**Data Structures & Algorithms**

**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.

**Code:**

import java.util.HashMap;

import java.util.Map;

import java.util.Scanner;

class Product {

int productId;

String productName;

int quantity;

double price;

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

this.productId = productId;

this.productName = productName;

this.quantity = quantity;

this.price = price;

}

public void display() {

System.***out***.println("ID: " + productId + ", Name: " + productName +

", Quantity: " + quantity + ", Price: " + price);

}

}

public class InventorySystem {

static Map<Integer, Product> *inventory* = new HashMap<>();

public static void addProduct(int id, String name, int qty, double price) {

if (*inventory*.containsKey(id)) {

System.***out***.println("Product ID already exists. Use update instead.");

} else {

*inventory*.put(id, new Product(id, name, qty, price));

System.***out***.println("Product added successfully.");

}

}

public static void updateProduct(int id, String name, int qty, double price) {

if (*inventory*.containsKey(id)) {

*inventory*.put(id, new Product(id, name, qty, price));

System.***out***.println("Product updated successfully.");

} else {

System.***out***.println("Product ID not found.");

}

}

public static void deleteProduct(int id) {

if (*inventory*.remove(id) != null) {

System.***out***.println("Product deleted successfully.");

} else {

System.***out***.println("Product ID not found.");

}

}

public static void displayInventory() {

if (*inventory*.isEmpty()) {

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

} else {

for (Product p : *inventory*.values()) {

p.display();

}

}

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.***in***);

int choice;

do {

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

System.***out***.println("1. Add Product");

System.***out***.println("2. Update Product");

System.***out***.println("3. Delete Product");

System.***out***.println("4. View Inventory");

System.***out***.println("5. Exit");

System.***out***.print("Enter choice: ");

choice = sc.nextInt();

switch (choice) {

case 1:

System.***out***.print("Enter ID, Name, Quantity, Price: ");

*addProduct*(sc.nextInt(), sc.next(), sc.nextInt(), sc.nextDouble());

break;

case 2:

System.***out***.print("Enter ID, New Name, New Quantity, New Price:");

*updateProduct*(sc.nextInt(), sc.next(), sc.nextInt(), sc.nextDouble());

break;

case 3:

System.***out***.print("Enter Product ID to delete: ");

*deleteProduct*(sc.nextInt());

break;

case 4:

*displayInventory*();

break;

case 5:

System.***out***.println("Exiting.");

break;

default:

System.***out***.println("Invalid choice.");

}

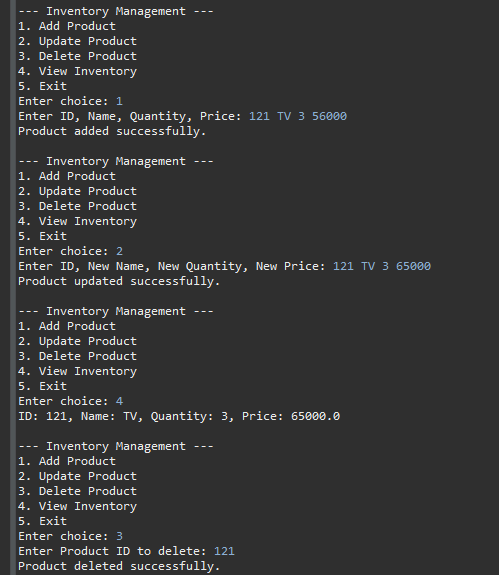
} while (choice != 5);

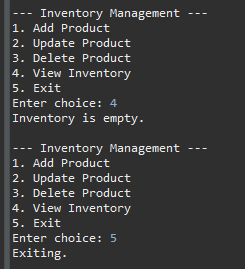
sc.close();

}

}

**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.

**Code:**

import java.util.\*;

class Product {

int productId;

String productName;

String category;

public Product(int productId, String productName, String category) {

this.productId = productId;

this.productName = productName;

this.category = category;

}

public void display() {

System.***out***.println("ID: " + productId + ", Name: " + productName + ", Category: " + category);

}

}

public class ProductSearch {

// Linear Search

public static Product linearSearch(Product[] products, String targetName) {

for (Product p : products) {

if (p.productName.equalsIgnoreCase(targetName)) {

return p;

}

}

return null;

