1. **Inventory Management System**

Hashmap is best suitable since it can access fast and time complexity is O(1) and is efficient for large inventories since performance is steady

**CODE:**

import java.util.\*;

class Product {

private int productId;

private String productName;

private int quantity;

private 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 int getProductId() { return productId; }

public String getProductName() { return productName; }

public int getQuantity() { return quantity; }

public double getPrice() { return price; }

public void setProductName(String name) { this.productName = name; }

public void setQuantity(int quantity) { this.quantity = quantity; }

public void setPrice(double price) { this.price = price; }

public String toString() {

return "ID: " + productId + ", Name: " + productName + ", Quantity: " + quantity + ", Price: ₹" + price;

}

}

interface InventoryService {

void addProduct(Product product);

void updateProduct(int productId, int quantity, double price);

void deleteProduct(int productId);

void displayInventory();

}

class InventoryManager implements InventoryService {

private final Map<Integer, Product> inventory = new HashMap<>();

public void addProduct(Product product) {

if (inventory.containsKey(product.getProductId())) {

System.out.println("Product already exists.");

} else {

inventory.put(product.getProductId(), product);

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

}

}

public void updateProduct(int productId, int quantity, double price) {

Product product = inventory.get(productId);

if (product != null) {

product.setQuantity(quantity);

product.setPrice(price);

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

} else {

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

}

}

public void deleteProduct(int productId) {

if (inventory.remove(productId) != null) {

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

} else {

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

}

}

public void displayInventory() {

if (inventory.isEmpty()) {

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

} else {

System.out.println("Current Inventory:");

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

System.out.println(product);

}

}

}

}

public class Main {

public static void main(String[] args) {

InventoryService manager = new InventoryManager();

Scanner scanner = new Scanner(System.in);

int choice;

do {

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

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 your choice: ");

choice = scanner.nextInt();

switch (choice) {

case 1 -> {

System.out.print("Enter ID: ");

int id = scanner.nextInt();

scanner.nextLine();

System.out.print("Enter Name: ");

String name = scanner.nextLine();

System.out.print("Enter Quantity: ");

int qty = scanner.nextInt();

System.out.print("Enter Price: ");

double price = scanner.nextDouble();

manager.addProduct(new Product(id, name, qty, price));

}

case 2 -> {

System.out.print("Enter ID to update: ");

int id = scanner.nextInt();

System.out.print("Enter New Quantity: ");

int qty = scanner.nextInt();

System.out.print("Enter New Price: ");

double price = scanner.nextDouble();

manager.updateProduct(id, qty, price);

}

case 3 -> {

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

int id = scanner.nextInt();

manager.deleteProduct(id);

}

case 4 -> manager.displayInventory();

case 5 -> System.out.println("Exiting...");

default -> System.out.println("Invalid choice.");

}

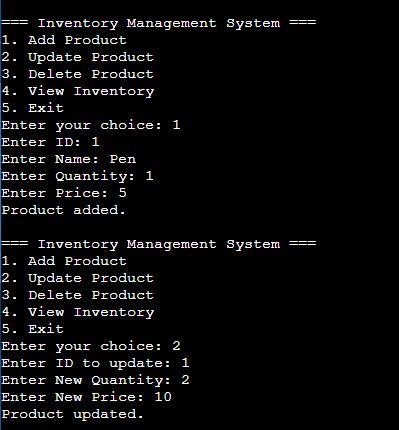
} while (choice != 5);

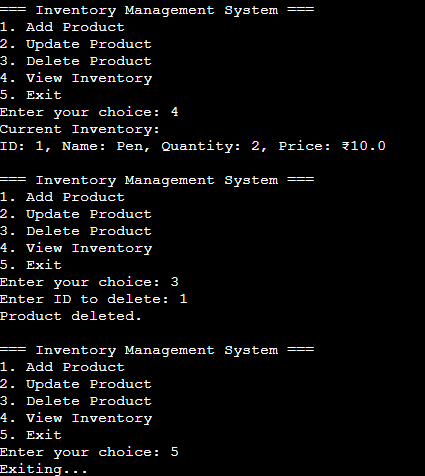
scanner.close();

}

}

**OUTPUT:**

****

****

1. **E-commerce Platform Search Function**

Big O notation is useful in analysing performance of algorithm and helps in comparison

| **Search Type** | **Best Case** | **Average Case** | **Worst Case** |
| --- | --- | --- | --- |
| **Linear Search** | O(1) | O(n) | O(n) |
| **Binary Search** | O(1) | O(log n) | O(log n) |

