Performance Analysis

- **Time Complexity Analysis**
    - Arrays: O(1) for access, O(n) for insertion/deletion.
    - Stacks: O(1) for push/pop/peek.
    - Queues: O(1) for enqueue, O(n) for dequeue using array implementation.
    - Linked Lists: O(1) for insertion at head, O(n) for deletion/access.
- **Trade-offs**
    - Arrays offer faster access times, while linked lists provide dynamic resizing and ease of insertion/deletion.
    - Stacks implemented with arrays may waste space if not carefully managed, whereas linked list stacks are more memory efficient.
- **Efficiency in Scenarios**
    - Arrays can be used for fixed-size collections that needs fast access.
    - Linked lists is used for dynamic collections where frequent insertions/deletions occur.

Discussion

- **Practical Applications:**
    - Arrays: Used in static data storage, image processing.
    - Stacks: Function calls, backtracking algorithms (e.g., depth-first search).
    - Queues: Task scheduling, breadth-first search algorithms.
    - Linked Lists: Dynamic memory allocation, implementing complex data structures like hash tables.