**Exercise 1: Inventory Management System**

**Scenario:**

You are developing an inventory management system for a warehouse. Efficient data storage and retrieval are crucial.

**Steps:**

1. **Understand the Problem:**
   * Explain why data structures and algorithms are essential in handling large inventories.
   * Discuss the types of data structures suitable for this problem.
2. **Setup:**
   * Create a new project for the inventory management system.
3. **Implementation:**
   * Define a class Product with attributes like **productId**, **productName**, **quantity**, and **price**.
   * Choose an appropriate data structure to store the products (e.g., ArrayList, HashMap).
   * Implement methods to add, update, and delete products from the inventory.
4. **Analysis:**
   * Analyze the time complexity of each operation (add, update, delete) in your chosen data structure.
   * Discuss how you can optimize these operations.

**SOLUTION:**

import java.util.HashMap;

import java.util.Map;

class Product {

String productId;

String productName;

int quantity;

double price;

public Product(String productId, String productName, int quantity, double price) {

this.productId = productId;

this.productName = productName;

this.quantity = quantity;

this.price = price;

}

@Override

public String toString() {

return "Product{" +

"productId='" + productId + '\'' +

", productName='" + productName + '\'' +

", quantity=" + quantity +

", price=" + price +

'}';

}

}

class Inventory {

private Map<String, Product> products;

public Inventory() {

this.products = new HashMap<>();

}

public void addProduct(Product product) {

if (products.containsKey(product.productId)) {

System.out.println("Error: Product with ID " + product.productId + " already exists.");

} else {

products.put(product.productId, product);

System.out.println("Added: " + product.productName);

}

}

public void updateProductQuantity(String productId, int newQuantity) {

Product product = products.get(productId);

if (product != null) {

product.quantity = newQuantity;

System.out.println("Updated quantity for " + product.productName + " to " + newQuantity);

} else {

System.out.println("Error: Product with ID " + productId + " not found.");

}

}

public void deleteProduct(String productId) {

if (products.containsKey(productId)) {

Product removedProduct = products.remove(productId);

System.out.println("Deleted: " + removedProduct.productName);

} else {

System.out.println("Error: Product with ID " + productId + " not found.");

}

}

public void viewProduct(String productId) {

Product product = products.get(productId);

if (product != null) {

System.out.println(product);

} else {

System.out.println("Error: Product with ID " + productId + " not found.");

}

}

public void viewAllProducts() {

if (products.isEmpty()) {

System.out.println("Inventory is empty.");

return;

}

for (Product product : products.values()) {

System.out.println(product);

}

}

}

public class Main {

public static void main(String[] args) {

System.out.println("Inventory Management System Initialized.");

Inventory inventory = new Inventory();

System.out.println("\n--- Adding Products ---");

inventory.addProduct(new Product("P001", "Laptop", 50, 1200.00));

inventory.addProduct(new Product("P002", "Mouse", 200, 25.00));

inventory.addProduct(new Product("P003", "Keyboard", 150, 75.00));

System.out.println("\n--- Current Inventory ---");

inventory.viewAllProducts();

System.out.println("\n--- Updating a Product ---");

inventory.updateProductQuantity("P001", 45);

inventory.viewProduct("P001");

System.out.println("\n--- Deleting a Product ---");

inventory.deleteProduct("P002");

System.out.println("\n--- Checking Deleted Product ---");

inventory.viewProduct("P002");

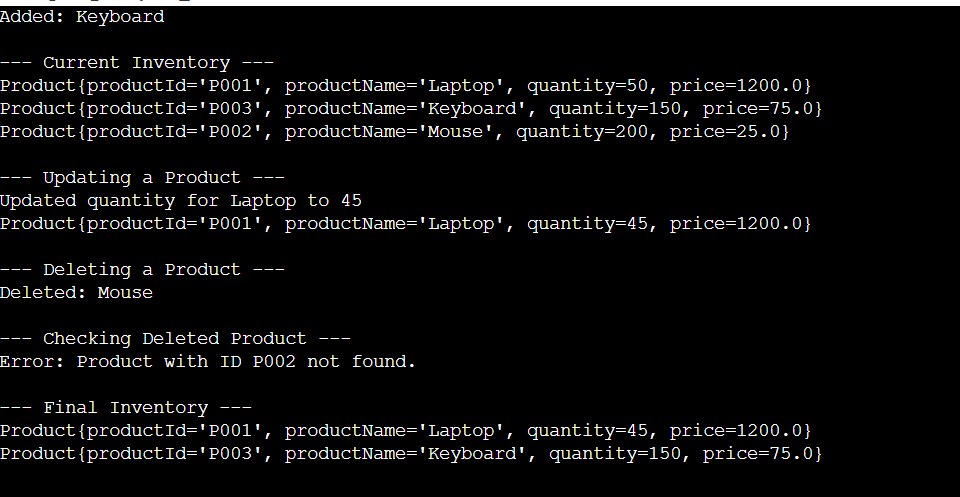
System.out.println("\n--- Final Inventory ---");

inventory.viewAllProducts();

}

}

**OUTPUT:**



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

**Scenario:**

You are working on the search functionality of an e-commerce platform. The search needs to be optimized for fast performance.

