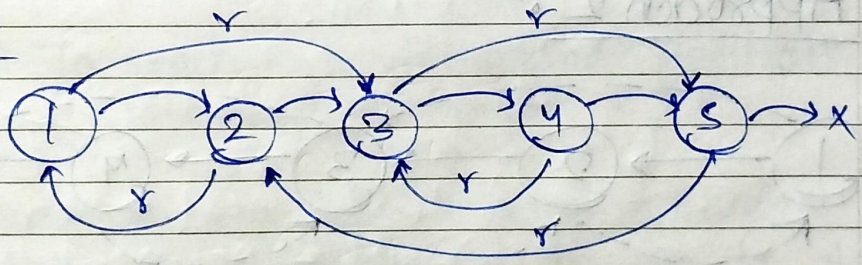
→ Structure of random pointing LL

Eg'



(*) Approach 1 :-

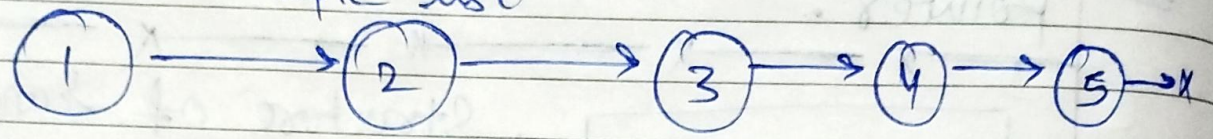
Eg'-



(I) Create a temp to traverse and create a cloneList (using orginal list next) O(n)

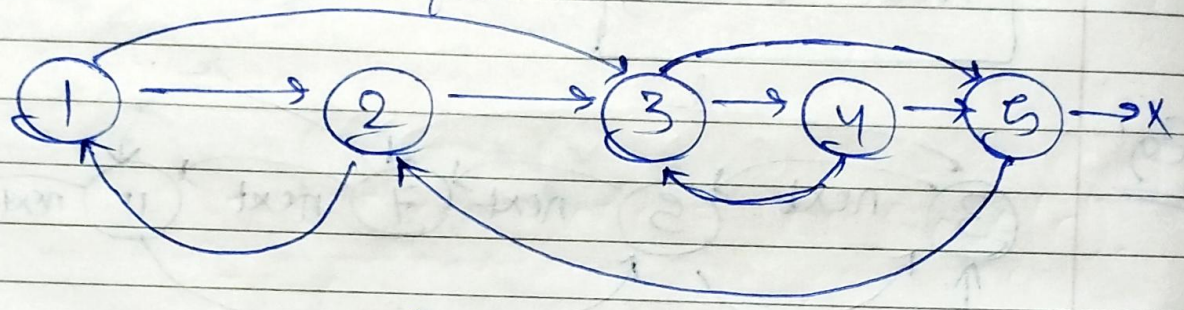
(II) Now check where the random pointer is pointing in clone list and place in the random pointer in clone list. O(n)

Create Simple list

(I) 