}

// Binary Search (requires sorted array by productName)

public static Product binarySearch(Product[] products, String targetName) {

int low = 0, high = products.length - 1;

while (low <= high) {

int mid = (low + high) / 2;

int cmp = products[mid].productName.compareToIgnoreCase(targetName);

if (cmp == 0) {

return products[mid];

} else if (cmp < 0) {

low = mid + 1;

} else {

high = mid - 1;

}

}

return null;

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.***in***);

System.***out***.print("Enter number of products: ");

int n = Integer.*parseInt*(sc.nextLine());

Product[] products = new Product[n];

// Input products

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

System.***out***.print("Enter product " + (i + 1) + " (productId, productName, category): ");

String[] input = sc.nextLine().split(",");

int id = Integer.*parseInt*(input[0].trim());

String name = input[1].trim();

String category = input[2].trim();

products[i] = new Product(id, name, category);

}

// Search input

System.***out***.print("\nEnter product name to search: ");

String searchName = sc.nextLine().trim();

// Perform Linear Search

System.***out***.println("\nLinear Search Result:");

Product linearResult = *linearSearch*(products, searchName);

if (linearResult != null) {

linearResult.display();

} else {

System.***out***.println("Product not found.");

}

// Sort for Binary Search

Arrays.*sort*(products, Comparator.*comparing*(p ->p.productName.toLowerCase()));

// Perform Binary Search

System.***out***.println("\nBinary Search Result:");

Product binaryResult = *binarySearch*(products, searchName);

if (binaryResult != null) {

binaryResult.display();

} else {

System.***out***.println("Product not found.");

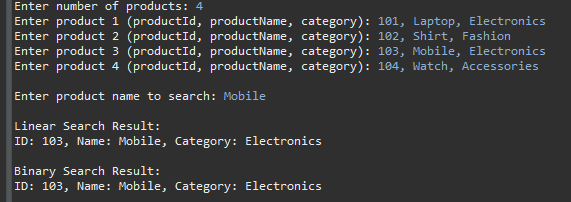
}

sc.close();

}

}

**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.

**Code:**

import java.util.\*;

class Order {

int orderId;

String customerName;

double totalPrice;

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

this.orderId = orderId;

this.customerName = customerName;

this.totalPrice = totalPrice;

}

public void display() {

System.***out***.println("Order ID: " + orderId + ", Customer: " + customerName + ", Total Price: " + totalPrice);

}

}

public class OrderSorter {

// Bubble Sort

public static void bubbleSort(Order[] orders) {

int n = orders.length;

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

boolean swapped = false;

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

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

Order temp = orders[j];

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

orders[j + 1] = temp;

swapped = true;

}

}

if (!swapped) break; // Optimization

}

}

// Quick Sort

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);

}

}

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

double pivot = orders[high].totalPrice;

int i = low - 1;

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

if (orders[j].totalPrice <= 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;

}

// Display orders

public static void displayOrders(Order[] orders) {

for (Order o : orders) {

o.display();

}

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.***in***);

System.***out***.print("Enter number of orders: ");

int n = Integer.*parseInt*(sc.nextLine());

Order[] orders = new Order[n];

Order[] copyForQuickSort = new Order[n];

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

System.***out***.print("Enter order " + (i + 1) + " (orderId, customerName, totalPrice): ");

String[] input = sc.nextLine().split(",");

int id = Integer.*parseInt*(input[0].trim());

String name = input[1].trim();

double price = Double.*parseDouble*(input[2].trim());

orders[i] = new Order(id, name, price);

copyForQuickSort[i] = new Order(id, name, price); // copy for quick sort

}

System.***out***.println("\nSorted by Bubble Sort (Ascending by totalPrice):");

*bubbleSort*(orders);

*displayOrders*(orders);

System.***out***.println("\nSorted by Quick Sort (Ascending by totalPrice):");

*quickSort*(copyForQuickSort, 0, n - 1);

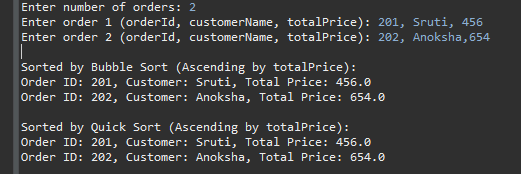
*displayOrders*(copyForQuickSort);

sc.close();