**Steps:**

1. **Understand Asymptotic Notation:**
   * Explain Big O notation and how it helps in analyzing algorithms.
   * Describe the best, average, and worst-case scenarios for search operations.
2. **Setup:**
   * Create a class **Product** with attributes for searching, such as **productId, productName**, and **category**.
3. **Implementation:**
   * Implement linear search and binary search algorithms.
   * Store products in an array for linear search and a sorted array for binary search.
4. **Analysis:**
   * Compare the time complexity of linear and binary search algorithms.
   * Discuss which algorithm is more suitable for your platform and why.

**SOLUTION:**

import time

class Product:

    def \_\_init\_\_(self, productId, productName, category):

        self.productId = productId

        self.productName = productName

        self.category = category

    def \_\_repr\_\_(self):

        return f"Product(id={self.productId}, name='{self.productName}', category='{self.category}')"

def linear\_search(products, target\_id):

    for product in products:

        if product.productId == target\_id:

            return product

    return None

def binary\_search(products, target\_id):

    low = 0

    high = len(products) - 1

    while low <= high:

        mid = (low + high) // 2

        if products[mid].productId == target\_id:

            return products[mid]

        elif products[mid].productId < target\_id:

            low = mid + 1

        else:

            high = mid - 1

    return None

products = [

    Product("P001", "Laptop", "Electronics"),

    Product("P002", "T-shirt", "Apparel"),

    Product("P003", "Coffee Maker", "Home Appliances"),

    Product("P004", "Book", "Books"),

    Product("P005", "Mouse", "Electronics"),

    Product("P006", "Jeans", "Apparel"),

]

sorted\_products = sorted(products, key=lambda p: p.productId)

print("--- Linear Search ---")

start\_time = time.time()

found\_product = linear\_search(products, "P003")

end\_time = time.time()

print(f"Found: {found\_product}")

print(f"Linear search took: {end\_time - start\_time:.6f} seconds")

start\_time = time.time()

found\_product = linear\_search(products, "P007")

end\_time = time.time()

print(f"Found: {found\_product}")

print(f"Linear search (not found) took: {end\_time - start\_time:.6f} seconds")

print("\n--- Binary Search ---")

start\_time = time.time()

found\_product = binary\_search(sorted\_products, "P003")

end\_time = time.time()

print(f"Found: {found\_product}")

print(f"Binary search took: {end\_time - start\_time:.6f} seconds")

start\_time = time.time()

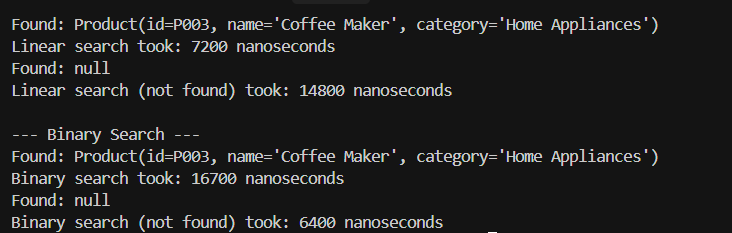
found\_product = binary\_search(sorted\_products, "P007")

end\_time = time.time()

print(f"Found: {found\_product}")

print(f"Binary search (not found) took: {end\_time - start\_time:.6f} seconds")

**OUTPUT:**



**Exercise 3: Sorting Customer Orders**

**Scenario:**

You are tasked with sorting customer orders by their total price on an e-commerce platform. This helps in prioritizing high-value orders.

**Steps:**

1. **Understand Sorting Algorithms:**
   * Explain different sorting algorithms (Bubble Sort, Insertion Sort, Quick Sort, Merge Sort).
2. **Setup:**
   * Create a class **Order** with attributes like **orderId**, **customerName**, and **totalPrice**.
3. **Implementation:**
   * Implement **Bubble Sort** to sort orders by **totalPrice**.
   * Implement **Quick Sort** to sort orders by **totalPrice**.
4. **Analysis:**
   * Compare the performance (time complexity) of Bubble Sort and Quick Sort.
   * Discuss why Quick Sort is generally preferred over Bubble Sort.

