// Sorting linkedList

**public** **class** sortLinkedList

{

node head = **null**;

**static** **int** *n*;

// node a,b;

**public** **static** **class** node

{

**int** val;

node next;

**public** node(**int** val)

{

**this**.val = val;

}

}

node sortedMerge(node a, node b)

{

node result = **null**;

/\* Base cases \*/

**if** (a == **null**)

**return** b;

**if** (b == **null**)

**return** a;

/\* Pick either a or b, and recur \*/

**if** (a.val <= b.val)

{

result = a;

result.next = sortedMerge(a.next, b);

}

**else**

{

result = b;

result.next = sortedMerge(a, b.next);

}

**return** result;

}

node mergeSort(node h)

{

// Base case : if head is null

**if** (h == **null** || h.next == **null**)

{

**return** h;

}

// get the middle of the list

node middle = getMiddle(h);

node nextofmiddle = middle.next;

// set the next of middle node to null

middle.next = **null**;

// Apply mergeSort on left list

node left = mergeSort(h);

// Apply mergeSort on right list

node right = mergeSort(nextofmiddle);

// Merge the left and right lists

node sortedlist = sortedMerge(left, right);

**return** sortedlist;

}

// Utility function to get the middle of the linked list

node getMiddle(node h)

{

//Base case

**if** (h == **null**)

**return** h;

node fastptr = h.next;

node slowptr = h;

// Move fastptr by two and slow ptr by one

// Finally slowptr will point to middle node

**while** (fastptr != **null**)

{

fastptr = fastptr.next;

**if**(fastptr!=**null**)

{

slowptr = slowptr.next;

fastptr=fastptr.next;

}

}

**return** slowptr;

}

**private** **static** sortLinkedList removeRepeatedElement(node h) {

// **TODO** Auto-generated method stub

node start = h;

node ele = h.next;

sortLinkedList linkedlist2 = **new** sortLinkedList();

**while**(ele!=**null**)

{

**if**(start.val!=ele.val)

{

linkedlist2.push(**new** node(start.val));

}

start = ele;

ele = ele.next;

}

linkedlist2.push(**new** node(start.val));

**return** linkedlist2;

}

**public** **void** push(node node) {

**if** (head == **null**) {

head = node;

} **else** {

node temp = head;

**while** (temp.next != **null**)

temp = temp.next;

temp.next = node;

}

}

// Utility function to print the linked list

**void** printList(node headref)

{

**while** (headref != **null**)

{

System.***out***.print(headref.val + " ");

headref = headref.next;

}

}

**public** **static** **void** main(String[] args)

{

sortLinkedList linkedlist = **new** sortLinkedList();

sortLinkedList linkedlist2 = **new** sortLinkedList();;

**int**[] arr = {5,1,7,2,1,3,9,3,10,1,3,3,3,3,10,15};

**for** (**int** i=0; i<arr.length; i++)

{

linkedlist.push(**new** node(arr[i]));

}

System.***out***.println("Linked List without sorting is :");

linkedlist.printList(linkedlist.head);

// Apply merge Sort

linkedlist.head = linkedlist.mergeSort(linkedlist.head);

System.***out***.print("\nSorted Linked List is : \n");

linkedlist.printList(linkedlist.head);

//// for(int i=0; i<arr.length; i++)

//// {

// node h = RemoveDuplicates(linkedlist.head);

//// linkedlist2.push(h);

//// }

linkedlist2 = *removeRepeatedElement*(linkedlist.head);

System.***out***.print("\nSorted Linked List is : \n");

linkedlist2.printList(linkedlist2.head);

}

}