import java.util.\*;

class Node {

int data;

Node next;

Node(int d)

{

data = d;

next = null;

}

}

class MergeLists {

Node head;

public void addToTheLast(Node node)

{

if (head == null) {

head = node;

}

else {

Node temp = head;

while (temp.next != null)

temp = temp.next;

temp.next = node;

}

}

/\* Method to print linked list \*/

void printList()

{

Node temp = head;

while (temp != null) {

System.out.print(temp.data + " ");

temp = temp.next;

}

System.out.println();

}

// Driver Code

public static void main(String args[])

{

/\* Let us create two sorted linked

lists to test the methods

Created lists:

llist1: 5->10->15,

llist2: 2->3->20

\*/

MergeLists llist1 = new MergeLists();

MergeLists llist2 = new MergeLists();

// Node head1 = new Node(5);

llist1.addToTheLast(new Node(5));

llist1.addToTheLast(new Node(10));

llist1.addToTheLast(new Node(15));

// Node head2 = new Node(2);

llist2.addToTheLast(new Node(2));

llist2.addToTheLast(new Node(3));

llist2.addToTheLast(new Node(20));

llist1.head = new Merge().sortedMerge(llist1.head,

llist2.head);

System.out.println("Merged Linked List is:");

llist1.printList();

}

}

class Merge {

Node sortedMerge(Node headA, Node headB)

{

Node dummyNode = new Node(0);

Node tail = dummyNode;

while (true) {

if (headA == null) {

tail.next = headB;

break;

}

if (headB == null) {

tail.next = headA;

break;

}

if (headA.data <= headB.data) {

tail.next = headA;

headA = headA.next;

}

else {

tail.next = headB;

headB = headB.next;

}

/\* Advance the tail \*/

tail = tail.next;

}

return dummyNode.next;

}

}

2.

class Solution {

public int[] searchRange(int[] nums, int target) {

if(nums == null) return new int[]{-1, -1};

int len = nums.length;

int index = binarySearch(0, len - 1, nums, target);

if(-1 == index) return new int[]{-1, -1};

else{

int high = index;

while(++high < len && nums[high] == target){};

int low = index;

while(--low >= 0 && nums[low] == target){}

return new int[]{low + 1, high - 1};

}

}

private static int binarySearch(int low, int high, int[] nums, int target){

if(low > high)

return -1;

int mid = low + (high - low) / 2;

int midVal = nums[mid];

if(midVal == target) return mid;

if(midVal > target)

return binarySearch(low, mid - 1, nums, target);

else

return binarySearch(mid + 1, high, nums, target);

}

}