**SOLUTION:**

**Order.java:**

public class Order {

    private String orderId;

    private String customerName;

    private double totalPrice;

    public Order(String *orderId*, String *customerName*, double *totalPrice*) {

        this.orderId = *orderId*;

        this.customerName = *customerName*;

        this.totalPrice = *totalPrice*;

    }

    public String getOrderId() {

        return orderId;

    }

    public String getCustomerName() {

        return customerName;

    }

    public double getTotalPrice() {

        return totalPrice;

    }

    @Override

    public String toString() {

        return "Order{" +

                "orderId='" + orderId + '\'' +

                ", customerName='" + customerName + '\'' +

                ", totalPrice=" + totalPrice +

                '}';

    }

}

**OrderSorting.java:**

import java.util.Arrays;

public class OrderSorting {

    public static void **bubbleSort**(Order[] *orders*) {

        int n = *orders*.length;

        for (int i = 0; i < n - 1; i++) {

            for (int j = 0; j < n - i - 1; j++) {

                if (*orders*[j].getTotalPrice() > *orders*[j + 1].getTotalPrice()) {

*// Swap orders[j] and orders[j+1]*

                    Order temp = *orders*[j];

*orders*[j] = *orders*[j + 1];

*orders*[j + 1] = temp;

                }

            }

        }

    }

    public static void **quickSort**(Order[] *orders*, int *low*, int *high*) {

        if (*low* < *high*) {

            int pi = partition(*orders*, *low*, *high*);

            quickSort(*orders*, *low*, pi - 1);

            quickSort(*orders*, pi + 1, *high*);

        }

    }

    private static int **partition**(Order[] *orders*, int *low*, int *high*) {

        double pivot = *orders*[*high*].getTotalPrice();

        int i = (*low* - 1);

        for (int j = *low*; j < *high*; j++) {

            if (*orders*[j].getTotalPrice() <= pivot) {

                i++;

                Order temp = *orders*[i];

*orders*[i] = *orders*[j];

*orders*[j] = temp;

            }

        }

        Order temp = *orders*[i + 1];

*orders*[i + 1] = *orders*[*high*];

*orders*[*high*] = temp;

        return i + 1;

    }

    public static void **main**(String[] *args*) {

        Order[] orders = {

                new Order("1", "Alice", 250.50),

                new Order("2", "Bob", 150.75),

                new Order("3", "Charlie", 300.00),

                new Order("4", "David", 200.25),

                new Order("5", "Eve", 100.00)

        };

        System.out.println("Original Orders:");

        System.out.println(Arrays.toString(orders));

        Order[] bubbleSortedOrders = Arrays.copyOf(orders, orders.length);

        bubbleSort(bubbleSortedOrders);

        System.out.println("\nOrders sorted by Bubble Sort:");

        System.out.println(Arrays.toString(bubbleSortedOrders));

        Order[] quickSortedOrders = Arrays.copyOf(orders, orders.length);

        quickSort(quickSortedOrders, 0, quickSortedOrders.length - 1);

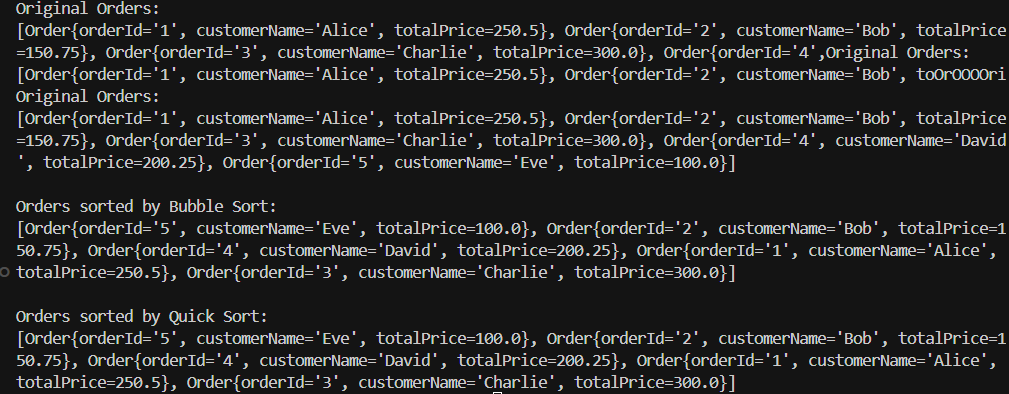
        System.out.println("\nOrders sorted by Quick Sort:");

        System.out.println(Arrays.toString(quickSortedOrders));

    }

}

**OUTPUT:**



**Exercise 4: Employee Management System**

**Scenario:**

You are developing an employee management system for a company. Efficiently managing employee records is crucial.

**Steps:**

1. **Understand Array Representation:**
   * Explain how arrays are represented in memory and their advantages.
2. **Setup:**
   * Create a class Employee with attributes like **employeeId**, **name**, **position**, and **salary**.
3. **Implementation:**
   * Use an array to store employee records.
   * Implement methods to **add**, **search**, **traverse**, and **delete** employees in the array.
4. **Analysis:**
   * Analyze the time complexity of each operation (add, search, traverse, delete).
   * Discuss the limitations of arrays and when to use them.