(II) Place random pointer

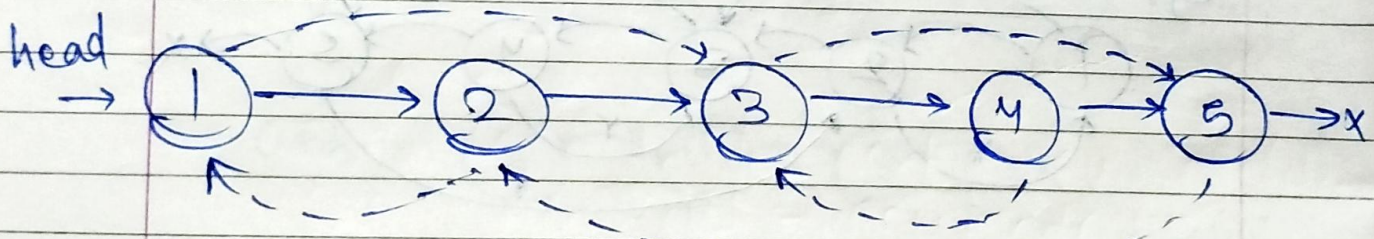
original



→ T.C = $O(n^2)$

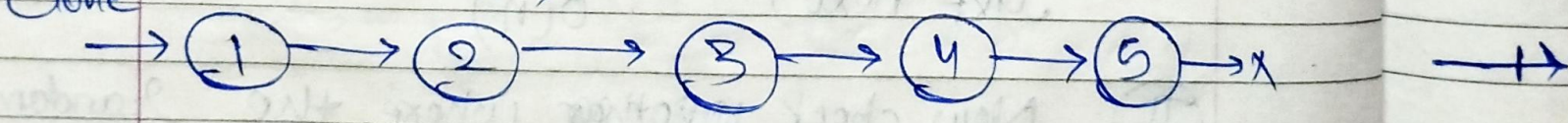
Clone

(*) Approach 2 :-

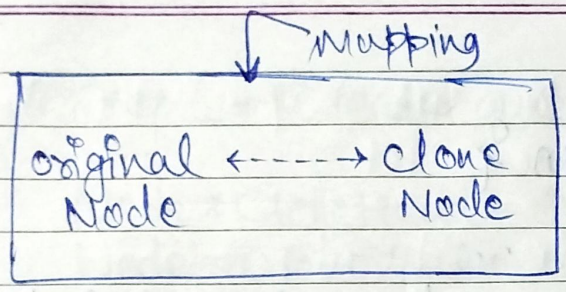


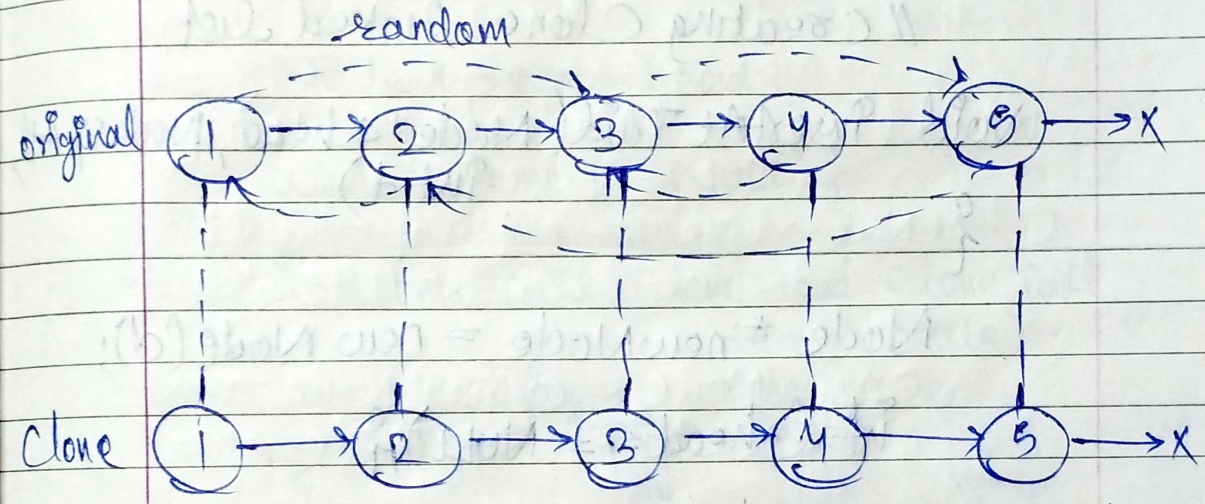
(I) create a clone list using the nextptr

→ O(n)



(II) Store the mapping using the map data structure.





CloneNode → random = ?

(III) To reach at destinated position.
(CloneNode → random = ?)

first go to originalNode → random and then use mapping to go to the required position for random.

→ CloneNode → random = mapping [originalNode → random]

T.C = O(n)
S.C = O(n)

Code :-

```
class Solution {
Private :
    // creating Clone linked list
void insertAtTail(Node *& head, Node*& tail,
                  int d)
  {
    Node * newNode = new Node(d);
    if (head == NULL) {
        head = newNode;
        tail = newNode;
        return;
      }
    else {
        tail->next = newNode(d);
        tail = newNode;
      }
  }
Public:
Node* CopyList (Node * head) {
```

// Step 1 - Create a clone list

```
Node *CloneHead = NULL;
Node *CloneTail = NULL;

Node * temp=head;
while( temp != NULL){
InsertAtTail(CloneHead, CloneTail,
                        temp-> data);
    temp= temp->next;
}
```

// Step-2 Create a map

```
unordered_map < Node*, Node* > mapping;

Node * originalNode= head;
Node * cloneNode = cloneHead;

while( originalNode != NULL){

Mapping[originalNode] = cloneNode;
    originalNode = originalNode->next;
    cloneNode = cloneNode ->next;

}
```

