



Green University of Bangladesh
Department of Computer Science and Engineering (CSE)
Faculty of Sciences and Engineering
Semester: (Fall, Year: 2024), B.Sc. in CSE (Day)

Lab Report NO #01
Course Title: Algorithm Lab
Course Code: CSE 206 Section: 223_E1

Lab Experiment Name: Implement merge sort on a linked list

Student Details

Name		ID
1.	Md. Sium	223015036

Lab Date : 13/09/2024
Submission Date : 20/09/2024
Course Teacher's Name : K.M Shadman Wadith

Lab Report Status

Marks:
Comments:

Signature:.....
Date:.....

1. TITLE OF THE LAB REPORT EXPERIMENT

Implement merge sort on a linked list.

2. OBJECTIVES/AIM

Implement and test a merge sort algorithm for sorting a singly linked list. The implementation includes methods to merge two sorted lists, split a linked list into two halves, and recursively sort the list using merge sort.

3. PROCEDURE / ANALYSIS / DESIGN

1. Initialize the Linked List:
 - Create an instance of the Linked List class. Use the push method to add elements to the linked list. Insert nodes with values 15, 10, 5, 20, 3, and 2.
2. Print the Original Linked List:
 - Use the Node head method to display the contents of the linked list before sorting.
3. Sort the Linked List:
 - The Node head method to sort the linked list. This method applies the merge sort algorithm, which includes: 1. Finding the middle of the list. 2. Merging the two sorted halves.
4. Print the Sorted Linked List:
 - Node head method again to display the contents of the linked list after sorting.
5. Output:
 - Compare the output of the original linked list and the sorted linked list to ensure that the sorting algorithm has worked correctly.

4. IMPLEMENTATION

The main method of the Linked List class tests the implementation by performing the following steps:

1. Creates an instance of Linked List.
2. Inserts nodes with data values 15, 10, 5, 20, 3, and 2 into the list.
3. Prints the original linked list.
4. Sorts the linked list using the merge sort method.
5. Prints the sorted linked list.

Java Code:

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

        Node(int data) {
            this.data = data;
            this.next = null;
        }
    }
}
```

```

    }
}
public void printList(Node head) {
    Node temp = head;
    while (temp != null) {
        System.out.print(temp.data + " ");
        temp = temp.next;
    }
    System.out.println();
}
public Node sortedMerge(Node a, Node b) {
    if (a == null)
        return b;
    if (b == null)
        return a;

    Node result;
    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 Node getMiddle(Node head) {
    if (head == null)
        return head;

    Node slow = head, fast = head;

    while (fast.next != null && fast.next.next != null) {
        slow = slow.next;
        fast = fast.next.next;
    }

    return slow;
}
public Node mergeSort(Node head) {
    if (head == null || head.next == null) {
        return head;
    }
    Node middle = getMiddle(head);
    Node nextOfMiddle = middle.next;
    middle.next = null;
    Node left = mergeSort(head);
    Node right = mergeSort(nextOfMiddle);
    return sortedMerge(left, right);
}

```

```

    }

    public void push(int new_data) {
        Node new_node = new Node(new_data);
        if (head == null) {
            head = new_node;
        } else {
            Node temp = head;
            while (temp.next != null) {
                temp = temp.next;
            }
            temp.next = new_node;
        }
    }

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

        list.push(15);
        list.push(10);
        list.push(5);
        list.push(20);
        list.push(3);
        list.push(2);

        System.out.println("Original Linked List:");
        list.printList(list.head);

        list.head = list.mergeSort(list.head);

        System.out.println("Sorted Linked List:");
        list.printList(list.head);
    }
}

```

5. TEST RESULT / OUTPUT

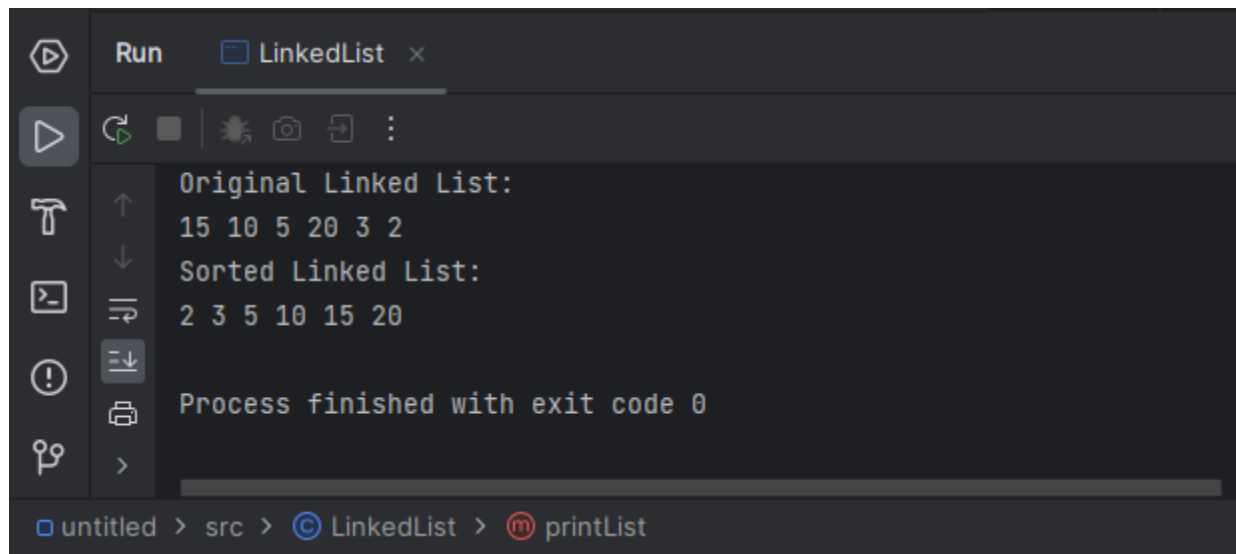
Original Linked List:

15 10 5 20 3 2

Sorted Linked List:

2 3 5 10 15 20

Process finished with exit code 0



The screenshot shows a Java IDE's Run console for a file named 'LinkedList'. The output displays the original linked list as '15 10 5 20 3 2' and the sorted linked list as '2 3 5 10 15 20'. It also indicates that the process finished with exit code 0. The breadcrumb at the bottom shows the file path: 'untitled > src > LinkedList > printList'.

```
Run    LinkedList x
Original Linked List:
15 10 5 20 3 2
Sorted Linked List:
2 3 5 10 15 20
Process finished with exit code 0
untitled > src > LinkedList > printList
```

6. ANALYSIS AND DISCUSSION

Merge sort has a time complexity of $O(n \log n)$, making it efficient for sorting large lists. This efficiency is due to its divide-and-conquer approach, which splits the list into smaller segments and then merges them in sorted order.

Advantages of Merge Sort:

Stability: Merge sort is a stable sorting algorithm, meaning it preserves the relative order of equal elements. This is particularly useful when the stability of element order is important.

Handling of Linked Lists: Merge sort is well-suited for linked lists because it does not require random access to elements. The sorted Merge method merges two sorted lists efficiently, and the get Middle method uses slow and fast pointers to split the list with linear time complexity.

7. SUMMARY:

The implementation of the merge sort algorithm for a singly linked list was successful. The merge sort algorithm efficiently handled the sorting of the list, with the final output confirming the correct order of elements.