}

}

**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.

**Code:**

import java.util.\*;

class Employee {

int employeeId;

String name;

String position;

double salary;

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

this.employeeId = employeeId;

this.name = name;

this.position = position;

this.salary = salary;

}

public void display() {

System.***out***.println("ID: " + employeeId + ", Name: " + name + ", Position: " + position + ", Salary: " + salary);

}

}

public class EmployeeSystem {

static Employee[] *employees*;

static int *count* = 0;

public static void addEmployee(Scanner sc) {

if (*count* >= *employees*.length) {

System.***out***.println("Employee list is full.");

return;

}

System.***out***.print("Enter employee (employeeId, name, position, salary): ");

String[] input = sc.nextLine().split(",");

int id = Integer.*parseInt*(input[0].trim());

String name = input[1].trim();

String position = input[2].trim();

double salary = Double.*parseDouble*(input[3].trim());

*employees*[*count*++] = new Employee(id, name, position, salary);

System.***out***.println("Employee added.");

}

public static void searchEmployee(Scanner sc) {

System.***out***.print("Enter employee ID to search: ");

int id = Integer.*parseInt*(sc.nextLine());

for (int i = 0; i < *count*; i++) {

if (*employees*[i].employeeId == id) {

*employees*[i].display();

return;

}

}

System.***out***.println("Employee not found.");

}

public static void traverseEmployees() {

if (*count* == 0) {

System.***out***.println("No employees to display.");

return;

}

for (int i = 0; i < *count*; i++) {

*employees*[i].display();

}

}

public static void deleteEmployee(Scanner sc) {

System.***out***.print("Enter employee ID to delete: ");

int id = Integer.*parseInt*(sc.nextLine());

for (int i = 0; i < *count*; i++) {

if (*employees*[i].employeeId == id) {

for (int j = i; j < *count* - 1; j++) {

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

}

*employees*[--*count*] = null;

System.***out***.println("Employee deleted.");

return;

}

}

System.***out***.println("Employee not found.");

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.***in***);

System.***out***.print("Enter maximum number of employees: ");

int max = Integer.*parseInt*(sc.nextLine());

*employees* = new Employee[max];

int choice;

do {

System.***out***.println("\n--- Employee Management ---");

System.***out***.println("1. Add Employee");

System.***out***.println("2. Search Employee");

System.***out***.println("3. Display All Employees");

System.***out***.println("4. Delete Employee");

System.***out***.println("5. Exit");

System.***out***.print("Enter choice: ");

choice = Integer.*parseInt*(sc.nextLine());

switch (choice) {

case 1: *addEmployee*(sc); break;

case 2: *searchEmployee*(sc); break;

case 3: *traverseEmployees*(); break;

case 4: *deleteEmployee*(sc); break;

case 5: System.***out***.println("Exiting."); break;

default: System.***out***.println("Invalid choice.");

}

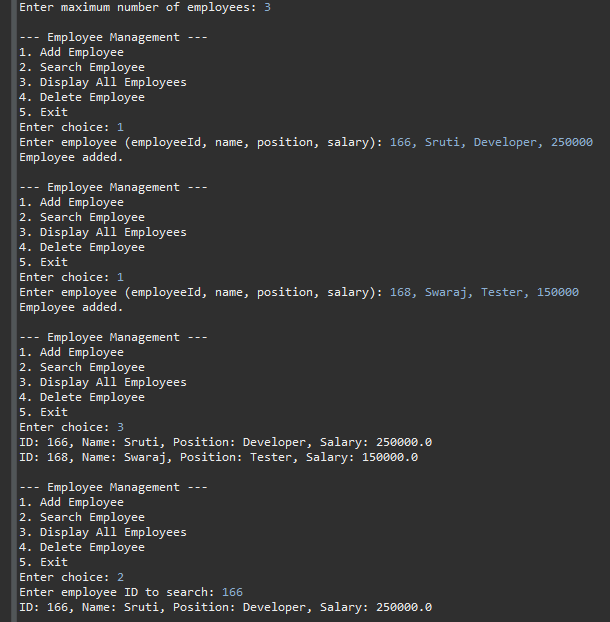
} while (choice != 5);