**SOLUTION:**

**Employee.java**

public class Employee {

    private String employeeId;

    private String name;

    private String position;

    private double salary;

    public Employee(String *employeeId*, String *name*, String *position*, double *salary*) {

        this.employeeId = *employeeId*;

        this.name = *name*;

        this.position = *position*;

        this.salary = *salary*;

    }

    public String getEmployeeId() {

        return employeeId;

    }

    public String getName() {

        return name;

    }

    public String getPosition() {

        return position;

    }

    public double getSalary() {

        return salary;

    }

    @Override

    public String toString() {

        return "Employee{" +

                "employeeId='" + employeeId + '\'' +

                ", name='" + name + '\'' +

                ", position='" + position + '\'' +

                ", salary=" + salary +

                '}';

    }

}

**EmployeeManagementSystem.java:**

public class EmployeeManagementSystem {

    private Employee[] employees;

    private int size;

    private static final int INITIAL\_CAPACITY = 10;

    public EmployeeManagementSystem() {

        employees = new Employee[INITIAL\_CAPACITY];

        size = 0;

    }

    public void **addEmployee**(Employee *employee*) {

        if (size == employees.length) {

            Employee[] newEmployees = new Employee[employees.length \* 2];

            System.arraycopy(employees, 0, newEmployees, 0, employees.length);

            employees = newEmployees;

        }

        employees[size++] = *employee*;

    }

    public Employee **searchEmployee**(String *employeeId*) {

        for (int i = 0; i < size; i++) {

            if (employees[i].getEmployeeId().equals(*employeeId*)) {

                return employees[i];

            }

        }

        return null;

    }

    public void **traverseEmployees**() {

        for (int i = 0; i < size; i++) {

            System.out.println(employees[i]);

        }

    }

    public void **deleteEmployee**(String *employeeId*) {

        int indexToDelete = -1;

        for (int i = 0; i < size; i++) {

            if (employees[i].getEmployeeId().equals(*employeeId*)) {

                indexToDelete = i;

                break;

            }

        }

        if (indexToDelete != -1) {

            for (int i = indexToDelete; i < size - 1; i++) {

                employees[i] = employees[i + 1];

            }

            employees[size - 1] = null;

            size--;

        }

    }

    public static void **main**(String[] *args*) {

        EmployeeManagementSystem ems = new EmployeeManagementSystem();

        ems.addEmployee(new Employee("E001", "Alice", "Developer", 75000));

        ems.addEmployee(new Employee("E002", "Bob", "Manager", 95000));

        ems.addEmployee(new Employee("E003", "Charlie", "Analyst", 60000));

        System.out.println("All employees:");

        ems.traverseEmployees();

        System.out.println("\nSearching for employee E002:");

        System.out.println(ems.searchEmployee("E002"));

        System.out.println("\nDeleting employee E002:");

        ems.deleteEmployee("E002");

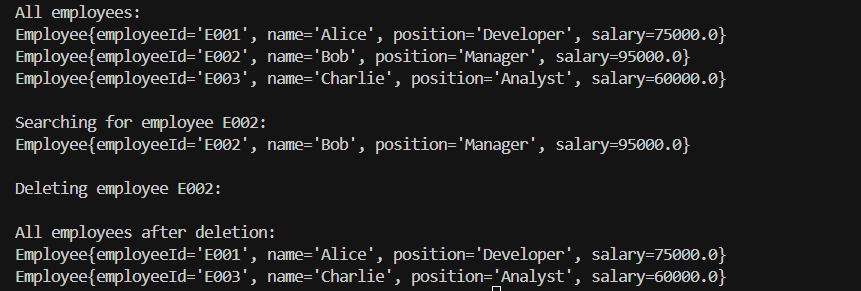
        System.out.println("\nAll employees after deletion:");

        ems.traverseEmployees();

    }

}

**OUTPUT:**

****

**Exercise 5: Task Management System**

**Scenario:**

You are developing a task management system where tasks need to be added, deleted, and traversed efficiently.

**Steps:**

1. **Understand Linked Lists:**
   * Explain the different types of linked lists (Singly Linked List, Doubly Linked List).
2. **Setup:**
   * Create a class **Task** with attributes like **taskId**, **taskName**, and **status**.
3. **Implementation:**
   * Implement a singly linked list to manage tasks.
   * Implement methods to **add**, **search**, **traverse**, and **delete** tasks in the linked list.
4. **Analysis:**
   * Analyze the time complexity of each operation.
   * Discuss the advantages of linked lists over arrays for dynamic data.

**SOLUTION:**

**Task.class:**

public class Task {

   private int taskId;

   private String taskName;

   private String status;

   public Task(int *var1*, String *var2*, String *var3*) {

      this.taskId = var1;

      this.taskName = var2;

      this.status = var3;

   }

   public int getTaskId() {

      return this.taskId;

   }

   public String getTaskName() {

      return this.taskName;

   }

   public String getStatus() {

      return this.status;

   }

   public void setTaskId(int *var1*) {

      this.taskId = var1;

   }

   public void setTaskName(String *var1*) {

      this.taskName = var1;

   }

   public void setStatus(String *var1*) {

      this.status = var1;

   }

   public String toString() {

      return "ID: " + this.taskId + ", Name: " + this.taskName + ", Status: " + this.status;

   }

}

**Node.class:**

public class Node {

   private Task task;

   private Node next;

   public Node(Task *var1*) {

      this.task = var1;

      this.next = null; }

   public Task getTask() {

      return this.task; }

   public Node getNext() {

      return this.next; }

   public void setTask(Task *var1*) {

      this.task = var1; }

   public void setNext(Node *var1*) {

      this.next = var1;

