

GREEN UNIVERSITY OF BANGLADESH



Department of Computer Science and Engineering (CSE)

Semester: (Fall, Year:2025), B.Sc. in CSE (Day)  
  
Lab Report NO: 02

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

Course Code: CSE 208 Section: D8

Student Details:

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

Submission Date. : 22.02.2025   
Course Teacher’s Name: Feroza Naznin

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

Assignment Report Status

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

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

Quick Sort on a Linked List

1. TITLE OF THE LAB REPORT EXPERIMENT

Implementation of Quick Sort on a Singly Linked List

2. OBJECTIVES/AIM

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

- Understand how Quick Sort works on linked lists.

- Implement an efficient sorting algorithm for linked lists.

- Compare Quick Sort’s performance with other sorting techniques for linked lists.

**3. PROCEDURE / ANALYSIS / DESIGN**

**Algorithm:**

1. If the linked list is empty or has only one node, return it (base case).

2. Select a pivot element (typically the last node).

3. Partition the linked list into two halves:

- Elements smaller than the pivot.

- Elements greater than or equal to the pivot.

4. Recursively apply Quick Sort on both halves.

5. Concatenate the sorted left half, pivot, and sorted right half.

6. Return the sorted linked list.

Flowchart:

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

**Pseudocode:**

function quickSort(head):

if head is NULL or head.next is NULL:

return head

pivot = selectPivot(head)

left, right = partition(head, pivot)

left = quickSort(left)

right = quickSort(right)

return concatenate(left, pivot, right)

4. IMPLEMENTATION

class ListNode {

int val;

ListNode next;

ListNode(int val) {

this.val = val;

this.next = null;

}

}

public class QuickSortLinkedList {

// Function to get the tail node

private static ListNode getTail(ListNode head) {

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

head = head.next;

}

return head;

}

// Partition function

private static ListNode partition(ListNode head, ListNode end, ListNode[] newHead, ListNode[] newEnd) {

ListNode pivot = end;

ListNode prev = null, cur = head, tail = pivot;

while (cur != pivot) {

if (cur.val < pivot.val) {

if (newHead[0] == null) {

newHead[0] = cur;

}

prev = cur;

} else {

if (prev != null) {

prev.next = cur.next;

}

ListNode temp = cur.next;

cur.next = null;

tail.next = cur;

tail = cur;

cur = temp;

continue;

}

cur = cur.next;

}

newHead[0] = (newHead[0] == null) ? pivot : newHead[0];

newEnd[0] = tail;

return pivot;

}

// QuickSort function

private static ListNode quickSort(ListNode head, ListNode end) {

if (head == null || head == end) return head;

ListNode[] newHead = {null}, newEnd = {null};

ListNode pivot = partition(head, end, newHead, newEnd);

if (newHead[0] != pivot) {

ListNode temp = newHead[0];

while (temp.next != pivot) {

temp = temp.next;

}

temp.next = null;

newHead[0] = quickSort(newHead[0], temp);

temp = getTail(newHead[0]);

temp.next = pivot;

}

pivot.next = quickSort(pivot.next, newEnd[0]);

return newHead[0];

}

public static ListNode quickSort(ListNode head) {

return quickSort(head, getTail(head));

}

}

**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 partitioned around a pivot.

- The left and right parts are recursively sorted.

- The final output is a sorted linked list.

**6. ANALYSIS AND DISCUSSION**

What went well?

- Successfully implemented Quick Sort on a linked list.

- Used an in-place partitioning approach to achieve efficient sorting.

Trouble Spots:

- Handling the base cases correctly.

- Maintaining correct list connections during partitioning.

Difficult Parts:

- Implementing the partition function correctly to handle edge cases.

Learnings:

- Quick Sort is an efficient sorting algorithm for linked lists when implemented correctly.

- The time complexity of Quick Sort remains \*\*O(n log n)\*\* in the average case.

Mapping of Objectives:

- Achieved the goal of sorting a linked list using Quick Sort.

- Demonstrated efficient linked list manipulation techniques.

**7. SUMMARY**

Quick Sort is an efficient sorting algorithm for linked lists with an average time complexity of **O(n log n).** The implementation involved selecting a pivot, partitioning the list, and recursively sorting the partitions. The experiment reinforced the understanding of Quick Sort and linked list operations.