



Experiment 3

Student Name: Zatch

UID:

Branch: CSE

Section/Group:

Semester:5

Date of Performance:

Subject Name: AP

Subject Code: 22CSP-314

1. Aim:

You're given the pointer to the head nodes of two linked lists. Compare the data in the nodes of the linked lists to check if they are equal. If all data attributes are equal and the lists are the same length, return 1. Otherwise, return 0. Example The two lists have equal data attributes for the first 3 nodes. Llist2 is longer, though, so the lists are not equal. Return 0.

2. Objective:

Implement a function compare Lists that takes the heads of two singly linked lists as parameters. The function should return 1 if the lists have identical data values in corresponding nodes and are of the same length. Otherwise, it should return 0

3. Implementation/Code:

```
static boolean compareLists(SinglyLinkedListNode head1,  
SinglyLinkedListNode head2) {  
    while (head1 != null && head2 != null) {  
  
        if (head1.data != head2.data) {  
            return false;  
        }  
    }
```



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
        head1 = head1.next;
        head2 = head2.next;
    }

    return head1 == null && head2 == null;

}
```

4. Output:

✔ Test case 0

✔ Test case 1

✔ Test case 2 🔒

✔ Test case 3 🔒

✔ Test case 4 🔒

✔ Test case 5 🔒

✔ Test case 6 🔒

Compiler Message

Success

Input (stdin)

Download

1	2
2	2
3	1
4	2
5	1
6	1
7	2
8	1
9	2

Problem -2

1. Aim:

Problem Statement: Given a reference to the head of a doubly-linked list and an integer, create a new DoublyLinkedListNode object having data value and insert it at the proper location to maintain the sort.

2. Objective:

The objective is to implement a function sortedInsert that achieves the following:

- Inserts a new node with the specified data into a doubly-linked list.
- Ensures that after insertion, the list remains sorted in ascending order based on the data values of the nodes.

3. Code:

```
class Result {  
    public static DoublyLinkedListNode sortedInsert(DoublyLinkedListNode  
l1, int data) {  
        DoublyLinkedListNode newNode = new DoublyLinkedListNode(data);  
  
        // If the list is empty, return the new node  
        if (l1 == null) {  
            return newNode;  
        }  
  
        // If the new node's data is less than the head's data, insert at the beginning  
        if (data < l1.data) {  
            newNode.next = l1;
```

```
        llist.prev = newNode;
        return newNode;
    }

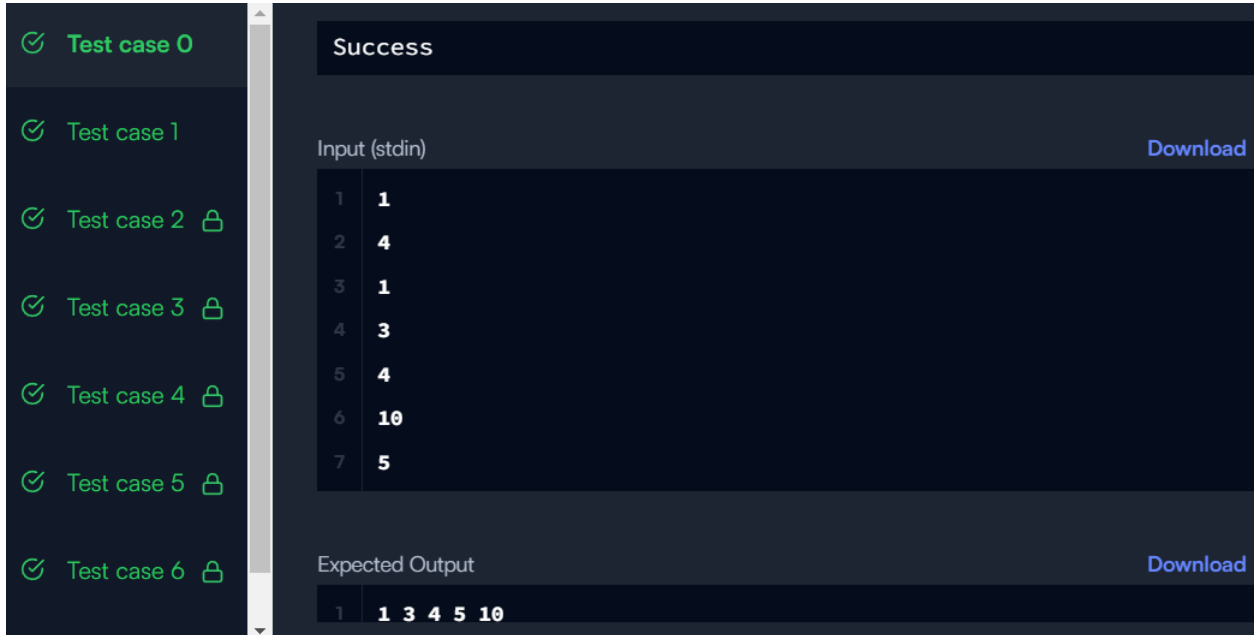
    // Traverse the list to find the correct position for the new node
    DoublyLinkedListNode current = llist;
    while (current.next != null && current.next.data < data) {
        current = current.next;
    }

    // Insert the new node at the correct position
    newNode.next = current.next;
    newNode.prev = current;
    if (current.next != null) {
        current.next.prev = newNode;
    }
    current.next = newNode;

    return llist;
}

}
```

4. Output:



The screenshot displays a user interface for test case results. On the left, a sidebar lists test cases from 0 to 6, all marked as successful with green checkmarks. The main area shows the details for 'Test case 0', which is also successful. It includes a 'Success' status bar, an 'Input (stdin)' section with a table of 7 rows, and an 'Expected Output' section with a single row. Both the input and output sections have a 'Download' link.

Input (stdin)	
1	1
2	4
3	1
4	3
5	4
6	10
7	5

Expected Output	
1	1 3 4 5 10

5. Learning Outcome

- Gain a deeper understanding of doubly-linked list structures, including how nodes are linked both forward (next) and backward (prev).
- Learn techniques for inserting new nodes into a sorted linked list while preserving the sorted order.
- Learn techniques to compare data structures such as linked lists, focusing on comparing individual node data.
- Practice designing algorithms that involve iterative comparison of nodes, ensuring efficient and correct execution.