   }

}

**SinglyLinkedList.class:**

public class SinglyLinkedList {

   private Node head = null;

   public SinglyLinkedList() {   }

   public void add(Task *var1*) {

      Node var2 = new Node(var1);

      if (this.head == null) {

         this.head = var2;

      } else {

         Node var3;

         for(var3 = this.head; var3.getNext() != null; var3 = var3.getNext()) {

         }

         var3.setNext(var2); } }

   public Task search(int *var1*) {

      for(Node var2 = this.head; var2 != null; var2 = var2.getNext()) {

         if (var2.getTask().getTaskId() == var1) {

            return var2.getTask();

         }}

return null;

   }

   public void traverse() {

      Node var1 = this.head;

      if (var1 == null) {

         System.out.println("The task list is empty.");

      } else {

         while(var1 != null) {

            System.out.println(var1.getTask());

            var1 = var1.getNext();} }

   }

   public boolean delete(int *var1*) {

      if (this.head == null) {

         return false;

      } else if (this.head.getTask().getTaskId() == var1) {

         this.head = this.head.getNext();

         return true;

      } else {

         for(Node var2 = this.head; var2.getNext() != null; var2 = var2.getNext()) {

            if (var2.getNext().getTask().getTaskId() == var1) {

               var2.setNext(var2.getNext().getNext());

               return true; }

         }

         return false; } }

}

**TaskManagementDemo.class:**

public class TaskManagementDemo {

   public TaskManagementDemo() {}

   public static void main(String[] *var0*) {

      SinglyLinkedList var1 = new SinglyLinkedList();

      var1.add(new Task(1, "Design database schema", "Pending"));

      var1.add(new Task(2, "Develop API endpoints", "In Progress"));

      var1.add(new Task(3, "Implement user authentication", "Pending"));

      var1.add(new Task(4, "Create frontend UI", "Completed"));

      System.out.println("--- All Tasks ---");

      var1.traverse();

      System.out.println();

      System.out.println("--- Searching for task with ID 2 ---");

      Task var2 = var1.search(2);

      if (var2 != null) {

         System.out.println("Found task: " + String.valueOf(var2));

      } else {

         System.out.println("Task not found."); }

      System.out.println();

      System.out.println("--- Searching for task with ID 5 ---");

      var2 = var1.search(5);

      if (var2 != null) {

         System.out.println("Found task: " + String.valueOf(var2));

      } else {

         System.out.println("Task not found."); }

      System.out.println();

      System.out.println("--- Deleting task with ID 3 ---");

      boolean var3 = var1.delete(3);

      if (var3) {

         System.out.println("Task deleted successfully.");

      } else {

         System.out.println("Task not found to delete.");

      }

      System.out.println();

      System.out.println("--- All Tasks after deletion ---");

      var1.traverse();

      System.out.println();

      System.out.println("--- Deleting task with ID 1 ---");

      var3 = var1.delete(1);

      if (var3) {

         System.out.println("Task deleted successfully.");

      } else {

         System.out.println("Task not found to delete."); }

      System.out.println();

      System.out.println("--- All Tasks after deleting head ---");

      var1.traverse();

      System.out.println();

      System.out.println("--- Deleting task with ID 10 ---");

      var3 = var1.delete(10);

      if (var3) {

         System.out.println("Task deleted successfully.");

      } else {

         System.out.println("Task not found to delete."); }

      System.out.println();

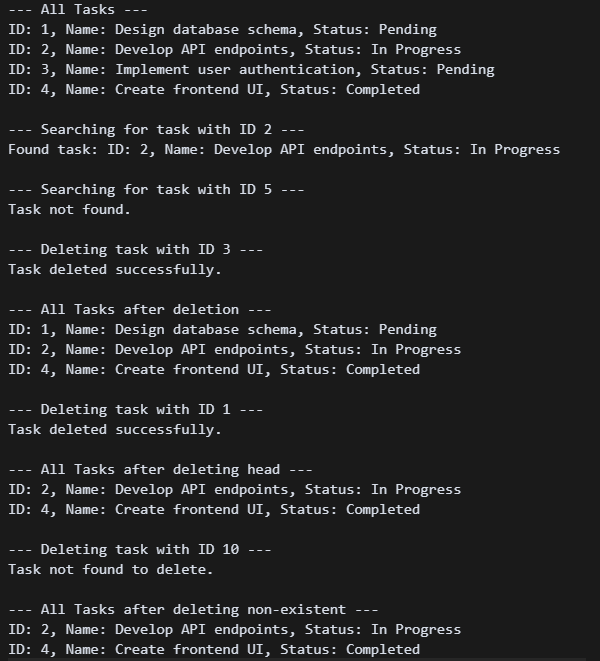
      System.out.println("--- All Tasks after deleting non-existent ---");

      var1.traverse();

   }

}

**OUTPUT:**



**Exercise 6: Library Management System**

**Scenario:**

You are developing a library management system where users can search for books by title or author.

