**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:

1. Data Structures and algorithms are essential because while handling large inventories adding, updating and deleting without data structures would be a difficult task and time and space complexity of the program would be increased rapidly.

The type of data structures that may be used for Inventory management systems are HashMap, ArrayList, TreeMap. But usage of HashMap would be best because it has key value pairs where key is unique and product could be easily found by its productid

import java.util.HashMap;

import java.util.Scanner;

class Product {

private String productId;

private String productName;

private int quantity;

private double price;

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

this.productId = productId;

this.productName = productName;

this.quantity = quantity;

this.price = price;

}

public String getProductId() {

return productId;

}

public String getProductName() {

return productName;

}

public int getQuantity() {

return quantity;

}

public double getPrice() {

return price;

}

public void setQuantity(int quantity) {

this.quantity = quantity;

}

public void setPrice(double price) {

this.price = price;

}

public String toString() {

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

}

}

class InventoryManager {

private HashMap<String, Product> inventory;

public InventoryManager() {

inventory = new HashMap<>();

}

public void addProduct(Product product) {

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

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

}

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

Product product = inventory.get(productId);

if (product != null) {

product.setQuantity(quantity);

product.setPrice(price);

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

} else {

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

}

}

public void deleteProduct(String productId) {

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

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

} else {

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

}

}

public void getProduct(String productId) {

Product product = inventory.get(productId);

if (product != null) {

System.out.println(product);

} else {

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

}

}

}

public class Main {

public static void main(String[] args) {

InventoryManager manager = new InventoryManager();

Scanner scanner = new Scanner(System.in);

String option;

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

do {

System.out.println("\nChoose an option:");

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

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

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

System.out.println("4. View Product");

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

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

option = scanner.nextLine();

switch (option) {

case "1":

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

String id = scanner.nextLine();

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

String name = scanner.nextLine();

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

int qty = Integer.parseInt(scanner.nextLine());

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

double price = Double.parseDouble(scanner.nextLine());

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

break;

case "2":

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

String updateId = scanner.nextLine();

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

int newQty = Integer.parseInt(scanner.nextLine());

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

double newPrice = Double.parseDouble(scanner.nextLine());

manager.updateProduct(updateId, newQty, newPrice);

break;

case "3":

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

String deleteId = scanner.nextLine();

manager.deleteProduct(deleteId);

break;

case "4":

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

String viewId = scanner.nextLine();

manager.getProduct(viewId);

break;

case "5":

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

break;

default:

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

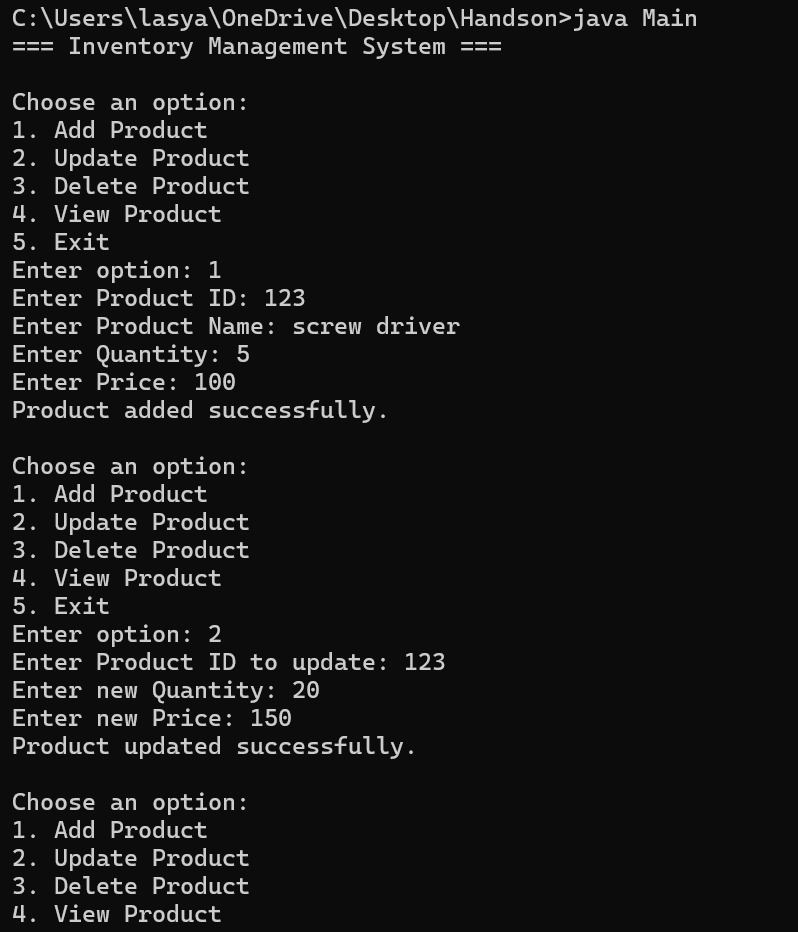
}

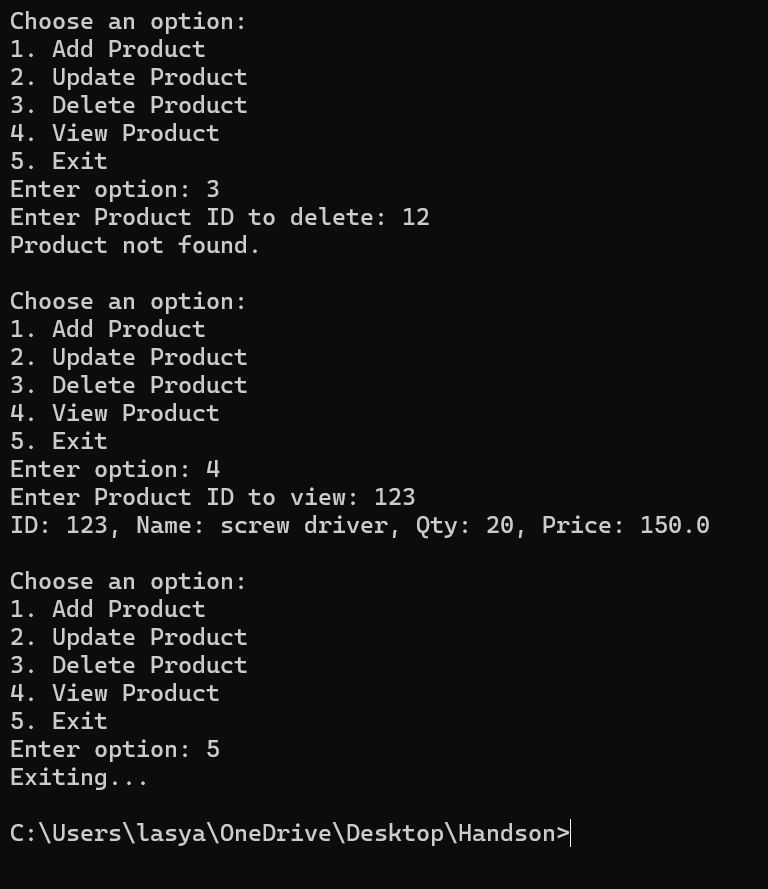
} while (!option.equals("5"));

scanner.close();

}

}





The time complexity for:

1. Adding is O(1)
2. Deleting is O(1)
3. Update is O(1)

**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:

1. It is used to compare efficiency of the algorithms. It provides an upper bound of an algorithm’s space and time complexity.

Given two functions f(n) and g(n), we say that f(n) is O(g(n)) if there exist constants c > 0 and n0 >= 0 such that f(n) <= c\*g(n) for all n >= n0.

For Linear Search: The best case scenario for search operation is O(1). It means the item to be searched is at the first position itself.

The worst case scenario would be O(n). It means the item to be searched is at the last position.

The average case would be O(n/2), that means in the middle.

For Binary Search: The best case would be O(1). It means the item to be searched is at the first position itself.

The average and worst case scenario would be O(log n). Because binary search always halves the input value.

import java.util.Arrays;

import java.util.Comparator;

class Product {

String productId;

String productName;

String category;

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

this.productId = productId;

this.productName = productName;

this.category = category;

}

public String toString() {

return "ID: " + productId + ", Name: " + productName + ", Category: " + category;

}

}

public class Main {

// Linear Search

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

for (Product product : products) {

if (product.productName.equalsIgnoreCase(name)) {

return product;

}

}

return null;

}

// Binary Search

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

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

int left = 0, right = products.length - 1;

while (left <= right) {

int mid = (left + right) / 2;

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

if (cmp == 0)

return products[mid];

else if (cmp < 0)

left = mid + 1;

else

right = mid - 1;

}

return null;

}

public static void main(String[] args) {

Product[] products = {

new Product("1", "Laptop", "Electronics"),

new Product("2", "Shoes", "Fashion"),

new Product("3", "Book", "Education"),

new Product("4", "Smartphone", "Electronics"),

new Product("5", "Table", "Furniture")

};

String searchName = "Laptop";

System.out.println("Linear Search:");

Product result1 = linearSearch(products, searchName);

System.out.println(result1 != null ? result1 : "Product not found");

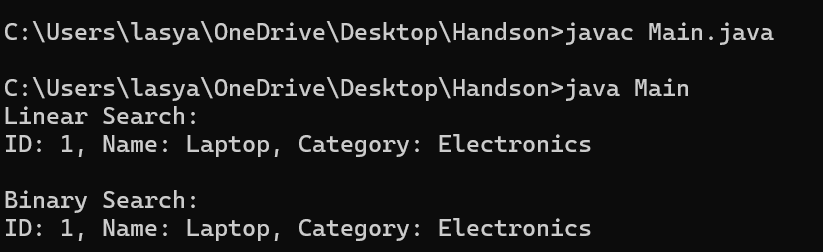
System.out.println("\nBinary Search:");

Product result2 = binarySearch(products, searchName);

System.out.println(result2 != null ? result2 : "Product not found");

}

}



For faster search in e-commerce websites binary search would be better if the data is already in sorted order.