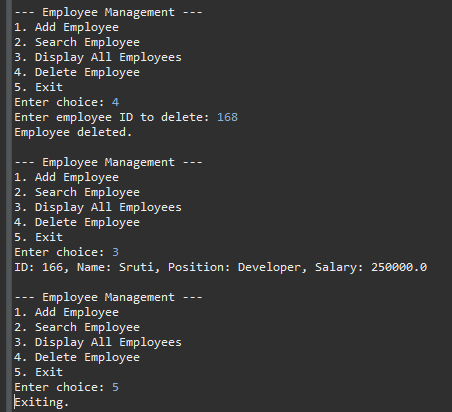
sc.close();

}

}

**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.

**Code:**

import java.util.Scanner;

class Task {

int taskId;

String taskName;

String status;

Task next;

public Task(int taskId, String taskName, String status) {

this.taskId = taskId;

this.taskName = taskName;

this.status = status;

this.next = null;

}

public void display() {

System.***out***.println("Task ID: " + taskId + ", Name: " + taskName + ", Status: " + status);

}

}

public class TaskManager {

static Task *head* = null;

public static void addTask(Scanner sc) {

System.***out***.print("Enter task (taskId, taskName, status): ");

String[] input = sc.nextLine().split(",");

int id = Integer.*parseInt*(input[0].trim());

String name = input[1].trim();

String status = input[2].trim();

Task newTask = new Task(id, name, status);

if (*head* == null) {

*head* = newTask;

} else {

Task temp = *head*;

while (temp.next != null) {

temp = temp.next;

}

temp.next = newTask;

}

System.***out***.println("Task added.");

}

public static void searchTask(Scanner sc) {

System.***out***.print("Enter task ID to search: ");

int id = Integer.*parseInt*(sc.nextLine());

Task temp = *head*;

while (temp != null) {

if (temp.taskId == id) {

temp.display();

return;

}

temp = temp.next;

}

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

}

public static void traverseTasks() {

if (*head* == null) {

System.***out***.println("No tasks to display.");

return;

}

Task temp = *head*;

while (temp != null) {

temp.display();

temp = temp.next;

}

}

public static void deleteTask(Scanner sc) {

System.***out***.print("Enter task ID to delete: ");

int id = Integer.*parseInt*(sc.nextLine());

if (*head* == null) {

System.***out***.println("Task list is empty.");

return;

}

if (*head*.taskId == id) {

*head* = *head*.next;

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

return;

}

Task current = *head*;

Task prev = null;

while (current != null && current.taskId != id) {

prev = current;

current = current.next;

}

if (current == null) {

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

} else {

prev.next = current.next;

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

}

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.***in***);

int choice;

do {

System.***out***.println("\n--- Task Management ---");

System.***out***.println("1. Add Task");

System.***out***.println("2. Search Task");

System.***out***.println("3. Display All Tasks");

System.***out***.println("4. Delete Task");

System.***out***.println("5. Exit");

System.***out***.print("Enter choice: ");

choice = Integer.*parseInt*(sc.nextLine());

switch (choice) {

case 1: *addTask*(sc); break;

case 2: *searchTask*(sc); break;

case 3: *traverseTasks*(); break;

case 4: *deleteTask*(sc); break;

case 5: System.***out***.println("Exiting."); break;

default: System.***out***.println("Invalid choice.");

}

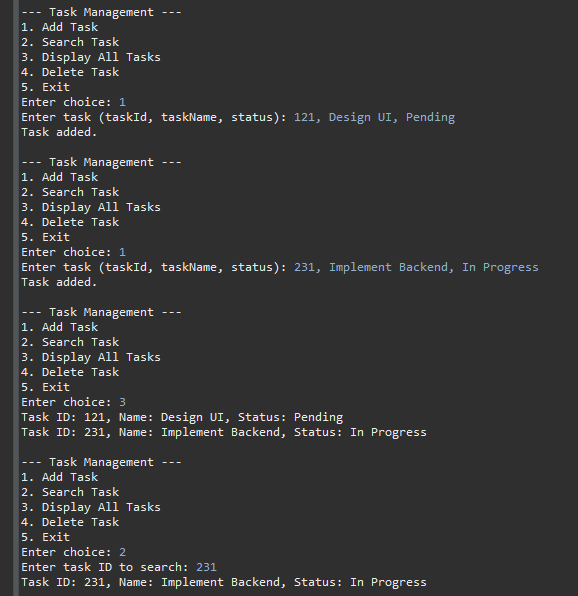
} while (choice != 5);