**Steps:**

1. **Understand Search Algorithms:**
   * Explain linear search and binary search algorithms.
2. **Setup:**
   * Create a class **Book** with attributes like **bookId**, **title**, and **author**.
3. **Implementation:**
   * Implement linear search to find books by title.
   * Implement binary search to find books by title (assuming the list is sorted).
4. **Analysis:**
   * Compare the time complexity of linear and binary search.
   * Discuss when to use each algorithm based on the data set size and order.

**SOLUTION:**

**Book.java**

public class Book {

    private int bookId;

    private String title;

    private String author;

    public Book(int *bookId*, String *title*, String *author*) {

        this.bookId = *bookId*;

        this.title = *title*;

        this.author = *author*; }

    public int getBookId() {

        return bookId; }

    public String getTitle() {

        return title; }

    public String getAuthor() {

        return author; }

    public String toString() {

        return "Book(ID: " + bookId + ", Title: '" + title + "', Author: '" + author + "')";

    }

    public boolean equals(Object *obj*) {

        if (this == *obj*) return true;

        if (*obj* == null || getClass() != *obj*.getClass()) return false;

        Book book = (Book) *obj*;

        return title.equalsIgnoreCase(book.title);

    }

    public int hashCode() {

        return title.toLowerCase().hashCode();

    }

}

**SearchAlgorithms.java:**

import java.util.Arrays;

import java.util.Comparator;

public class SearchAlgorithms {

    public static int **linearSearch**(Book[] *books*, String *targetTitle*) {

        System.out.println("=== LINEAR SEARCH ===");

        System.out.println("Searching for: '" + *targetTitle* + "'");

        System.out.println("Algorithm: Check each book sequentially from start to end");

        int comparisons = 0;

        for (int i = 0; i < *books*.length; i++) {

            comparisons++;

            System.out.println("Step " + comparisons + ": Checking book at index " + i +

                             " - '" + *books*[i].getTitle() + "'");

            if (*books*[i].getTitle().equalsIgnoreCase(*targetTitle*)) {

                System.out.println("✓ FOUND! Book found at index " + i);

                System.out.println("Total comparisons: " + comparisons);

                System.out.println("Time Complexity: O(n) - Linear time");

                return i;

            }

        }

        System.out.println("✗ NOT FOUND! Book not in library");

        System.out.println("Total comparisons: " + comparisons);

        System.out.println("Time Complexity: O(n) - Linear time");

        return -1;

    }

    public static int **binarySearch**(Book[] *books*, String *targetTitle*) {

        System.out.println("\n=== BINARY SEARCH ===");

        System.out.println("Searching for: '" + *targetTitle* + "'");

        System.out.println("Algorithm: Divide and conquer on sorted data");

        Book[] sortedBooks = *books*.clone();

        Arrays.sort(sortedBooks, Comparator.comparing(Book::getTitle, String.CASE\_INSENSITIVE\_ORDER));

        System.out.println("Books sorted by title for binary search:");

        for (int i = 0; i < sortedBooks.length; i++) {

            System.out.println("  [" + i + "] " + sortedBooks[i].getTitle());

        }

        int left = 0;

        int right = sortedBooks.length - 1;

        int comparisons = 0;

        while (left <= right) {

            int mid = left + (right - left) / 2;

            comparisons++;

            System.out.println("Step " + comparisons + ": Checking middle element at index " + mid +

                             " - '" + sortedBooks[mid].getTitle() + "'");

            int comparison = *targetTitle*.compareToIgnoreCase(sortedBooks[mid].getTitle());

            if (comparison == 0) {

                System.out.println("✓ FOUND! Book found at index " + mid);

                System.out.println("Total comparisons: " + comparisons);

                System.out.println("Time Complexity: O(log n) - Logarithmic time");

                return mid;

            } else if (comparison < 0) {

                System.out.println("  Target is alphabetically before middle element");

                right = mid - 1;

            } else {

                System.out.println("  Target is alphabetically after middle element");

                left = mid + 1;

            }

        }

        System.out.println("✗ NOT FOUND! Book not in library");

        System.out.println("Total comparisons: " + comparisons);

        System.out.println("Time Complexity: O(log n) - Logarithmic time");

        return -1;

    }

    public static void **compareSearchPerformance**(Book[] *books*, String *targetTitle*) {

        System.out.println("\n" + "=".repeat(60));

        System.out.println("PERFORMANCE COMPARISON");

        System.out.println("=".repeat(60));

        long startTime = System.nanoTime();

        int linearResult = linearSearch(*books*, *targetTitle*);

        long linearTime = System.nanoTime() - startTime;

        startTime = System.nanoTime();

        int binaryResult = binarySearch(*books*, *targetTitle*);

        long binaryTime = System.nanoTime() - startTime;

        System.out.println("\n" + "=".repeat(60));

        System.out.println("PERFORMANCE SUMMARY");

        System.out.println("=".repeat(60));