**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 class Main {

public static Product linearSearch(Product[] products, int id) {

for (Product product : products) {

if (product.productId == id) {

return product;

}

}

return null;

}

public static Product binarySearch(Product[] products, int id) {

int left = 0;

int right = products.length - 1;

while (left <= right) {

int mid = (left + right) / 2;

if (products[mid].productId == id) {

return products[mid];

} else if (products[mid].productId < id) {

left = mid + 1;

} else {

right = mid - 1;

}

}

return null;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

Product[] unsortedProducts = {

new Product(1020, "Cooking Oil", "Grocery"),

new Product(1005, "T-shirt", "Clothing"),

new Product(1012, "Mixer", "Appliances"),

new Product(1001, "Basmati Rice", "Grocery")

};

Product[] sortedProducts = {

new Product(1001, "Basmati Rice", "Grocery"),

new Product(1005, "T-shirt", "Clothing"),

new Product(1012, "Mixer", "Appliances"),

new Product(1020, "Cooking Oil", "Grocery")

};

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

int id = scanner.nextInt();

Product linearResult = linearSearch(unsortedProducts, id);

System.out.println("Linear Search Result: " +

(linearResult != null ? linearResult.productName : "Product not found"));

Product binaryResult = binarySearch(sortedProducts, id);

System.out.println("Binary Search Result: " +

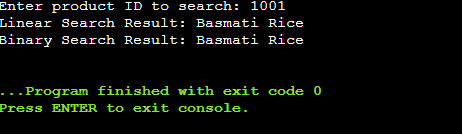
(binaryResult != null ? binaryResult.productName : "Product not found"));

}

}

**//Binary search is faster and efficient way**

**OUTPUT:**

****

1. **Sorting Customer Orders**

**Explanation:**

**Bubble Sort:**Repeatedly compares adjacent elements and swaps them if they are in the wrong order.After each pass, the largest element to the end.

**Insertion Sort:**Builds the final sorted array one item at a time by inserting items into the correct position.

**Merge Sort:**Divides the array into halves, recursively sorts them, and merges the sorted halves back.

**Quick Sort:**Selects a pivot element, partitions the array so that elements less than pivot are on the left, greater than pivot on the right, then recursively sorts both halves.

| **Sorting Algorithm** | **Best Case** | **Average Case** | **Worst Case** |
| --- | --- | --- | --- |
| BUBBLE SORT | O(n) | O(n²) | O(n²) |
| INSERTION SORT | O(n) | O(n²) | O(n²) |
| QUICK SORT | O(n log n) | O(n log n) | O(n²) |
| MERGE SORT | O(n log n) | O(n log n) | O(n log n) |

**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: " + totalPrice);

}

}

public class Main {

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

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);

}

}

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;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

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

int n = scanner.nextInt();

scanner.nextLine();

Order[] orders1 = new Order[n];

Order[] orders2 = new Order[n];

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

System.out.print("Enter Order ID: ");

int id = scanner.nextInt();

scanner.nextLine();

System.out.print("Enter Customer Name: ");

String name = scanner.nextLine();

System.out.print("Enter Total Price: ");

double price = scanner.nextDouble();

scanner.nextLine();

orders1[i] = new Order(id, name, price);

orders2[i] = new Order(id, name, price);

}

System.out.println("\nBefore Bubble Sort:");

for (Order o : orders1) o.display();

bubbleSort(orders1);

System.out.println("\nAfter Bubble Sort:");

for (Order o : orders1) o.display();

System.out.println("\nBefore Quick Sort:");

for (Order o : orders2) o.display();

quickSort(orders2, 0, orders2.length - 1);

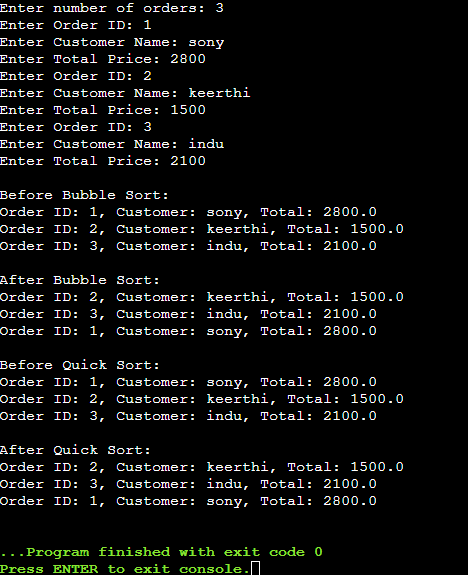
System.out.println("\nAfter Quick Sort:");

for (Order o : orders2) o.display();

