Linked List Solutions

Solution 1:

Time Complexity : o(m*n)
Space Complexity: o(1)

```
class Solution {

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

    public Node getIntersectionNode(Node head1, Node head2)
    {
        while (head2 != null) {
            Node temp = head1;
            while (temp != null) {
                if (temp == head2) {
                    return head2;
                }
                temp = temp.next;
            }
            head2 = head2.next;
        }
        return null;
    }

    public static void main(String[] args){
        Solution list = new Solution();

        Node head1, head2;
        head1 = new Node(10);
        head2 = new Node(3);
```

```
                Node newNode = new Node(6);
                head2.next = newNode;

                newNode = new Node(9);
                head2.next.next = newNode;

                newNode = new Node(15);
                head1.next = newNode;
                head2.next.next.next = newNode;

                newNode = new Node(30);
                head1.next.next = newNode;

                head1.next.next.next = null;

                Node intersectionPoint
                        = list.getIntersectionNode(head1, head2);

                if (intersectionPoint == null) {
                        System.out.print(" No Intersection Point \n");
                }
                else {
                        System.out.print("Intersection Point: "
                                                        + intersectionPoint.data);
                }
        }
}
```

Solution 2:

Time Complexity : o(n)
Space Complexity: o(1)

```
import java.util.*;
class Solution{

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

```
};

static Node push( Node head_ref, int new_data){
        Node new_node = new Node();
        new_node.data = new_data;
        new_node.next = (head_ref);
        (head_ref) = new_node;
        return head_ref;
}

static void printList( Node head){
        Node temp = head;
        while (temp != null){
                System.out.printf("%d ", temp.data);
                temp = temp.next;
        }
        System.out.printf("\n");
}

static void skipMdeleteN( Node head, int M, int N){
        Node curr = head, t;
        int count;

        while (curr!=null){
                for (count = 1; count < M && curr != null; count++)
                        curr = curr.next;

                if (curr == null)
                        return;
                t = curr.next;
                for (count = 1; count <= N && t != null; count++){
                        Node temp = t;
                        t = t.next;
                }

                curr.next = t;
                curr = t;
        }
}
```

```
public static void main(String args[]){
        Node head = null;
        int M=2, N=3;
        head=push(head, 10);
        head=push(head, 9);
        head=push(head, 8);
        head=push(head, 7);
        head=push(head, 6);
        head=push(head, 5);
        head=push(head, 4);
        head=push(head, 3);
        head=push(head, 2);
        head=push(head, 1);

        System.out.printf("M = %d, N = %d \n" +
                                        "Linked list we have is :\n", M, N);
        printList(head);

        skipMdeleteN(head, M, N);

        System.out.printf("\nLinked list on deletion is :\n");
        printList(head);
}
}
```

Solution 3 :

Time Complexity : o(n)
Space Complexity: o(1)

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

```
}

class Solution {
    Node head;

    public void swapNodes(int x, int y){
        if (x == y)
            return;

        Node prevX = null, currX = head;
        while (currX != null && currX.data != x) {
            prevX = currX;
            currX = currX.next;
        }

        Node prevY = null, currY = head;
        while (currY != null && currY.data != y) {
            prevY = currY;
            currY = currY.next;
        }

        if (currX == null || currY == null)
            return;

        if (prevX != null)
            prevX.next = currY;
        else
            head = currY;

        if (prevY != null)
            prevY.next = currX;
        else
            head = currX;

        Node temp = currX.next;
        currX.next = currY.next;
        currY.next = temp;
    }

    public void push(int new_data){
```

```
        Node new_Node = new Node(new_data);
        new_Node.next = head;
        head = new_Node;
    }

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

    public static void main(String[] args){
        Solution llist = new Solution();

        llist.push(7);
        llist.push(6);
        llist.push(5);
        llist.push(4);
        llist.push(3);
        llist.push(2);
        llist.push(1);

        System.out.print(
            "\n Linked list before ");
        llist.printList();

        llist.swapNodes(4, 3);

        System.out.print(
            "\n Linked list after ");
        llist.printList();
    }
}
```

Solution 4 :

Time Complexity : o(n)
Space Complexity: o(1)

```
class Solution{
    Node head;
    class Node{
        int data;
        Node next;
        Node(int d){
            data = d;
            next = null;
        }
    }

    void segregateEvenOdd(){
        Node end = head;
        Node prev = null;
        Node curr = head;

        while (end.next != null)
            end = end.next;

        Node new_end = end;

        while (curr.data %2 !=0 && curr != end){
            new_end.next = curr;
            curr = curr.next;
            new_end.next.next = null;
            new_end = new_end.next;
        }

        if (curr.data %2 ==0){
            head = curr;
            while (curr != end){
                if (curr.data % 2 == 0){
                    prev = curr;
                    curr = curr.next;
                }
                else{
                    prev.next = curr.next;
                    curr.next = null;
                    new_end.next = curr;
```

```
                    new_end = curr;
                    curr = prev.next;
                }
            }
        }
        else prev = curr;
        if (new_end != end && end.data %2 != 0){
            prev.next = end.next;
            end.next = null;
            new_end.next = end;
        }
    }
    void push(int new_data){
        Node new_node = new Node(new_data);
        new_node.next = head;
        head = new_node;
    }

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

    public static void main(String args[]){
        Solution llist = new Solution();
        llist.push(11);
        llist.push(10);
        llist.push(8);
        llist.push(6);
        llist.push(4);
        llist.push(2);
        llist.push(0);
        System.out.println("Linked List");
        llist.printList();

        llist.segregateEvenOdd();
```

```
            System.out.println("updated Linked List");
            llist.printList();
        }
}
```

Solution 5 :

Time Complexity : o(n*logk)
Space Complexity: o(n)

```
public class Solution {
    public static Node SortedMerge(Node a, Node b){
        Node result = null;
        if (a == null)
            return b;
        else if (b == null)
            return a;
        if (a.data <= b.data) {
            result = a;
            result.next = SortedMerge(a.next, b);
        }
        else {
            result = b;
            result.next = SortedMerge(a, b.next);
        }

        return result;
    }

    public static Node mergeKLists(Node arr[], int last)
    {
        while (last != 0) {
            int i = 0, j = last;
            while (i < j) {
                arr[i] = SortedMerge(arr[i], arr[j]);
                i++;
                j--;
                if (i >= j)
                    last = j;
            }
```

```
            }

            return arr[0];
        }
        public static void printList(Node node){
            while (node != null) {
                System.out.print(node.data + " ");
                node = node.next;
            }
        }

        public static void main(String args[]){
            int k = 3;
            int n = 4;
            Node arr[] = new Node[k];

            arr[0] = new Node(1);
            arr[0].next = new Node(3);
            arr[0].next.next = new Node(5);
            arr[0].next.next.next = new Node(7);

            arr[1] = new Node(2);
            arr[1].next = new Node(4);
            arr[1].next.next = new Node(6);
            arr[1].next.next.next = new Node(8);

            arr[2] = new Node(0);
            arr[2].next = new Node(9);
            arr[2].next.next = new Node(10);
            arr[2].next.next.next = new Node(11);

            Node head = mergeKLists(arr, k - 1);
            printList(head);
        }
}

class Node {
        int data;
        Node next;
        Node(int data){
```

```
            this.data = data;
        }
}
```