

1. TITLE OF THE LAB REPORT EXPERIMENT

QuickSort Implementation Using a Singly Linked List in Java.

2. OBJECTIVES/AIM

The QuickSort algorithm on a singly linked list in Java and to understand how sorting can be applied to dynamic data structures such as linked lists. The QuickSort algorithm is a widely used, efficient, and comparison-based sorting algorithm that follows a divide-and-conquer approach.

3. PROCEDURE / ANALYSIS / DESIGN

1. QuickSort Overview:

QuickSort is a recursive algorithm that selects a "pivot" element and partitions the list such that elements smaller than the pivot come before it, and elements larger than the pivot come after it. After partitioning, the algorithm recursively sorts the sub lists formed on either side of the pivot.

2. QuickSort on Linked Lists:

Unlike arrays, where random access allows simple index swapping, linked lists require more careful manipulation of pointers during sorting. This implementation uses the last element of the list as the pivot and rearranges the nodes without allocating extra memory for arrays.

Algorithm Steps

1. **Node Creation:** A linked list is built with unsorted integer values.
2. **Partitioning:** The last node is chosen as the pivot. The list is rearranged so that smaller elements precede the pivot, and larger elements follow.
3. **Output:** The linked list is printed before and after sorting to demonstrate the QuickSort operation.

4. IMPLEMENTATION

Java Code Implementation:

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

```

}
Node head;
void addToEnd(int data) {
    Node newNode = new Node(data);
    if (head == null) {
        head = newNode;
    } else {
        Node temp = head;
        while (temp.next != null) {
            temp = temp.next;
        }
        temp.next = newNode;
    }
}
Node partitionLast(Node start, Node end) {
    if (start == end || start == null || end == null) {
        return start;
    }
    Node pivot_prev = start;
    Node curr = start;
    int pivot = end.data;
    while (start != end) {
        if (start.data < pivot) {
            pivot_prev = curr;
            int temp = curr.data;
            curr.data = start.data;
            start.data = temp;
            curr = curr.next;
        }
        start = start.next;
    }
    int temp = curr.data;
    curr.data = pivot;
    end.data = temp;
    return pivot_prev;
}

```

```

void quickSortRec(Node start, Node end) {
    if (start == null || start == end || start == end.next)
    {
        return;
    }
    Node pivot_prev = partitionLast(start, end);
    quickSortRec(start, pivot_prev);
    if (pivot_prev != null && pivot_prev == start) {
        quickSortRec(pivot_prev.next, end);
    } else if (pivot_prev != null && pivot_prev.next !=
null) {
        quickSortRec(pivot_prev.next.next, end);
    }
}

Node getTail(Node node) {
    while (node != null && node.next != null) {
        node = node.next;
    }
    return node;
}

void quickSort() {
    Node tail = getTail(head);
    quickSortRec(head, tail);
}

void printList() {
    Node temp = head;
    while (temp != null) {
        System.out.print(temp.data + " ");
        temp = temp.next;
    }
    System.out.println();
}

public static void main(String[] args) {
    QuickSortLinkedList list = new QuickSortLinkedList();
    list.addToEnd(10);
    list.addToEnd(30);
}

```

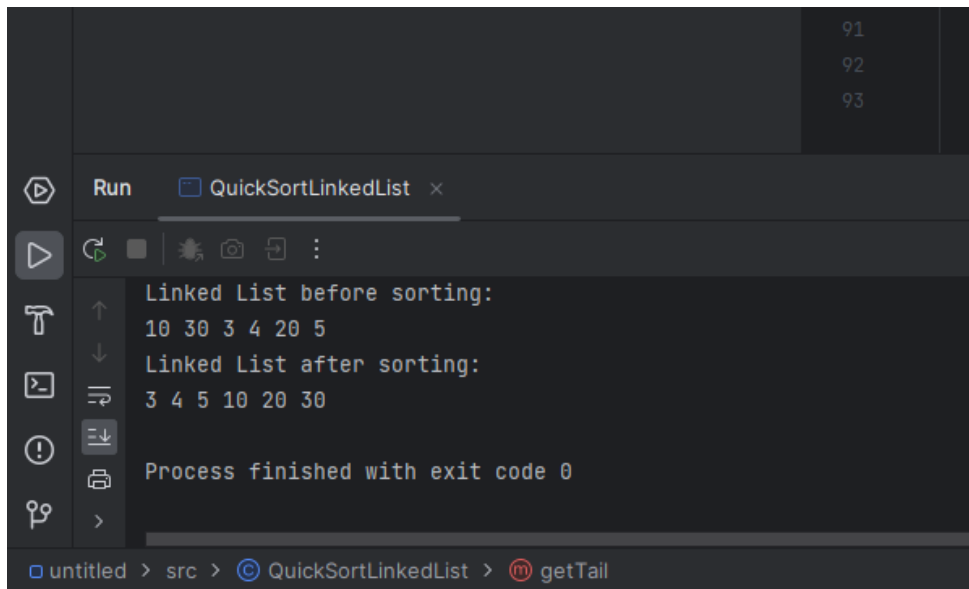
```

        list.addToEnd(3);
        list.addToEnd(4);
        list.addToEnd(20);
        list.addToEnd(5);

        System.out.println("Linked List before sorting:");
        list.printList();
        list.quickSort();
        System.out.println("Linked List after sorting:");
        list.printList();
    }
}

```

5. TEST RESULT / OUTPUT



```

Run QuickSortLinkedList x
Linked List before sorting:
10 30 3 4 20 5
Linked List after sorting:
3 4 5 10 20 30
Process finished with exit code 0
untitled > src > QuickSortLinkedList > getTail

```

6. ANALYSIS AND DISCUSSION

The QuickSort algorithm effectively sorted the singly linked list in ascending order by choosing the last element as the pivot. The program handled pointer manipulation carefully to ensure no extra space was required the original linked list structure.