}

}

**OUTPUT:**

****

//QUICK SORT IS PREFERRED OVER BUBBLE SORT SINCE IT HAS EFFICIENT STORAGE CAPACITY AND HIGH SPEED

1. **Employee Management System:**

**Explanation:**

Arrays are stored in **contiguous blocks of memory**.They are used for fast access and for easy implementation

| **Operation** | **Time Complexity** |
| --- | --- |
| ADD | O(1) |
| SEARCH | O(n) |
| TRAVERSE | O(n) |
| DELETE | O(n) |

LIMITATIONS AND WHEN TO USE:

Fixed size, inefficient insert/delete, possible wasted memory and it is used when no of elements are already known

**CODE:**

import java.util.Scanner;

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 Main {

static Employee[] employees = new Employee[100];

static int count = 0;

public static void addEmployee(Employee e) {

if (count < employees.length) {

employees[count++] = e;

}

}

public static Employee searchEmployee(int id) {

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

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

return employees[i];

}

}

return null;

}

public static void traverseEmployees() {

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

employees[i].display();

}

}

public static void deleteEmployee(int id) {

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;

break;

}

}

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

while (true) {

System.out.println("\n1. Add Employee\n2. View All Employees\n3. Search Employee\n4. Delete Employee\n5. Exit");

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

int choice = sc.nextInt();

sc.nextLine();

if (choice == 1) {

System.out.print("Enter ID: ");

int id = sc.nextInt();

sc.nextLine();

System.out.print("Enter Name: ");

String name = sc.nextLine();

System.out.print("Enter Position: ");

String position = sc.nextLine();

System.out.print("Enter Salary: ");

double salary = sc.nextDouble();

sc.nextLine();

addEmployee(new Employee(id, name, position, salary));

} else if (choice == 2) {

if (count == 0) System.out.println("No employees to display.");

else traverseEmployees();

} else if (choice == 3) {

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

int id = sc.nextInt();

Employee e = searchEmployee(id);

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

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

} else if (choice == 4) {

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

int id = sc.nextInt();

deleteEmployee(id);

} else if (choice == 5) {

break;

} else {

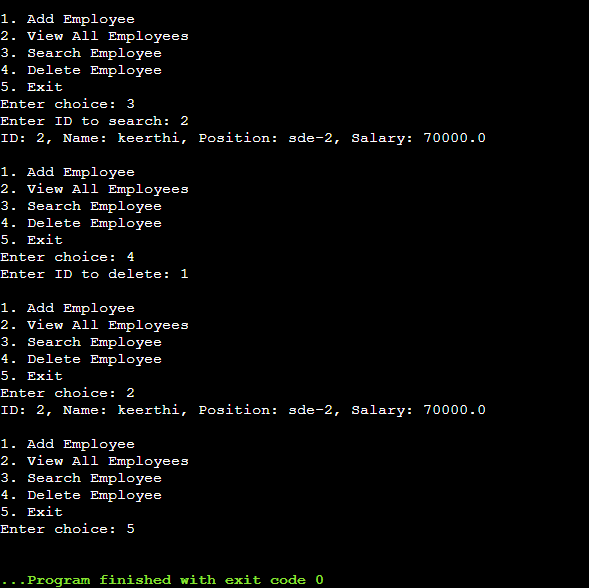
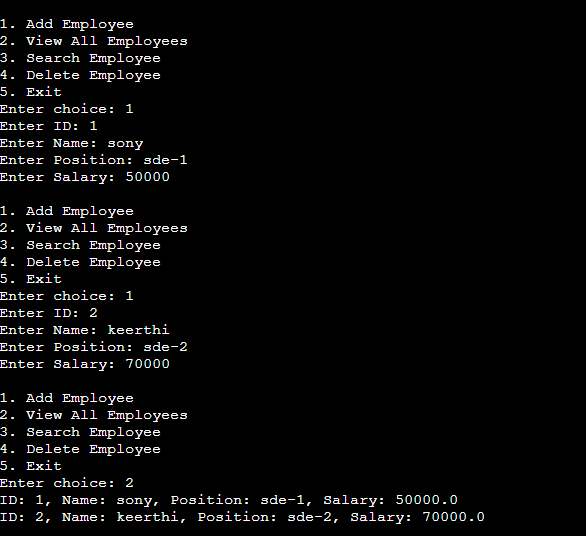
System.out.println("Invalid choice");

