COMSATS University Islamabad, Attock Campus

Department of Computer Science

Quiz-1 (solution)

CLO-1 (Employ linear data structures to solve computing problems.)

Question 1: Insert an element at the end of a Singly Linked List

Given: Singly Linked List: `10 -> 20 -> 30`

Visual Representation:

- **Before**:

  `10 -> 20 -> 30`
- **After Insertion** of `40`:

  `10 -> 20 -> 30 -> 40`

Pseudo Code:

```
function insert_at_end(value):
    new_node = create_node(value)

    if head is NULL:
        head = new_node
        return

    current = head
    while current.next is not NULL:
        current = current.next

    current.next = new_node
```

Question 2: Insert an element before a specific value in a Singly Linked List

Given: Singly Linked List: `5 -> 15 -> 25 -> 35`

Task: Insert `10` before `15`.
Visual Representation:

- **Before**:

  `5 -> 15 -> 25 -> 35`
- **After Insertion** of `10`:

  `5 -> 10 -> 15 -> 25 -> 35`

Pseudo Code:

```
function insert_before(value, target):
    new_node = create_node(value)

    if head is NULL:
        return NULL

    # If the target is the first node
    if head.data == target:
        new_node.next = head
        head = new_node
        return head

    current = head
    while current.next is not NULL and current.next.data != target:
        current = current.next

    if current.next != NULL:
        new_node.next = current.next
        current.next = new_node
```

Question 3: Insert an Element at a Specific Position in a Doubly Linked List

Given: Doubly Linked List: `100 <-> 200 <-> 300`

Task: Insert `150` between `100` and `200`.
Visual Representation:

- **Before**:

  `100 <-> 200 <-> 300`
- **After Insertion** of `150`:

  `100 <-> 150 <-> 200 <-> 300`

Pseudo Code:

```
function insert_at_position(value, position):
    new_node = create_node(value)

    if position == 1:  # Insert at head
        new_node.next = head
        head.prev = new_node
        head = new_node
        return head
    current = head
    for i = 1 to position - 1:
        current = current.next
        if current is NULL:
            return  # Position out of bounds

    new_node.next = current.next
    new_node.prev = current

    if current.next is not NULL:
        current.next.prev = new_node

    current.next = new_node
```

Question 4: Traverse a Singly Linked List

Given: Singly Linked List: `8 -> 16 -> 32 -> 64`

Task: Write pseudo code to traverse and print the values.
Pseudo Code:

```
function traverse_singly(head):
    current = head
    while current is not NULL:
        print(current.data)
        current = current.next
```

Question 5: Demonstrate how operations are performed on arrays and linked lists

1. Accessing an Element:

- **Array**: Accessing an element at a specific index (e.g., 4th element) is direct and takes **O(1)** time. The array allows random access using an index.
- **Linked List**: Accessing an element in a linked list requires traversing from the head node until reaching the desired position. This operation takes **O(n)** time as it involves sequential traversal.

2. Deleting an Element from the End:

- **Array**: Deleting an element from the end of an array is **O(1)** if there is no resizing required. However, in a dynamic array (e.g., C++ `vector`), resizing could result in **O(n)**.
- **Linked List**: Deleting from the end of a singly linked list requires traversing to the second-last element to update its `next` pointer, which takes **O(n)** time. In a doubly linked list, it can be done in **O(1)** if a reference to the last node is maintained.