**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:

public class Main {

static class Order {

String orderId;

String customerName;

double totalPrice;

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

this.orderId = orderId;

this.customerName = customerName;

this.totalPrice = totalPrice;

}

public String toString() {

return "OrderID: " + orderId + ", Customer: " + customerName + ", Total: " + totalPrice;

}

}

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 pivotIndex = partition(orders, low, high);

quickSort(orders, low, pivotIndex - 1);

quickSort(orders, pivotIndex + 1, high);

}

}

private 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 printOrders(String message, Order[] orders) {

System.out.println(message);

for (Order order : orders) {

System.out.println(order);

}

System.out.println();

}

public static void main(String[] args) {

Order[] orders = {

new Order("1", "Alice", 2123.50),

new Order("2", "Bob", 15012.00),

};

Order[] bubbleSorted = orders.clone();

bubbleSort(bubbleSorted);

printOrders("Bubble Sorted Orders (by totalPrice):", bubbleSorted);

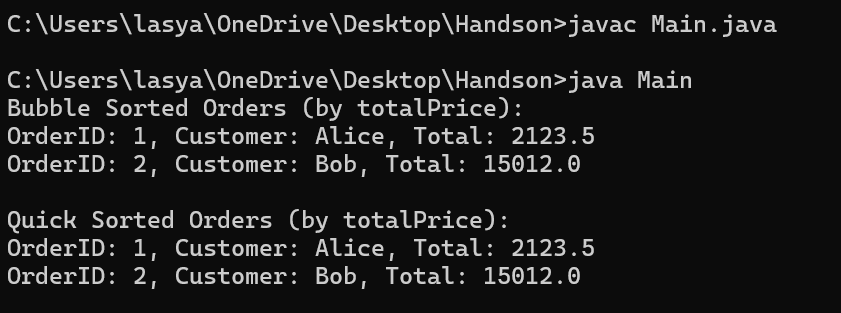
Order[] quickSorted = orders.clone();

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

printOrders("Quick Sorted Orders (by totalPrice):", quickSorted);

}

}



Bubble Sort: Best time complexity-O(n) Avg and worst case-O(n^2)

Quick Sort: Best and avg time complexity-O(n log n) Worst case- O(n^2)

So, Quick sort is preferred in most cases compared to bubble sort.

**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.

Arrays are stored in continuous memory locations. Each element is accessed by index. Because accessing the element is fast.

import java.util.Scanner;

public class Main {

static class Employee {

String employeeId;

String name;

String position;

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 toString() {

return "ID: " + employeeId + ", Name: " + name + ", Position: " + position + ", Salary: " + salary;

}

}

static final int MAX\_EMPLOYEES = 100;

static Employee[] employees = new Employee[MAX\_EMPLOYEES];

static int count = 0;

public static void addEmployee(Employee e) {

if (count < MAX\_EMPLOYEES) {

employees[count++] = e;

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

} else {

System.out.println("full.");

}

}

public static void searchEmployee(String id) {

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

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

System.out.println("Employee Found: " + employees[i]);

return;

}

}

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

}

public static void traverseEmployees() {

if (count == 0) {

System.out.println("No employees.");

return;

}

System.out.println("Employee List:");

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

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

}

}

public static void deleteEmployee(String id) {

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

if (employees[i].employeeId.equals(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 scanner = new Scanner(System.in);

String option;

do {

System.out.println("\nEmployee Management System");

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

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

System.out.println("3. View All Employees");

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

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

System.out.print("Choose an option: ");

option = scanner.nextLine();

switch (option) {

case "1":

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

String id = scanner.nextLine();

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

String name = scanner.nextLine();

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

String position = scanner.nextLine();

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

double salary = Double.parseDouble(scanner.nextLine());

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

break;

case "2":

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

String searchId = scanner.nextLine();

searchEmployee(searchId);

break;

case "3":

traverseEmployees();

break;

case "4":

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

String deleteId = scanner.nextLine();

deleteEmployee(deleteId);

break;

case "5":

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

break;

default:

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

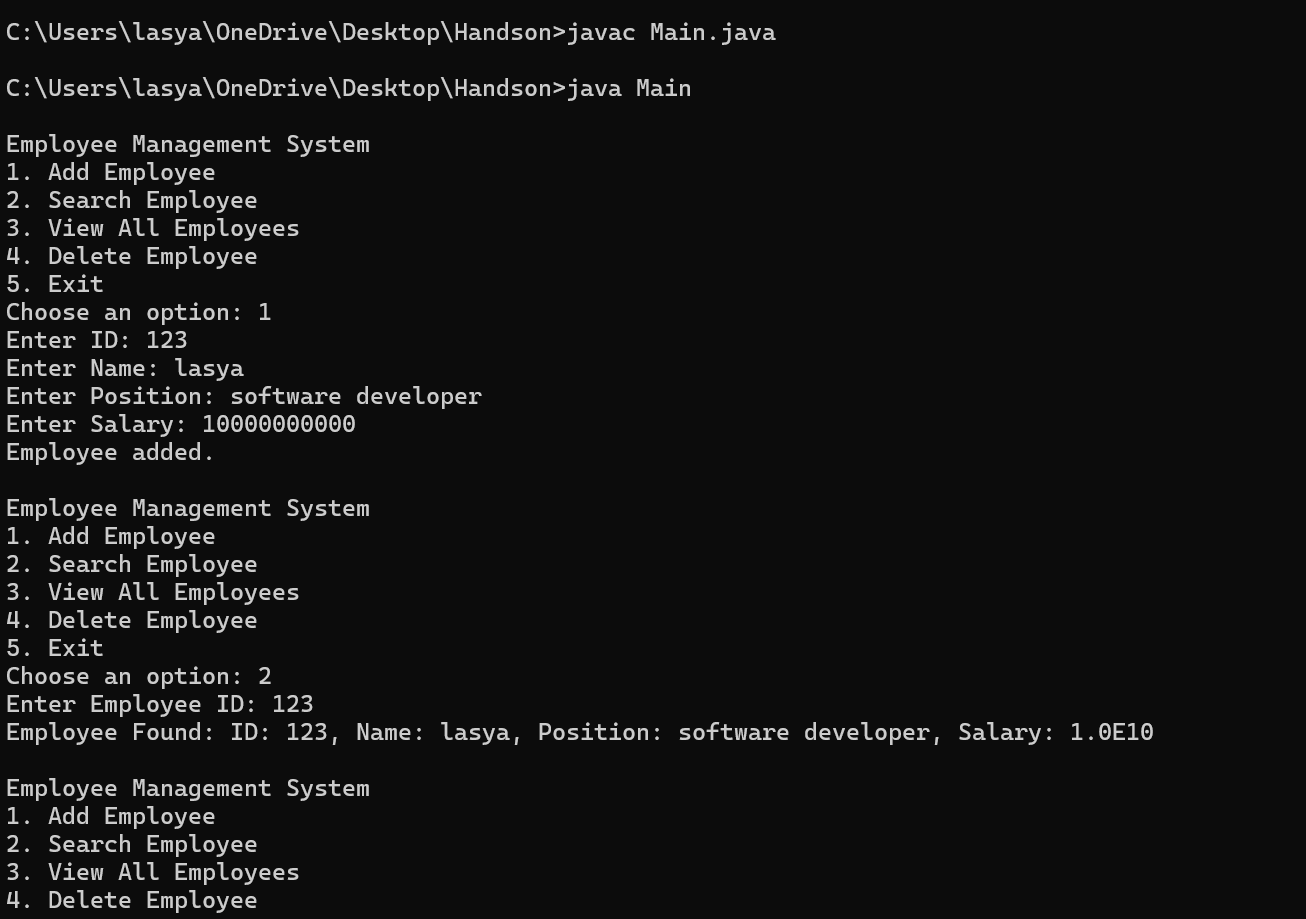
}

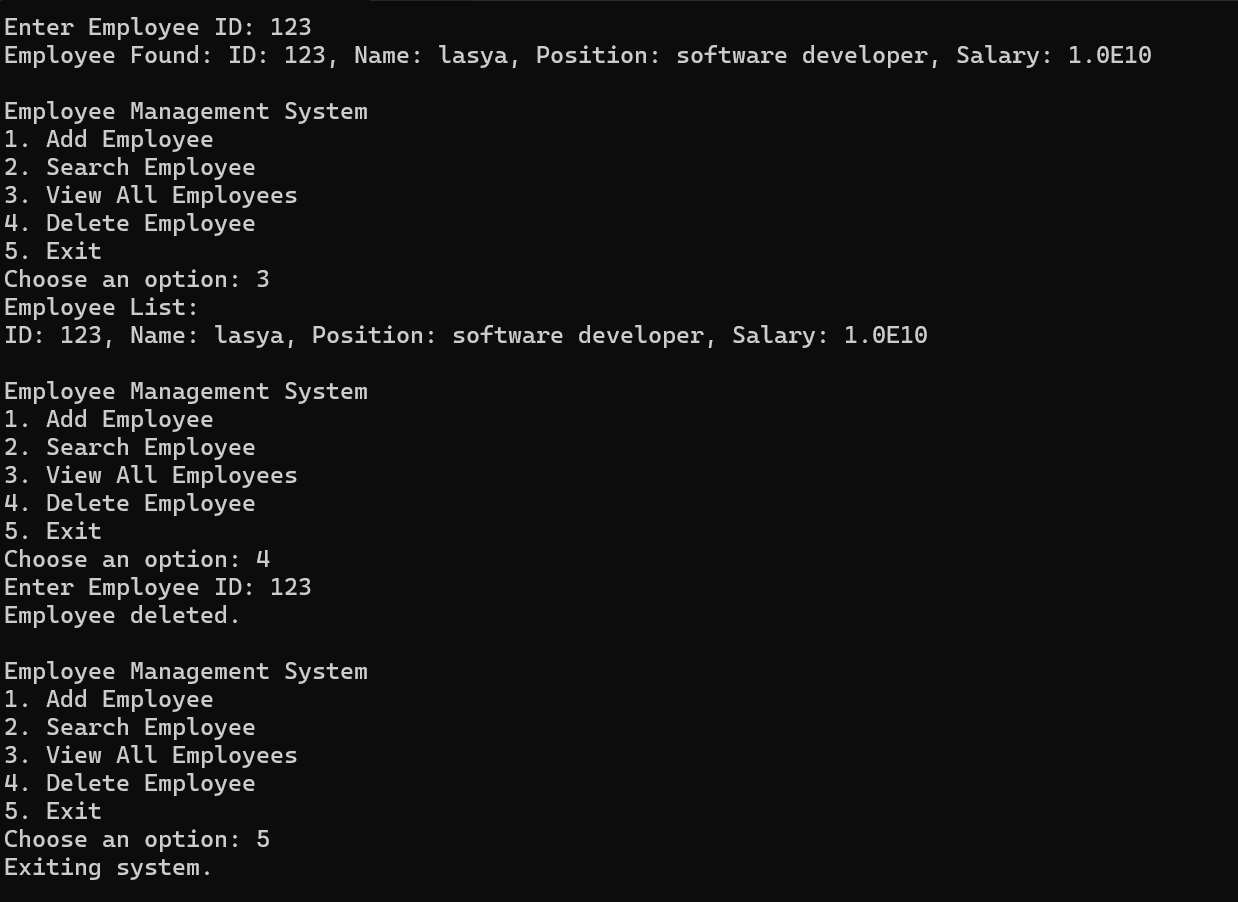
} while (!option.equals("5"));

