

GREEN UNIVERSITY OF BANGLADESH



Department of Computer Science and Engineering (CSE)

Semester: (Fall, Year:2025), B.Sc. in CSE (Day)  
  
Lab Report NO: 01

**Experiment Name: Implement Merge Sort Algorithm.**  
**Course Title** : Algorithm lab.

Course Code: CSE 208 Section: D8

Student Details:

|  |  |  |  |
| --- | --- | --- | --- |
|  | Name |  | ID |
| 1. | Majharul Islam | 232002256 |  |

Submission Date. : 26.02.2025   
Course Teacher’s Name: Feroza Naznin.

[For Teachers use only: Don’t Write Anything inside this box]

Assignment Report Status

Marks: ………………………………… Signature:.....................

Comments:.............................................. Date:..............................

**Merge Sort on a Linked List**

**1. TITLE OF THE LAB REPORT EXPERIMENT**

Implementation of Merge Sort on a Singly Linked List

**2. OBJECTIVES/AIM**

The objective of this experiment is to implement the Merge Sort algorithm on a singly linked list and analyze its efficiency. The goal is to:

* Understand how Merge Sort works on linked lists.
* Implement an efficient sorting algorithm for linked lists.
* Compare Merge Sort’s performance with other sorting techniques for linked lists.

**3. PROCEDURE / ANALYSIS / DESIGN**

**Algorithm:**

1. If the linked list has 0 or 1 node, return it (base case).
2. Split the linked list into two halves using the slow and fast pointer approach.
3. Recursively sort both halves using Merge Sort.
4. Merge the sorted halves using a helper function.
5. Return the sorted linked list.

**Flowchart:**

(You can create a flowchart using any tool or draw it manually.)

**Pseudocode:**

function mergeSort(head):

if head is NULL or head.next is NULL:

return head

middle = getMiddle(head)

nextToMiddle = middle.next

middle.next = NULL

left = mergeSort(head)

right = mergeSort(nextToMiddle)

sortedList = merge(left, right)

return sortedList

**4. IMPLEMENTATION**

class ListNode {

int val;

ListNode next;

ListNode(int val) {

this.val = val;

this.next = null;

}

}

public class MergeSortLinkedList {

// Function to split the linked list

private static ListNode getMiddle(ListNode head) {

if (head == null) return head;

ListNode slow = head, fast = head;

while (fast.next != null && fast.next.next != null) {

slow = slow.next;

fast = fast.next.next;

}

return slow;

}

// Function to merge two sorted linked lists

private static ListNode merge(ListNode left, ListNode right) {

if (left == null) return right;

if (right == null) return left;

if (left.val < right.val) {

left.next = merge(left.next, right);

return left;

} else {

right.next = merge(left, right.next);

return right;

}

}

// Merge Sort function

public static ListNode mergeSort(ListNode head) {

if (head == null || head.next == null) return head;

ListNode middle = getMiddle(head);

ListNode nextToMiddle = middle.next;

middle.next = null;

ListNode left = mergeSort(head);

ListNode right = mergeSort(nextToMiddle);

return merge(left, right);

}

}

**5. TEST RESULT / OUTPUT**

**Test Cases:**

**Input:**

Unsorted Linked List: 4 -> 2 -> 1 -> 3

**Output:**

Sorted Linked List: 1 -> 2 -> 3 -> 4

**Explanation:**

* The linked list is split recursively until each part contains a single node.
* The sorted sublists are merged using the merge function.
* The final output is a sorted linked list.

**6. ANALYSIS AND DISCUSSION**

**What went well?**

* Successfully implemented Merge Sort on a linked list.
* Used a recursive approach to achieve an efficient sorting mechanism.

**Trouble Spots:**

* Handling the base cases correctly.
* Properly splitting the linked list to avoid infinite loops.

**Difficult Parts:**

* Implementing the merge function correctly to handle edge cases.

**Learnings:**

* Merge Sort works well with linked lists due to constant-time insertions and deletions.
* The time complexity of Merge Sort remains **O(n log n)**.

**Mapping of Objectives:**

* Achieved the goal of sorting a linked list using Merge Sort.
* Demonstrated efficient linked list manipulation techniques.

**7. SUMMARY**

Merge Sort is an efficient sorting algorithm for linked lists due to its **O(n log n)** time complexity. The implementation involved recursively splitting and merging the linked list. The experiment reinforced the understanding of recursive sorting algorithms and linked list operations.