        System.out.println("Linear Search Time: " + linearTime + " nanoseconds");

        System.out.println("Binary Search Time: " + binaryTime + " nanoseconds");

        System.out.println("Speed Improvement: " + String.format("%.2f", (double)linearTime/binaryTime) + "x faster");

        System.out.println("\n" + "=".repeat(60));

        System.out.println("ALGORITHM ANALYSIS");

        System.out.println("=".repeat(60));

        System.out.println("LINEAR SEARCH:");

        System.out.println("  • Time Complexity: O(n)");

        System.out.println("  • Space Complexity: O(1)");

        System.out.println("  • Best Case: O(1) - Target is first element");

        System.out.println("  • Worst Case: O(n) - Target is last element or not found");

        System.out.println("  • Use when: Small datasets, unsorted data, infrequent searches");

        System.out.println("\nBINARY SEARCH:");

        System.out.println("  • Time Complexity: O(log n)");

        System.out.println("  • Space Complexity: O(1)");

        System.out.println("  • Best Case: O(1) - Target is middle element");

        System.out.println("  • Worst Case: O(log n) - Target is not found");

        System.out.println("  • Use when: Large datasets, sorted data, frequent searches");

        System.out.println("  • Requirement: Data must be sorted");

    }

}

**LibraryManagementSystem.java:**

import java.util.Scanner;

public class LibraryManagementSystem {

        public static void **main**(String[] *args*) {

        System.out.println("=".repeat(70));

        System.out.println("           LIBRARY MANAGEMENT SYSTEM");

        System.out.println("           Search Algorithms Demo");

        System.out.println("=".repeat(70));

        Book[] library = createSampleLibrary();

        displayLibrary(library);

        Scanner scanner = new Scanner(System.in);

        while (true) {

            System.out.println("\n" + "=".repeat(70));

            System.out.println("SEARCH OPTIONS:");

            System.out.println("1. Linear Search (Sequential search through unsorted data)");

            System.out.println("2. Binary Search (Divide and conquer on sorted data)");

            System.out.println("3. Performance Comparison (Compare both algorithms)");

            System.out.println("4. Exit");

            System.out.println("=".repeat(70));

            System.out.print("Enter your choice (1-4): ");

            int choice = scanner.nextInt();

            scanner.nextLine();

            switch (choice) {

                case 1:

                    performLinearSearch(library, scanner);

                    break;

                case 2:

                    performBinarySearch(library, scanner);

                    break;

                case 3:

                    performPerformanceComparison(library, scanner);

                    break;

                case 4:

                    System.out.println("Thank you for using the Library Management System!");

                    scanner.close();

                    return;

                default:

                    System.out.println("Invalid choice. Please enter 1-4.");

            }

        }

    }

    private static Book[] **createSampleLibrary**() {

        return new Book[] {

            new Book(1, "The Great Gatsby", "F. Scott Fitzgerald"),

            new Book(2, "To Kill a Mockingbird", "Harper Lee"),

            new Book(3, "1984", "George Orwell"),

            new Book(4, "Pride and Prejudice", "Jane Austen"),

            new Book(5, "The Catcher in the Rye", "J.D. Salinger"),

            new Book(6, "Lord of the Flies", "William Golding"),

            new Book(7, "Animal Farm", "George Orwell"),

            new Book(8, "The Hobbit", "J.R.R. Tolkien"),

            new Book(9, "Brave New World", "Aldous Huxley"),

            new Book(10, "Fahrenheit 451", "Ray Bradbury")

        };

    }

    private static void **displayLibrary**(Book[] *library*) {

        System.out.println("\n📚 LIBRARY CONTENTS:");

        System.out.println("-".repeat(50));

        for (int i = 0; i < *library*.length; i++) {

            System.out.printf("[%2d] %s\n", i, *library*[i]);

        }

        System.out.println("-".repeat(50));

        System.out.println("Total books: " + *library*.length);

    }

    private static void **performLinearSearch**(Book[] *library*, Scanner *scanner*) {

        System.out.print("\nEnter book title to search (Linear Search): ");

        String targetTitle = *scanner*.nextLine();

        if (targetTitle.trim().isEmpty()) {

            System.out.println("Please enter a valid book title.");

            return;

        }

        SearchAlgorithms.linearSearch(*library*, targetTitle);

    }

    private static void **performBinarySearch**(Book[] *library*, Scanner *scanner*) {

        System.out.print("\nEnter book title to search (Binary Search): ");

        String targetTitle = *scanner*.nextLine();

        if (targetTitle.trim().isEmpty()) {

            System.out.println("Please enter a valid book title.");

            return;

        }

        SearchAlgorithms.binarySearch(*library*, targetTitle);

    }

    private static void **performPerformanceComparison**(Book[] *library*, Scanner *scanner*) {

        System.out.print("\nEnter book title for performance comparison: ");

        String targetTitle = *scanner*.nextLine();

        if (targetTitle.trim().isEmpty()) {

            System.out.println("Please enter a valid book title.");

            return;