scanner.close();

}

}





Add- O(1)- Adding is done at the end.

Search- O(n)- As the array will be an unsorted one.

Traverse- O(n)- Iterating through each element

Limitations of array: 1. Fixed array

2. Frequent insertions and deletions are complicated.

**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:

import java.util.Scanner;

public class Main {

static class Task {

String taskId;

String taskName;

String status;

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

this.taskId = taskId;

this.taskName = taskName;

this.status = status;

}

public String toString() {

return "TaskID: " + taskId + ", Name: " + taskName + ", Status: " + status;

}

}

static class Node {

Task task;

Node next;

public Node(Task task) {

this.task = task;

this.next = null;

}

}

static class TaskLinkedList {

Node head;

public void addTask(Task task) {

Node newNode = new Node(task);

if (head == null) {

head = newNode;

} else {

Node current = head;

while (current.next != null) {

current = current.next;

}

current.next = newNode;

}

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

}

public void searchTask(String id) {

Node current = head;

while (current != null) {

if (current.task.taskId.equals(id)) {

System.out.println("Found: " + current.task);

return;

}

current = current.next;

}

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

}

public void displayTasks() {

if (head == null) {

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

return;

}

Node current = head;

System.out.println("Task List:");

while (current != null) {

System.out.println(current.task);

current = current.next;

}

}

public void deleteTask(String id) {

if (head == null) {

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

return;

}

if (head.task.taskId.equals(id)) {

head = head.next;

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

return;

}

Node current = head;

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

current = current.next;

}

if (current.next != null) {

current.next = current.next.next;

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

} else {

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

}

}

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

TaskLinkedList taskList = new TaskLinkedList();

String option;

do {

System.out.println("\nTask Management System");

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

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

System.out.println("3. View All Tasks");

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

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

System.out.print("Choose an option: ");

option = scanner.nextLine();

switch (option) {

case "1":

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

String id = scanner.nextLine();

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

String name = scanner.nextLine();

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

String status = scanner.nextLine();

taskList.addTask(new Task(id, name, status));

break;

case "2":

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

String searchId = scanner.nextLine();

taskList.searchTask(searchId);

break;

case "3":

taskList.displayTasks();

break;

case "4":

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

String deleteId = scanner.nextLine();

taskList.deleteTask(deleteId);

break;

case "5":

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

break;

default:

System.out.println("Invalid option. Try again.");

