Assignment 2 – Linked List

Q1: Swap by Pairs

class Solution {

public ListNode swapPairs(ListNode head) {

if (head == null || head.next == null) return head;

ListNode first = head;

ListNode second = first.next;

first.next = swapPairs(second.next);

second.next = first;

return second;

}

}

Q2: Delete a Node

class Solution {

public void deleteNode(ListNode node) {

if (node == null) return;

node.val = node.next.val;

node.next = node.next.next;

}

}

Q3: Even Odd Linked List

class Solution {

public ListNode oddEvenList(ListNode head) {

if (head == null || head.next == null || head.next.next == null) return head;

ListNode odd = head;

ListNode evenHead = new ListNode(0, head.next);

ListNode even = evenHead.next;

while (odd != null && even != null && even.next != null) {

odd.next = even.next;

odd = odd.next;

even.next = odd.next;

even = even.next;

}

odd.next = evenHead.next;

return head;

}

}

Q4: Split Linked List in Pairs

class Solution {

public ListNode[] splitListToParts(ListNode root, int k) {

int length = countList(root);

int subLength = length / k;

ListNode[] result = new ListNode[k];

int left = length % k;

ListNode temp = root;

int groups = k;

if (k > length) groups = length;

for (int i = 0; i < groups; i++) {

ListNode end = temp;

int sub = subLength;

if (left > 0) {

sub = subLength + 1;

left--;

}

for (int j = 0; j < sub - 1; j++) {

if (end.next == null) break;

end = end.next;

}

result[i] = temp;

if (end.next != null) temp = end.next;

end.next = null;

}

return result;

}

private int countList(ListNode root) {

int num = 0;

ListNode temp = root;

while (temp != null) {

num++;

temp = temp.next;

}

return num;

}

}

Q5: Inserted in Circular Linked List

class Solution {

public Node insert(Node head, int insertVal) {

Node add = new Node(insertVal);

Node front = head;

if (front == null) {

add.next = add;

return add;

}

Node second = front.next;

if (second == front) {

head.next = add;

add.next = head;

return head;

}

while (second != head) {

if (front.val <= insertVal && second.val >= insertVal) break;

if (front.val > second.val && (front.val <= insertVal || second.val >= insertVal)) break;

front = second;

second = second.next;

}

add.next = second;

front.next = add;

return head;

}

}