}

}

}

}

**OUTPUT:**

1. **Task Management System**

**Explanation:**

Single linked List:Traversal is unidirectional

Double Linked List:Traversal is Bidirectional

| **OPERATION** | **TIME COMPLEXITY** |
| --- | --- |
| ADD | O(n) |
| SEARCH | O(n) |
| TRAVERSE | O(n) |
| DELETE | O(n) |

Linked Lists are used over arrays for efficient insertion and deletion

**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("ID: " + taskId + ", Name: " + taskName + ", Status: " + status);

}

}

public class Main {

static Task head = null;

public static void addTask(int id, String name, String status) {

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;

}

}

public static void traverseTasks() {

Task temp = head;

if (temp == null) {

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

return;

}

while (temp != null) {

temp.display();

temp = temp.next;

}

}

public static void searchTask(int id) {

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 deleteTask(int id) {

if (head == null) {

System.out.println("No tasks to delete");

return;

}

if (head.taskId == id) {

head = head.next;

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

return;

}

Task temp = head;

while (temp.next != null) {

if (temp.next.taskId == id) {

temp.next = temp.next.next;

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

return;

}

temp = temp.next;

}

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

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

while (true) {

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

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

System.out.println("2. View All Tasks");

System.out.println("3. Search Task");

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

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

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

int choice = sc.nextInt();

sc.nextLine();

if (choice == 1) {

System.out.print("Enter Task ID: ");

int id = sc.nextInt();

sc.nextLine();

System.out.print("Enter Task Name: ");

String name = sc.nextLine();

System.out.print("Enter Task Status: ");

String status = sc.nextLine();

addTask(id, name, status);

} else if (choice == 2) {

traverseTasks();

} else if (choice == 3) {

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

int id = sc.nextInt();

searchTask(id);

} else if (choice == 4) {

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

int id = sc.nextInt();

deleteTask(id);

} else if (choice == 5) {

System.out.println("Exiting Task Manager...");

break;

} else {

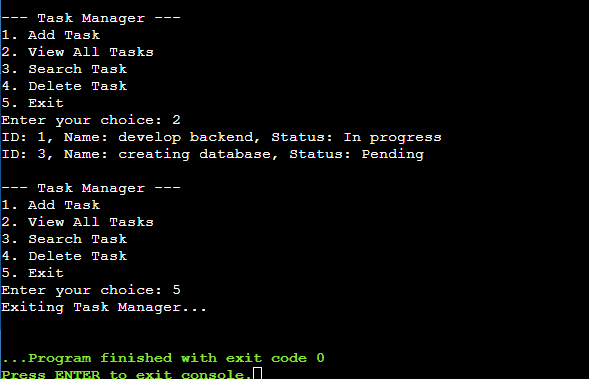
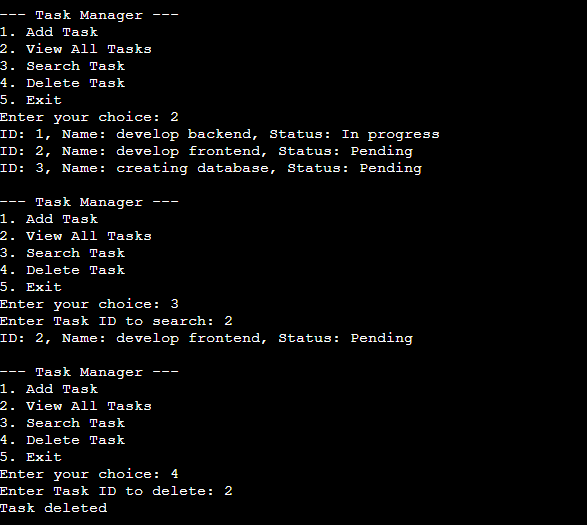
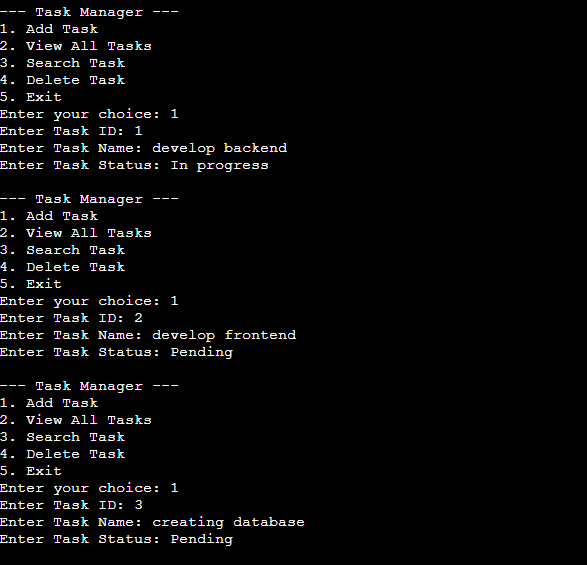
System.out.println("Invalid choice");