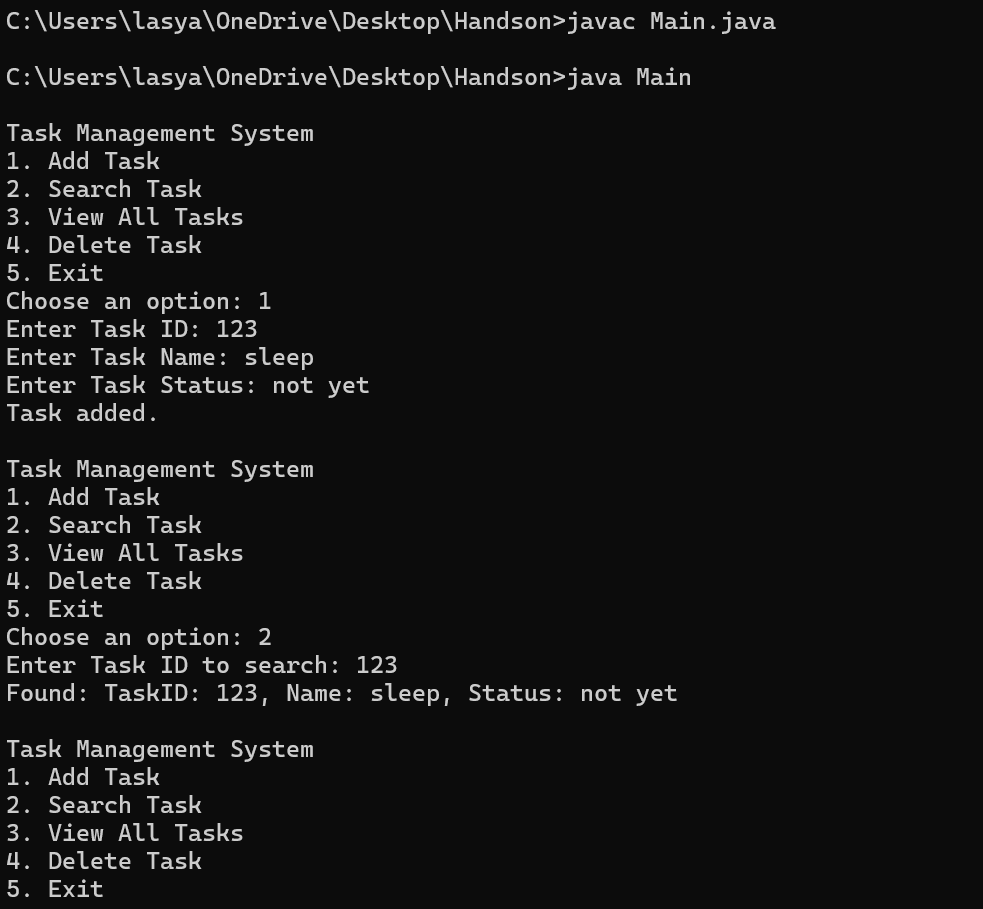
}

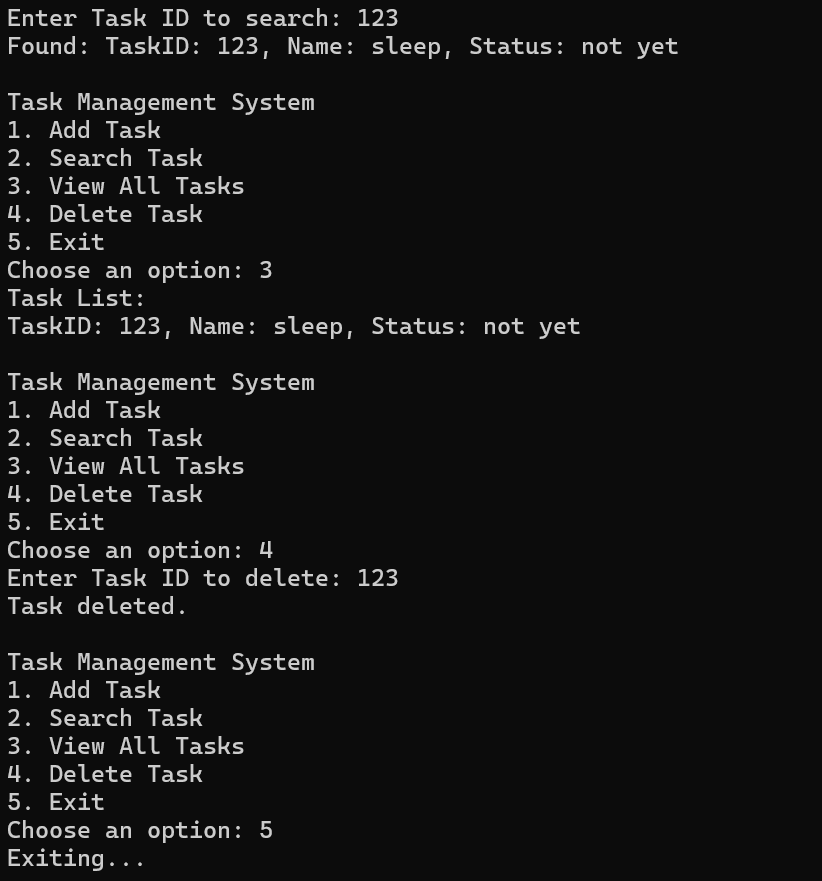
} while (!option.equals("5"));

scanner.close();

}

}





**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:Linear Search: Searches the element one at a time(item by item). It is best for unsorted data.

Binary Search: Searches the sorted by finding middle element then searching according to the key element by seeing if key<middle element or >middle element then going left or right according to it.

import java.util.Arrays;

import java.util.Comparator;

import java.util.Scanner;

public class Main {

static class Book {

String bookId;

String title;

String author;

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

this.bookId = bookId;

this.title = title;

this.author = author;

}

public String toString() {

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

}

}

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

boolean found = false;

for (Book book : books) {

if (book != null && book.title.equalsIgnoreCase(title)) {

System.out.println("Found: " + book);

found = true;

}

}

if (!found) {

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

}

}

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

int low = 0;

int high = books.length - 1;

while (low <= high) {

int mid = (low + high) / 2;

if (books[mid] == null) {

high--;

continue;

}

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

if (comparison == 0) {

System.out.println("Found: " + books[mid]);

return;

