4. Analysis:

Time Complexity:

- addEmployee(): O(1)

→ Adding at the end of the array requires constant time if space is available.

- searchEmployee(): O(n)

→ In the worst case, we may need to search through the entire array.

- traverseEmployees(): O(n)

→ We must visit every element in the array.

- deleteEmployee(): O(n)

→ After finding the index, we shift all subsequent elements one position to the left.

Space Complexity:

- The space complexity is O(n), where n is the size of the array.

- Memory for array is pre-allocated even if not fully used.

Limitations of Arrays:

- Fixed size: Cannot grow dynamically; must be resized manually or by using another data structure.

- Wastage of memory if actual usage is significantly less than declared capacity.

- Insertion and deletion in the middle are inefficient due to element shifting.

When to Use Arrays:

- When the size of the data is known and will not change.

- When you need fast access to elements by index.

- When memory constraints and performance requirements are minimal.

Real-World Recommendation:

- For scalable and dynamic applications, use ArrayList or LinkedList.

- Use HashMap when key-based access (e.g., employeeId) is frequent.