Arrays – Insertion and Deletion

1. Introduction

Insertion and **Deletion** are fundamental operations performed on arrays to **add a new element** or **remove an existing element**.

Since arrays use **contiguous memory and fixed size**, these operations require **shifting elements**, making them more expensive compared to simple access or traversal.

Understanding insertion and deletion helps learners grasp:

- How memory shifting works
- Why arrays are less flexible than linked lists
- The cost of modifying data structures

---

2. Array Insertion

2.1 What is Array Insertion?

Array insertion is the process of **adding a new element at a specified position** in an array.

Since arrays have fixed size:

- A new element cannot be inserted if the array is already full
- Existing elements may need to be shifted to create space

2.2 Types of Insertion in Arrays

1️⃣ Insertion at the Beginning

- New element is placed at index 0
- All existing elements are shifted **one position to the right**
- Most expensive insertion type

2️⃣ Insertion at a Given Position

- Element is inserted at a specific index
- Elements from that position onwards are shifted right

3️⃣ Insertion at the End

- Element is added at the last index
- No shifting required (if space exists)
- Least expensive insertion

2.3 Logical Steps for Insertion

1. Check if array has space
2. Decide insertion position
3. Shift elements to the right (if needed)
4. Insert the new element
5. Increase the array size

2.4 Time Complexity of Insertion

- Best Case (Insertion at end): **O(1)**
- Worst Case (Insertion at beginning): **O(n)**

---

3. Array Deletion

3.1 What is Array Deletion?

Array deletion is the process of **removing an element from a specified position** in an array.

Deletion does not reduce array size physically but:

- Shifts elements to fill the gap
- Logical size of the array is reduced

3.2 Types of Deletion in Arrays

1️⃣ Deletion from the Beginning

- First element is removed
- All remaining elements shift **one position to the left**
- Most expensive deletion

2️⃣ Deletion from a Given Position

- Element at a specific index is removed
- Elements after that position are shifted left

3️⃣ Deletion from the End

- Last element is removed
- No shifting required
- Least expensive deletion

3.3 Logical Steps for Deletion

1. Check if array is empty
2. Identify the position to delete
3. Shift elements left to fill the gap
4. Reduce logical size of the array

3.4 Time Complexity of Deletion

- Best Case (Deletion at end): **O(1)**
- Worst Case (Deletion at beginning): **O(n)**

---

4. Visualization of Insertion and Deletion

Insertion Example:

```
Before: [10, 20, 30, 40]
Insert 25 at index 2
After:  [10, 20, 25, 30, 40]
```

Deletion Example:

```
Before: [10, 20, 30, 40]
Delete element at index 1
After:  [10, 30, 40]
```

Elements shift to maintain contiguous memory.

---

5. Advantages of Insertion and Deletion in Arrays

- Simple logic
- Easy to implement
- Useful for static datasets
- Good for small-sized arrays

---

6. Limitations of Insertion and Deletion

- Fixed size arrays cannot grow
- Shifting elements increases time complexity
- Inefficient for frequent insertions/deletions
- Memory wastage if size is overestimated

---

7. Real-World Examples

- Adding a new score to a leaderboard
- Removing an invalid sensor reading
- Updating attendance records

- Deleting completed tasks from a list

---

8. Comparison: Insertion vs Deletion

Aspect	Insertion	Deletion
Purpose	Add element	Remove element
Shifting	Right shift	Left shift
Worst Time Complexity	O(n)	O(n)
Best Case	O(1)	O(1)

---

9. Summary

- Insertion adds a new element to an array
- Deletion removes an existing element
- Both operations may require shifting elements
- Time complexity depends on position
- Arrays are inefficient for frequent insertions and deletions

---