} else if (comparison < 0) {

low = mid + 1;

} else {

high = mid - 1;

}

}

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

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

Book[] books = new Book[100];

int count = 0;

String option;

do {

System.out.println("\nLibrary Management System");

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

System.out.println("2. Linear Search by Title");

System.out.println("3. Binary Search by Title (sorted)");

System.out.println("4. View All Books");

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

System.out.print("Choose an option: ");

option = scanner.nextLine();

switch (option) {

case "1":

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

String id = scanner.nextLine();

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

String title = scanner.nextLine();

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

String author = scanner.nextLine();

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

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

break;

case "2":

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

String titleSearch = scanner.nextLine();

linearSearch(Arrays.copyOf(books, count), titleSearch);

break;

case "3":

Book[] sortedBooks = Arrays.copyOf(books, count);

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

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

String binaryTitle = scanner.nextLine();

binarySearch(sortedBooks, binaryTitle);

break;

case "4":

if (count == 0) {

System.out.println("No books to show.");

} else {

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

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

}

}

break;

case "5":

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

break;

default:

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

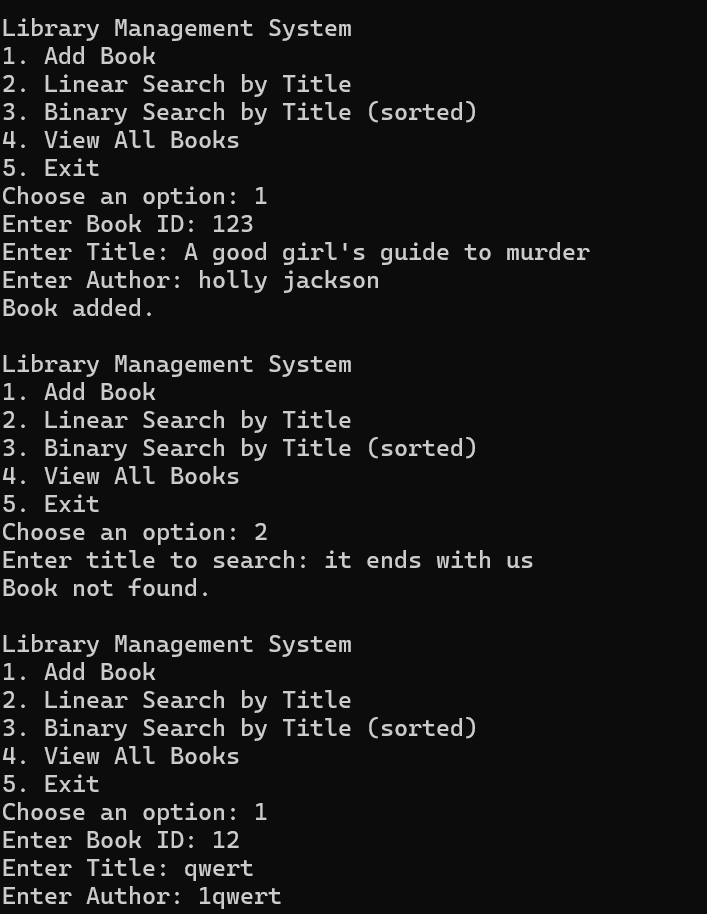
}

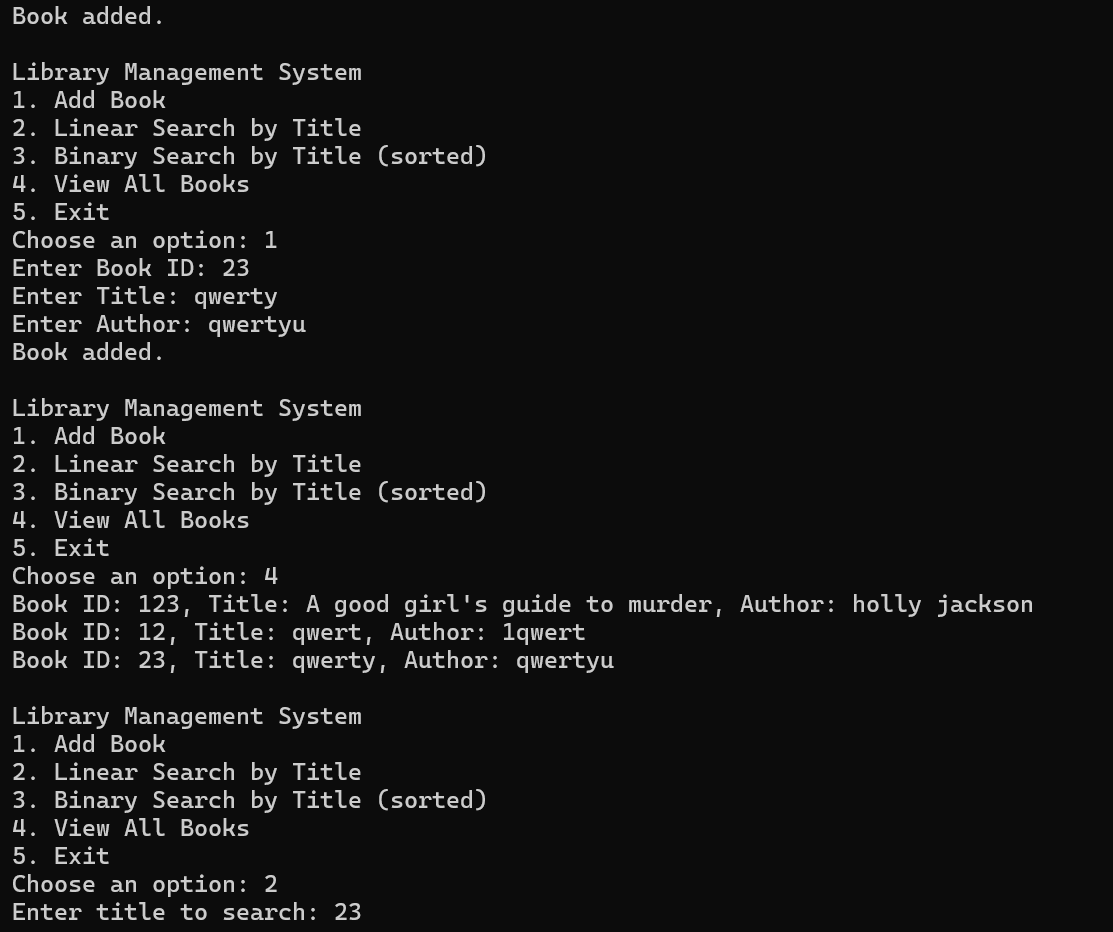
} while (!option.equals("5"));