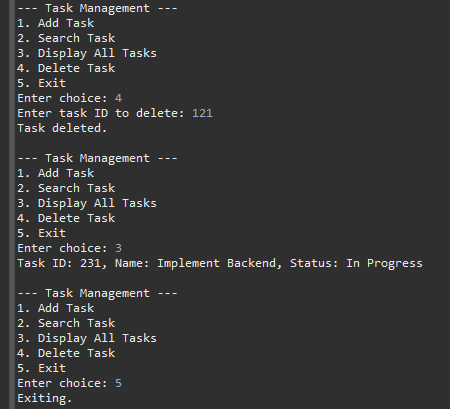
sc.close();

}

}

**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.

**Code:**

import java.util.\*;

class Book {

int bookId;

String title;

String author;

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

this.bookId = bookId;

this.title = title;

this.author = author;

}

public void display() {

System.***out***.println("Book ID: " + bookId + ", Title: " + title + ", Author: " + author);

}

}

public class LibraryManager {

// Linear Search by Title

public static Book linearSearch(Book[] books, String title) {

for (Book book : books) {

if (book.title.equalsIgnoreCase(title)) {

return book;

}

}

return null;

}

// Binary Search by Title (Assumes sorted list)

public static Book binarySearch(Book[] books, String title) {

int low = 0, high = books.length - 1;

while (low <= high) {

int mid = (low + high) / 2;

int cmp = books[mid].title.compareToIgnoreCase(title);

if (cmp == 0) {

return books[mid];

} else if (cmp < 0) {

low = mid + 1;

} else {

high = mid - 1;

}

}

return null;

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.***in***);

System.***out***.print("Enter number of books: ");

int n = Integer.*parseInt*(sc.nextLine());

Book[] books = new Book[n];

Book[] booksSorted = new Book[n];

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

System.***out***.print("Enter book " + (i + 1) + " (bookId, title, author): ");

String[] input = sc.nextLine().split(",");

int id = Integer.*parseInt*(input[0].trim());

String title = input[1].trim();

String author = input[2].trim();

books[i] = new Book(id, title, author);

booksSorted[i] = new Book(id, title, author); // for binary search

}

// Sort booksSorted by title for binary search

Arrays.*sort*(booksSorted, Comparator.*comparing*(b -> b.title.toLowerCase()));

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

String searchTitle = sc.nextLine().trim();

System.***out***.println("\nLinear Search Result:");

Book result1 = *linearSearch*(books, searchTitle);

if (result1 != null) result1.display();

else System.***out***.println("Book not found.");

System.***out***.println("\nBinary Search Result:");

Book result2 = *binarySearch*(booksSorted, searchTitle);

if (result2 != null) result2.display();

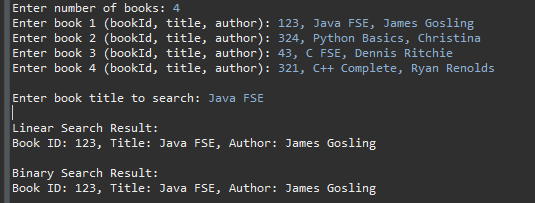
else System.***out***.println("Book not found.");

sc.close();

}

}

**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.

**Code:**

import java.util.\*;

public class ForecastCalculator {

// Memoization cache to avoid repeated computations

static Map<Integer, Double> *cache* = new HashMap<>();

// Recursive method with memoization

public static double forecast(int years, double initialValue, double growthRate) {

if (years == 0) return initialValue;

if (*cache*.containsKey(years)) return *cache*.get(years);

double previous = *forecast*(years - 1, initialValue, growthRate);

double current = previous \* (1 + growthRate);

*cache*.put(years, current);

return current;

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.***in***);

System.***out***.print("Enter initial value, growth rate (as decimal), number of years: ");

String[] input = sc.nextLine().split(",");

double initialValue = Double.*parseDouble*(input[0].trim());

double growthRate = Double.*parseDouble*(input[1].trim());

int years = Integer.*parseInt*(input[2].trim());

double futureValue = *forecast*(years, initialValue, growthRate);

System.***out***.printf("Forecasted value after %d years: %.2f\n", years, futureValue);

sc.close();

}

}

**Output:**

****