### **Exercise 1: Inventory Management System**

**Explanation:** This exercise demonstrates how to create an inventory management system to keep track of products. The Product class defines the properties of a product such as ID, name, quantity, and price. The Inventory class uses a HashMap to store and manage products. You can add, update, delete, and retrieve products from the inventory. The main function shows examples of how to use these methods.

**Complexity:**

* Add product: O(1)O(1)O(1)
* Update product: O(1)O(1)O(1)
* Delete product: O(1)O(1)O(1)
* Search product: O(1)O(1)O(1)

**Importance of Data Structure:**

* HashMap is chosen for its average-case constant time complexity for insertion, deletion, and search operations, making it efficient for managing large inventories.

### **Exercise 2: E-commerce Platform Search Function**

**Explanation:** This exercise focuses on implementing search functionalities for an e-commerce platform. The Product class defines the properties of a product such as ID, name, and category. The Search class contains methods for linear search and binary search to find products by their name. The products are first sorted alphabetically, and then both search methods are demonstrated in the main function.

**Complexity:**

* Linear search: O(n)O(n)O(n)
* Binary search: O(log⁡n)O(\log n)O(logn) (after sorting)

**Importance of Data Structure:**

* Linear search is simple and effective for unsorted lists but inefficient for large datasets.
* Binary search is more efficient but requires the data to be sorted, making it suitable for large, sorted datasets.

### **Exercise 3: Sorting Customer Orders**

**Explanation:** This exercise covers sorting customer orders based on their total price. The Order class defines the properties of an order such as ID, customer name, and total price. The Sorting class provides implementations of bubble sort and quicksort algorithms to sort the orders. The main function demonstrates how to sort orders using both algorithms and print the sorted results.

**Complexity:**

* Bubble Sort: O(n2)O(n^2)O(n2)
* Quick Sort: O(nlog⁡n)O(n \log n)O(nlogn) (average), O(n2)O(n^2)O(n2) (worst)

**Importance of Algorithm:**

* Bubble Sort is simple but inefficient for large datasets due to its quadratic time complexity.
* Quick Sort is generally preferred due to its average-case logarithmic time complexity and practical performance.

### **Exercise 4: Employee Management System**

**Explanation:** This exercise involves creating an employee management system to handle employee records. The Employee class defines the properties of an employee such as ID, name, position, and salary. The EmployeeManagementSystem class manages an array of employees, allowing you to add, search, traverse, and delete employees. The main function demonstrates adding, searching, and deleting employees and prints the employee list.

**Complexity:**

* Add employee: O(1)O(1)O(1) (if there's space), O(n)O(n)O(n) (if resizing is needed)
* Search employee: O(n)O(n)O(n)
* Traverse employees: O(n)O(n)O(n)
* Delete employee: O(n)O(n)O(n)

**Importance of Data Structure:**

* Arrays provide constant time access and are simple to implement but have limitations like fixed size and expensive resizing operations. Suitable for a relatively static number of elements.

### **Exercise 5: Task Management System**

**Explanation:** This exercise demonstrates how to manage tasks using a linked list. The Task class defines the properties of a task such as ID, name, and status, and includes a reference to the next task in the list. The TaskManagementSystem class manages the linked list, allowing you to add, search, traverse, and delete tasks. The main function shows examples of adding, searching, and deleting tasks and prints the task list.

**Complexity:**

* Add task: O(n)O(n)O(n)
* Search task: O(n)O(n)O(n)
* Traverse tasks: O(n)O(n)O(n)
* Delete task: O(n)O(n)O(n)

**Importance of Data Structure:**

* Linked lists allow for dynamic memory allocation, efficient insertion, and deletion but have linear time complexity for access and search operations. Suitable for applications with frequent insertions and deletions.

### **Exercise 6: Library Management System**

**Explanation:** This exercise focuses on implementing a search function for a library management system. The Book class defines the properties of a book such as ID, title, and author. The Library class provides linear and binary search methods to find books by their title. The books are first sorted alphabetically, and both search methods are demonstrated in the main function.

**Complexity:**

* Linear search: O(n)O(n)O(n)
* Binary search: O(log⁡n)O(\log n)O(logn) (after sorting)

**Importance of Data Structure:**

* Linear search is straightforward and works on unsorted data but is inefficient for large datasets.
* Binary search is much faster for sorted data and ideal for large datasets where search efficiency is critical.

### **Exercise 7: Financial Forecasting**

**Explanation:** This exercise involves predicting the future value of an investment using recursive calculation. The FinancialForecasting class contains a method to predict the future value based on the current value, growth rate, and number of periods. The main function demonstrates how to use this method to predict the future value of an investment over a given period with a specified growth rate.

**Complexity:**

* Recursive prediction: O(n)O(n)O(n)

**Importance of Algorithm:**

* Recursion simplifies the implementation of problems that can be broken down into smaller subproblems. However, it can lead to excessive computation and stack overflow if not optimized (e.g., using memoization or converting to an iterative approach).