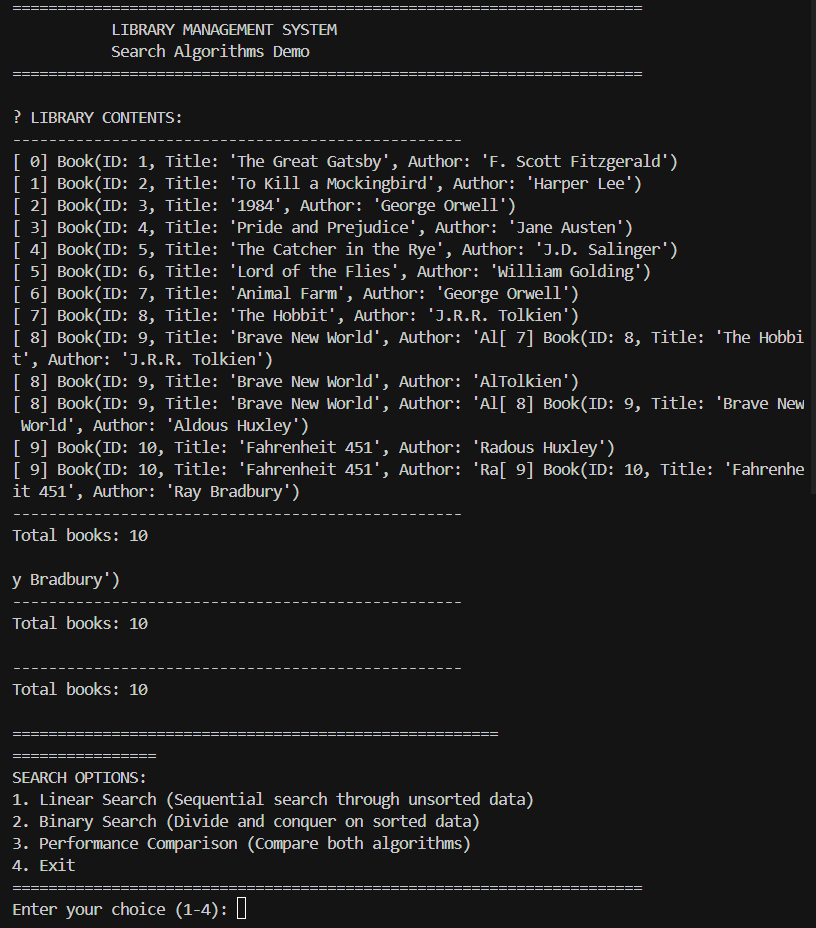
        }

        SearchAlgorithms.compareSearchPerformance(*library*, targetTitle);

    }

}

**OUTPUT:**

****

**Exercise 7: Financial Forecasting**

**Scenario:**

You are developing a financial forecasting tool that predicts future values based on past data.

**Steps:**

1. **Understand Recursive Algorithms:**
   * Explain the concept of recursion and how it can simplify certain problems.
2. **Setup:**
   * Create a method to calculate the future value using a recursive approach.
3. **Implementation:**
   * Implement a recursive algorithm to predict future values based on past growth rates.
4. **Analysis:**
   * Discuss the time complexity of your recursive algorithm.
   * Explain how to optimize the recursive solution to avoid excessive computation.

**SOLUTION:**

**FinancialForecasting.java:**

public class FinancialForecasting {

    public static double **calculateFutureValue**(double *presentValue*, double *growthRate*, int *years*) {

      if (*years* == 0) {

            return *presentValue*;

        }

         return calculateFutureValue(*presentValue* \* (1 + *growthRate*), *growthRate*, *years* - 1);

    }    public static void **main**(String[] *args*) {

        double presentValue = 1000.0;

        double growthRate = 0.05; *// 5% growth rate*

        int years = 10;

        double futureValue = calculateFutureValue(presentValue, growthRate, years);

        System.out.println("Future Value (Recursive): " + String.format("%.2f", futureValue));

        System.out.println("\n--- Optimized Calculation ---");

        double futureValueMemoized = calculateFutureValueMemoizedWrapper(presentValue, growthRate, years);

        System.out.println("Future Value (Memoized): " + String.format("%.2f", futureValueMemoized));

    }  public static double **calculateFutureValueMemoizedWrapper**(double *presentValue*, double *growthRate*, int *years*) {

        Double[] memo = new Double[*years* + 1];

        return calculateFutureValueMemoized(*presentValue*, *growthRate*, *years*, memo);

    }  public static double **calculateFutureValueMemoized**(double *presentValue*, double *growthRate*, int *years*, Double[] *memo*) {

        if (*years* == 0) {

            return *presentValue*;

        }  if (*memo*[*years*] != null) {

            return *memo*[*years*];

        }  double previousValue = calculateFutureValueMemoized(*presentValue*, *growthRate*, *years* - 1, *memo*);

*memo*[*years*] = previousValue \* (1 + *growthRate*);

        return *memo*[*years*];

    }

}

**OUTPUT:**