// Step 3 - place the random ptr in clone list

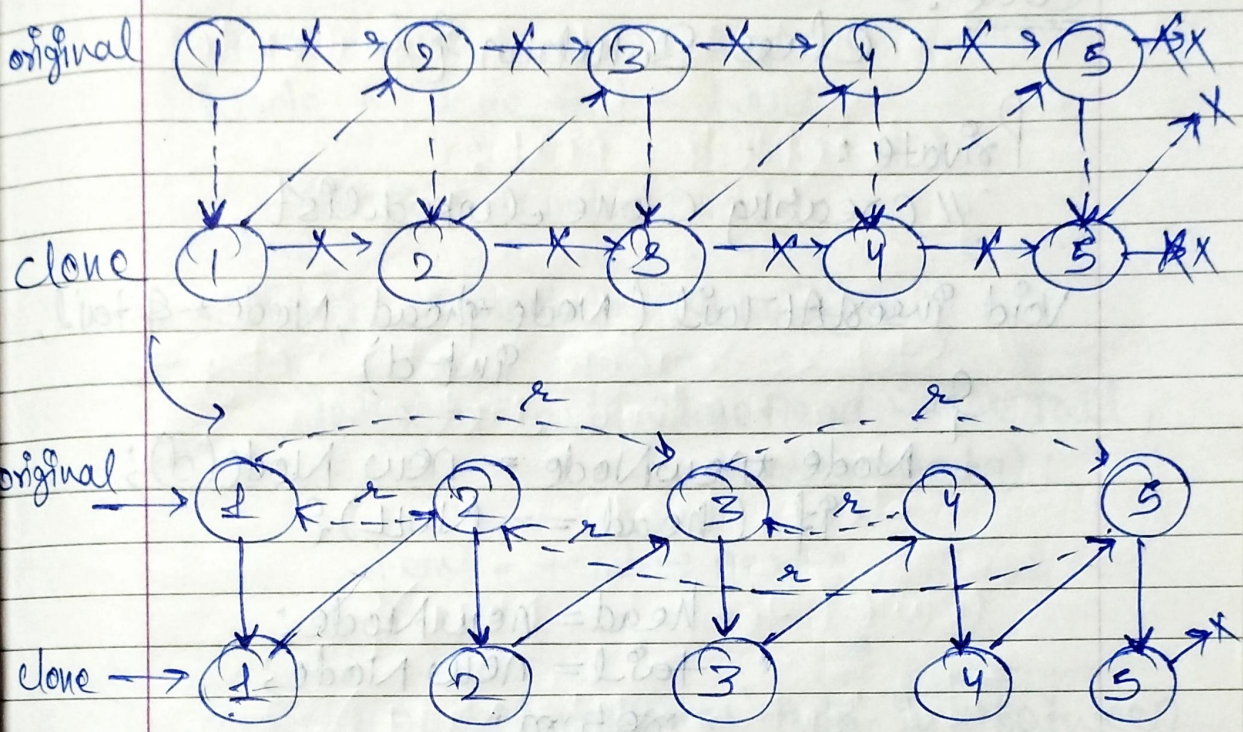
```
originalNode = head;
cloneNode = cloneHead;

while ( originalNode != NULL) {

cloneNode->random= mapping[originalNode->
                          random];

    originalNode = originalNode->next;
    cloneNode = cloneNode->next;
}

    return cloneHead;
}
};
```

T.C = O(n)
S.C = O(n)

(*) Approach 3 :- we need to reduce the space complexity
S.C O(n) → O(1)

(I) Create a clone list

(II) add the clone node inbetween the orginal list.



(III) The point the random ptr

```
temp -> next = temp -> random -> next;
```

All the random pointers are set.

(IV) Revert the changes done in step 2.

(V) Return the clone head;

* T.C = O(n)
  S.C = O(1)

Page No.
Date

Code :-

```
class Solution {
private:
    // creating clone linked list
void insertAtTail(Node *head, Node *&tail,
                  int d)
{
    Node *newNode = new Node(d);
    if (head == NULL){
        head = newNode;
        tail = newNode;
        return;
    }
    else {
        tail->next = newNode(d);
        tail = newNode;
    }
}
public:
    Node * copyList(Node *head)
{
    // step 1 : create a clone list
```

```
Node *cloneHead = NULL;
Node *cloneTail = NULL;

    Node *temp = head;

  while (temp != NULL){
    insertAtTail(cloneHead, cloneTail,
                 temp->data);

      temp = temp->next;
    }
// Step 2 : CloneNodes add in between
          the orginal list

  Node * originalNode = head;
  Node * cloneNode = cloneHead;

  while(originalNode != NULL &&
        cloneNode != NULL)
  {
      Node * next = originalNode->next;

      originalNode->next = cloneNode;

      originalNode = next;

      next = cloneNode->next;
```

```
    cloneNode->next = orginalNode;
    cloneNode = next;
}

// Step 3: random pointer copy to
            clone list
    temp = head;
while (temp != NULL){
if (temp->next != NULL){
    if (temp->random != NULL){
    temp->next->random = temp->random
                          ->next;
    }
    else {
    temp-> next = temp->random;
    }
}
    temp = temp->next->next;
}
```

// Step 4: Revert the changes done in Step 2

```
originalNode = head;
CloneNode = CloneHead;

while( originalNode != NULL && CloneNode != NULL)
  {
    originalNode->next = CloneNode->next;
    originalNode = originalNode->next;
    if( originalNode != NULL){
    CloneNode->next = originalNode->next;
    }
    CloneNode = CloneNode->next;
  }
```

// Step 5: Return ans

```
    return CloneHead;
  }
};
```

* T.C = O(n)
  S.C = O(1)