Analysis

Time Complexity Analysis

1. **Add Employee**
   - Time Complexity: O(1) if adding at the end (assuming space is available)
   - Time Complexity: O(n) if resizing is required in a dynamic array
   - Explanation: Adding to the end of an array is fast when space is available, but resizing the array involves copying all elements to a new array.
2. **Search Employee**
   - Time Complexity: O(n) in the worst case
   - Explanation: A linear search is required unless the array is sorted or indexed, resulting in scanning each element until a match is found.
3. **Traverse Employees**
   - Time Complexity: O(n)
   - Explanation: Each element in the array is visited once to process or display employee details.
4. **Delete Employee**
   - Time Complexity: O(n)
   - Explanation: Once the employee is located, all subsequent elements need to be shifted one position left to fill the gap.

Limitations of Arrays

- Arrays have a fixed size (in static implementations), requiring prior knowledge of the number of elements or manual resizing in dynamic arrays.
- Insertion and deletion at positions other than the end are inefficient due to element shifting.
- Arrays provide poor performance for frequent searches unless additional structures (like indexing) are used.
- Wasted memory may occur if over-allocated; under-allocation leads to frequent resizing.

When to Use Arrays

- When the number of records is known and relatively small.
- When fast, constant-time indexed access is needed.
- In applications where insertions and deletions are rare or only occur at the end.

- For simple or low-performance requirements where minimal overhead is preferred.