}

}

}

}

**OUTPUT:**

1. **Library Management System**

**Explanation:**

**Lineary Search:** Scans element by element,used for unsorted data and for small amounts of data

**Binary Search:** Works on sorted data

| **ALGORITHM** | **BEST CASE** | **AVERAGE CASE** | **WORST CASE** |
| --- | --- | --- | --- |
| LINEAR SEARCH | O(1) | O(n) | O(n) |
| BINARY SEARCH | O(1) | O(log n) | O(log n) |

**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("ID: " + bookId + ", Title: " + title + ", Author: " + author);

}

}

public class Main {

public static void linearSearch(List<Book> books, String searchTitle) {

boolean found = false;

for (Book b : books) {

if (b.title.equalsIgnoreCase(searchTitle)) {

b.display();

found = true;

}

}

if (!found) {

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

}

}

public static void binarySearch(List<Book> books, String searchTitle) {

int left = 0;

int right = books.size() - 1;

while (left <= right) {

int mid = (left + right) / 2;

String midTitle = books.get(mid).title.toLowerCase();

if (midTitle.equals(searchTitle.toLowerCase())) {

books.get(mid).display();

return;

} else if (midTitle.compareTo(searchTitle.toLowerCase()) < 0) {

left = mid + 1;

} else {

right = mid - 1;

}

}

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

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

List<Book> books = new ArrayList<>();

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

int n = sc.nextInt();

sc.nextLine();

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

System.out.print("Enter Book ID: ");

int id = sc.nextInt();

sc.nextLine();

System.out.print("Enter Title: ");

String title = sc.nextLine();

System.out.print("Enter Author: ");

String author = sc.nextLine();

books.add(new Book(id, title, author));

}

System.out.println("\n--- Linear Search ---");

System.out.print("Enter title to search (Linear): ");

String searchLinear = sc.nextLine();

linearSearch(books, searchLinear);

books.sort(Comparator.comparing(b -> b.title.toLowerCase()));

System.out.println("\n--- Binary Search ---");

System.out.print("Enter title to search (Binary): ");

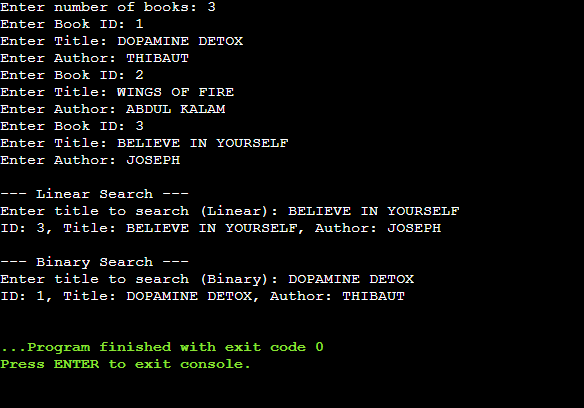
String searchBinary = sc.nextLine();

binarySearch(books, searchBinary);

}

}

**OUTPUT:**

****

1. **Financial Forecasting**

**Explanation:**

**Recursion:**Recursion is a method where a function calls itself to solve smaller instances of the same problem.

Due to recursion ,it solves complex problems

Time Complexity of Recursive algorithm is O(n)

To optimise recursive solution use iterative (i.e loop based )approach

**CODE:**

import java.util.Scanner;

public class Main {

public static double calculateFutureValue(double baseValue, double growthRate, int years) {

if (years == 0) {

return baseValue;

}

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

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Enter initial value: ");

double baseValue = sc.nextDouble();

System.out.print("Enter annual growth rate(in percentage): ");

double growthRate = sc.nextDouble();

System.out.print("Enter number of years to forecast: ");

int years = sc.nextInt();

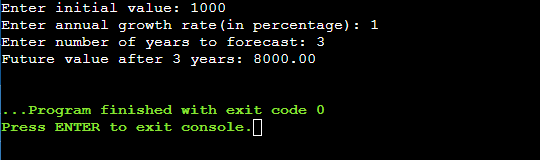
double futureValue = calculateFutureValue(baseValue, growthRate, years);

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

}

}

**OUTPUT:**

****