1. Linked List Cycle

Solution:

```
public class Solution {
        public boolean hasCycle(ListNode head) {
            ListNode slow = head;
            ListNode fast = head;
            while(slow!=null && fast!=null && fast.next!=null){
            slow = slow.next;
            fast = fast.next.next;
            if(slow==fast) return true;
            }
            return false;
        }
    }
```

TimeComplexity: o(n)

Space complexity: o(1)

2. Add two numbers 2
   Solution:

```
/**
* Definition for singly-linked list.
* public class ListNode {
* int val;
* ListNode next;
```

```
* ListNode() {}
* ListNode(int val) { this.val = val; }
* ListNode(int val, ListNode next) { this.val = val; this.next = next; }
* }
*/
class Solution {
public ListNode addTwoNumbers(ListNode l1, ListNode l2) {
ListNode list1 = new ListNode(l1.val);
ListNode curr = l1.next;
while(curr!=null){
ListNode newNode = new ListNode(curr.val);
newNode.next = list1;
list1 = newNode;
curr = curr.next;
}
curr = l2.next;
ListNode list2 = new ListNode(l2.val);
while(curr!=null){
ListNode newNode = new ListNode(curr.val);
newNode.next = list2;
list2 = newNode;
curr = curr.next;
```

```
}
ListNode ans = null;
int carry = 0;
while(list1!=null || list2!=null){
int sum = ((list1!=null)?list1.val:0)+(list2!=null?list2.val:0)+carry;
if(sum>=10){
carry = sum/10;
sum = sum%10;
}else{
carry = 0;
}
ListNode newNode = new ListNode(sum);
newNode.next = ans;
ans = newNode;
if(list1!=null) list1 = list1.next;
if(list2!=null) list2 = list2.next;
}
if(carry>0){
ListNode newNode = new ListNode(carry);
newNode.next = ans;
ans = newNode;
}
```

```
return ans;

}

public void print(ListNode head){

ListNode curr = head;

while(curr!=null){

System.out.print(curr.val+"->");

curr = curr.next;

}

System.out.println();

}

}
```

TimeComplexity: o(n)
space Complexity: o(n)

3. Merge two sorted lists
   Solution:

```
/**

* Definition for singly-linked list.

* public class ListNode {

* int val;

* ListNode next;

* ListNode() {}

* ListNode(int val) { this.val = val; }

* ListNode(int val, ListNode next) { this.val = val; this.next = next; }
```

```
* }
*/
class Solution {
public ListNode mergeTwoLists(ListNode list1, ListNode list2) {
ListNode ans = new ListNode(0);
ListNode dummy = ans;
while(list1!=null && list2!=null){
if(list1.val<list2.val){
ans.next = new ListNode(list1.val);
list1 = list1.next;
}else{
ans.next = new ListNode(list2.val);
list2 = list2.next;
}
ans = ans.next;
}
while(list1!=null){
ans.next = new ListNode(list1.val);
list1 = list1.next;
ans = ans.next;
}
while(list2!=null){
```

```
ans.next = new ListNode(list2.val);

list2 = list2.next;

ans = ans.next;

}

return dummy.next;

}

}
```

Time Complexity: o(n)
space Complexity: o(n)

4. Copy LinkedList with random pointer
   Solution:

```
/*
// Definition for a Node.
class Node {
int val;
Node next;
Node random;

public Node(int val) {
this.val = val;
this.next = null;
this.random = null;
}
}
*/

class Solution {
public Node copyRandomList(Node head) {
HashMap<Node,Node> hm = new HashMap<>();
Node curr = head;
while(curr!=null){
hm.put(curr,new Node(curr.val));
curr = curr.next;
}
curr = head;
while(curr!=null){
```

```
Node node = hm.get(curr);
node.next = hm.get(curr.next);
node.random = hm.get(curr.random);
curr = curr.next;
}
return hm.get(head);
}
}
```

Time Complexity: o(n)

Space Complexity: o(n)

5. Revese LinkedList 2
   Solution:

```
/**
* Definition for singly-linked list.
* public class ListNode {
* int val;
* ListNode next;
* ListNode() {}
* ListNode(int val) { this.val = val; }
* ListNode(int val, ListNode next) { this.val = val; this.next = next; }
* }
*/
class Solution {
public ListNode reverseBetween(ListNode head, int left, int right) {
ListNode first = new ListNode(-1);
ListNode dummy = first;
ListNode curr = head;
int count = 1;
while(curr!=null && count<left){
first.next = new ListNode(curr.val);
first = first.next;
curr = curr.next;
count++;
}
while(curr!=null && count<=right){
ListNode newNode = new ListNode(curr.val);
newNode.next = first.next;
first.next = newNode;
curr = curr.next;
count++;
}
while(first.next!=null){
first = first.next;
}
while(curr!=null){
first.next = new ListNode(curr.val);
```

```
first = first.next;
curr = curr.next;
}
return dummy.next;
}
}
```

