Exercise 1: Inventory Management System

File: Inventory.java

Time Complexity Analysis:

- **Add Product:** O(1) - Adding a product to a HashMap is on average O(1) due to the constant time complexity of hash table insertion.
- **Update Product:** O(1) - Updating a product in a HashMap is also O(1) on average, as it involves accessing the element by key and modifying its fields.
- **Delete Product:** O(1) - Removing a product from a HashMap is O(1) on average, as it involves finding the key and deleting the corresponding entry.
- **Display Products:** O(n) - Displaying all products requires iterating through all the elements, so the complexity is O(n), where n is the number of products.

Optimization: The use of HashMap provides average O(1) time complexity for add, update, and delete operations, making it an optimal choice for this use case. However, in cases of hash collisions, the worst-case time complexity could degrade to O(n).

Exercise 2: E-commerce Platform Search Function

File: ProductSearch.java

Time Complexity Analysis:

- **Linear Search:** O(n) - In the worst case, the algorithm needs to check each element once, where n is the number of products.
- **Binary Search:** O(log n) - Since the array is sorted, binary search can halve the search space at each step, resulting in a logarithmic time complexity.

Optimization: Binary search is preferred over linear search for sorted data due to its O(log n) complexity compared to O(n) for linear search. However, binary search requires the data to be sorted, which could involve additional preprocessing time.

Exercise 3: Sorting Customer Orders

File: OrderSorting.java

Time Complexity Analysis:

- **Bubble Sort:** $O(n^2)$ - In the worst case, bubble sort requires n passes through the list, with each pass taking up to n comparisons.
- **Quick Sort:** O(n log n) on average and $O(n^2)$ in the worst case. Quick Sort is generally faster due to its efficient partitioning, though it can degrade to $O(n^2)$ with poor pivot choices.

Optimization: Quick Sort is generally preferred over Bubble Sort due to its average-case time complexity of $O(n \log n)$. Choosing a good pivot (e.g., using the median-of-three method) can help avoid the worst-case $O(n^2)$ scenario.

Exercise 4: Employee Management System

File: EmployeeManagement.java

Time Complexity Analysis:

- **Add Employee:** O(1) - Adding an employee to the array is O(1) as long as there is space.
- **Search Employee:** O(n) - Linear search through the array requires O(n) time.
- **Delete Employee:** O(n) - Finding the employee requires O(n) time, and shifting elements after deletion also requires O(n) time in the worst case.
- **Display Employees:** O(n) - Iterating through the array to display employees takes O(n) time.

Optimization: Arrays provide O(1) time complexity for accessing elements but have limitations like fixed size and O(n) deletion time due to shifting elements. Dynamic data structures like linked lists or hash tables may be more efficient for certain operations.

Exercise 5: Task Management System

File: TaskManagement.java

Time Complexity Analysis:

- **Add Task:** O(1) - Adding a task to the end of a singly linked list is O(1).
- **Search Task:** O(n) - Linear search through the linked list requires O(n) time.
- **Delete Task:** O(n) - Finding the task requires O(n) time, and adjusting pointers after deletion also requires O(n) time in the worst case.
- **Display Tasks:** O(n) - Iterating through the list to display tasks takes O(n) time.

Optimization: Linked lists are efficient for dynamic data where frequent insertions and deletions occur. However, they have O(n) search time complexity. Doubly linked lists could improve efficiency for certain operations like deletion.

Exercise 6: Library Management System

File: LibraryManagement.java

Time Complexity Analysis:

- **Linear Search:** O(n) - Linear search through the list of books requires O(n) time.

- **Binary Search:** O(log n) - For a sorted list, binary search has a logarithmic time complexity.

Optimization: Binary search is optimal for sorted data due to its O(log n) time complexity. Linear search can be useful for small, unsorted datasets where the cost of sorting may not be justified.

Exercise 7: Financial Forecasting

File: FinancialForecasting.java

Time Complexity Analysis:

- **Recursive Prediction Calculation:** O(n) - The time complexity is linear as the function calls itself recursively n times.

Optimization: The recursive approach can be optimized using memoization to store previously computed results and avoid redundant calculations. This technique can reduce the time complexity to O(n) with respect to the depth of the recursive calls.