

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:** **Sorting a Linked List using Merge Sort and Quick Sort**

**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:..............................

**Sorting a Linked List using Merge Sort and Quick Sort**

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

Implementing Merge Sort and Quick Sort on a Linked List

**2. OBJECTIVES/AIM**

The objective of this experiment is to implement and compare two sorting algorithms—Merge Sort and Quick Sort—on a singly linked list. The goals include:

* Understanding sorting techniques for linked lists.
* Implementing Merge Sort and Quick Sort in Java.
* Analyzing the time complexity of both sorting methods.

**3. PROCEDURE / ANALYSIS / DESIGN**

**Algorithm for Merge Sort:**

1. Divide the linked list into two halves.
2. Recursively sort both halves.
3. Merge the sorted halves.

**Algorithm for Quick Sort:**

1. Choose a pivot element.
2. Partition the linked list into elements smaller and larger than the pivot.
3. Recursively sort both partitions.
4. Concatenate the sorted partitions.

**4. IMPLEMENTATION**

**Merge Sort Implementation:**

|  |
| --- |
| class Node {  int data;  Node next;  Node(int data) {  this.data = data;  this.next = null;  }  }  class MergeSortLinkedList {  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 merge(left, right);  }  Node merge(Node left, Node right) {  if (left == null) return right;  if (right == null) return left;  if (left.data < right.data) {  left.next = merge(left.next, right);  return left;  } else {  right.next = merge(left, right.next);  return right;  }  }  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;  }  void printList(Node head) {  while (head != null) {  System.out.print(head.data + " -> ");  head = head.next;  }  System.out.println("null");  }  public static void main(String[] args) {  MergeSortLinkedList list = new MergeSortLinkedList();  Node head = new Node(4);  head.next = new Node(2);  head.next.next = new Node(1);  head.next.next.next = new Node(3);  head = list.mergeSort(head);  list.printList(head);  }  } |

**Quick Sort Implementation:**

|  |
| --- |
| class QuickSortLinkedList {  Node quickSort(Node head) {  if (head == null || head.next == null) {  return head;  }  Node[] partitioned = partition(head);  Node leftSorted = quickSort(partitioned[0]);  Node rightSorted = quickSort(partitioned[2]);  return concatenate(leftSorted, partitioned[1], rightSorted);  }  Node[] partition(Node head) {  Node pivot = head;  Node smaller = new Node(0), larger = new Node(0);  Node smallHead = smaller, largeHead = larger;  Node curr = head.next;  while (curr != null) {  if (curr.data < pivot.data) {  smaller.next = curr;  smaller = smaller.next;  } else {  larger.next = curr;  larger = larger.next;  }  curr = curr.next;  }  smaller.next = larger.next = null;  return new Node[]{smallHead.next, pivot, largeHead.next};  }  Node concatenate(Node left, Node pivot, Node right) {  if (left == null) {  pivot.next = right;  return pivot;  }  Node temp = left;  while (temp.next != null) temp = temp.next;  temp.next = pivot;  pivot.next = right;  return left;  }  void printList(Node head) {  while (head != null) {  System.out.print(head.data + " -> ");  head = head.next;  }  System.out.println("null");  }  public static void main(String[] args) {  QuickSortLinkedList list = new QuickSortLinkedList();  Node head = new Node(4);  head.next = new Node(2);  head.next.next = new Node(1);  head.next.next.next = new Node(3);  head = list.quickSort(head);  list.printList(head);  }  } |

**5. TEST RESULT / OUTPUT**

**Test Cases:**

**Input:**

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

**Output for Merge Sort:**

1 -> 2 -> 3 -> 4 -> null

**Output for Quick Sort:**

1 -> 2 -> 3 -> 4 -> null

**6. ANALYSIS AND DISCUSSION**

**What went well?**

* Successfully implemented both sorting techniques.
* Demonstrated recursive approaches for sorting linked lists.

**Trouble Spots:**

* Handling the partitioning step in Quick Sort correctly.

**Difficult Parts:**

* Ensuring the sorted linked list is properly merged and concatenated.

**Learnings:**

* Merge Sort is stable and works well for linked lists, with **O(n log n)** complexity.
* Quick Sort's partitioning is trickier but still achieves **O(n log n)** on average.

**Mapping of Objectives:**

* Achieved sorting of a linked list using Merge Sort and Quick Sort.
* Analyzed the efficiency of both algorithms.

**7. SUMMARY**

Merge Sort and Quick Sort can efficiently sort a linked list. Merge Sort follows a divide-and-conquer approach, while Quick Sort relies on partitioning around a pivot. Both achieve **O(n log n)** time complexity but have different advantages in practice. This experiment deepened the understanding of linked list sorting techniques.