Time complexity: o(n)

Space complexity: o(n)

6. Reverse nodes in k-groups
   Solution:

```
/**
* Definition for singly-linked list.
* public class ListNode {
* int val;
* ListNode next;
* ListNode() {}
* ListNode(int val) { this.val = val; }
* ListNode(int val, ListNode next) { this.val = val; this.next = next; }
* }
*/
class Solution {
public ListNode reverseKGroup(ListNode head, int k) {
Stack<ListNode> sta = new Stack<>();
int n = 0;
ListNode curr = head;
ListNode ans = new ListNode(0);
ListNode dummy = ans;

while(curr!=null){
curr = curr.next;
n++;
}
curr = head;
while(n>=k){
int count = 0;
while(count<k){
sta.add(curr);
count++;
curr = curr.next;
}
while(!sta.isEmpty()){
ans.next = sta.pop();
ans = ans.next;
```

```
}
n -= k;
}
ans.next = curr;
return dummy.next;
}
}
```

Time Complexity: o(n)

Space Complexity: o(n)

7. Remove nth node from the end of the list
   Solution:

```
/**
* Definition for singly-linked list.
* public class ListNode {
* int val;
* ListNode next;
* ListNode() {}
* ListNode(int val) { this.val = val; }
* ListNode(int val, ListNode next) { this.val = val; this.next = next; }
* }
*/
class Solution {
public ListNode removeNthFromEnd(ListNode head, int n) {
int total = 0;
HashMap<Integer, ListNode> hm = new HashMap<>();
ListNode curr = head;
while(curr!=null){
total++;
hm.put(total,curr);
curr = curr.next;
}
int fromLast = total-n;
if(fromLast==0) return head.next;
ListNode node = hm.getOrDefault(fromLast,null);
if(node!=null){
node.next = node.next.next;
}
return head;
}
}
```

Time Complexity: o(n)

Space Complexity: o(n)

8. Remove duplicates from the sorted List 2
   Solution:

```
/**
* Definition for singly-linked list.
* public class ListNode {
* int val;
* ListNode next;
* ListNode() {}
* ListNode(int val) { this.val = val; }
* ListNode(int val, ListNode next) { this.val = val; this.next = next; }
* }
*/
class Solution {
public ListNode deleteDuplicates(ListNode head) {
ListNode ans = new ListNode(0);
ListNode dummy = ans;
Set<Integer> set = new HashSet<>();
ListNode prev = ans;
ListNode curr = head;
while(curr!=null){
if(set.add(curr.val)){
ans.next = new ListNode(curr.val);
prev = ans;
ans = ans.next;
}else{
if(ans.val==curr.val){
prev.next = null;
ans = prev;
}
}
curr = curr.next;
}
return dummy.next;
}
}
```

Time Complexity: o(n)

Space complexity: o(n)

9. Rotate List
   Solution:

```
/**
```

```
* Definition for singly-linked list.
* public class ListNode {
* int val;
* ListNode next;
* ListNode() {}
* ListNode(int val) { this.val = val; }
* ListNode(int val, ListNode next) { this.val = val; this.next = next; }
* }
*/
class Solution {
public ListNode rotateRight(ListNode head, int k) {
if(head==null || k==0 ) return head;
HashMap<Integer,ListNode> hm = new HashMap<>();
int count = 0;
ListNode curr = head;
while(curr!=null){
count++;
hm.put(count,curr);
curr = curr.next;
}
if(k>count) k %= count;
int need = count-k;
if(k==0) return head;
ListNode node = hm.getOrDefault(need,null);
if(node==null) return head;
node.next = null;
hm.get(count).next = head;
head = hm.get(need+1);
return head;
}
}
```

Time complexity: o(n)

Space complexity: o(n)

10. Partition List

    Solution:

```
/**
* Definition for singly-linked list.
* public class ListNode {
* int val;
* ListNode next;
* ListNode() {}
* ListNode(int val) { this.val = val; }
* ListNode(int val, ListNode next) { this.val = val; this.next = next; }
```

```
* }
*/
class Solution {
public ListNode partition(ListNode head, int x) {
ListNode ans = new ListNode(0);
ListNode dummy = ans;
ListNode prev = ans;
ListNode temp = null;
ListNode next = prev.next;
ListNode curr = head;
while(curr!=null){
if(curr.val<x){
ListNode newNode = new ListNode(curr.val);
if(temp!=null){
newNode.next = temp;
}
prev.next = newNode;
prev = prev.next;
}else{
boolean found = false;
if(next==null){
found = true;
next = prev;
}
next.next = new ListNode(curr.val);
next = next.next;
if(found) temp = next;
}
curr = curr.next;
}
return dummy.next;
}
}
```

Time Complexity; o(n)
Space Complexity: o(n)