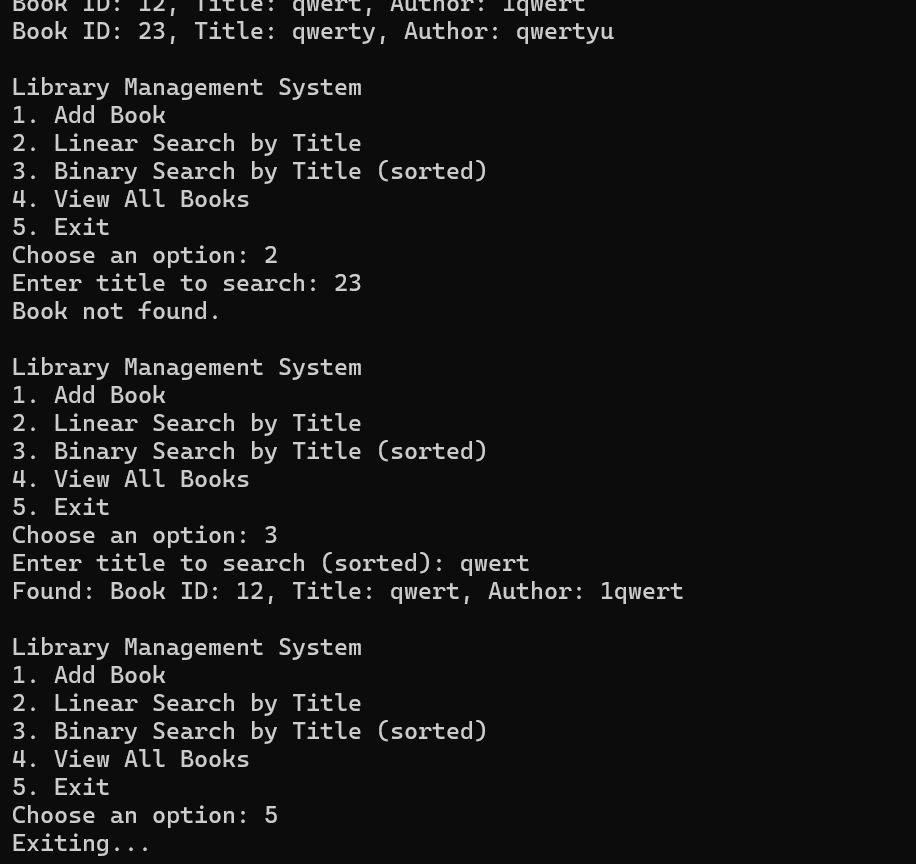
scanner.close();

}

}





****

**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:

Recursion is a technique where a function calls itself to solve smaller instances of a problem. It simplifies problems that can be broken into smaller sub-problems, such as: Fibonacci numbers, Factorials, Financial growth (compounded values).

import java.util.Scanner;

public class Main {

public static double calculateFutureValueRecursive(double currentValue, double growthRate, int years) {

if (years == 0) {

return currentValue;

}

return calculateFutureValueRecursive(currentValue \* (1 + growthRate), growthRate, years - 1);

}

public static double calculateFutureValueIterative(double currentValue, double growthRate, int years) {

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

currentValue \*= (1 + growthRate);

}

return currentValue;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.println("Financial Forecasting Tool");

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

double currentValue = scanner.nextDouble();

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

double growthRate = scanner.nextDouble() / 100;

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

int years = scanner.nextInt();

double futureValue = calculateFutureValueRecursive(currentValue, growthRate, years);

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

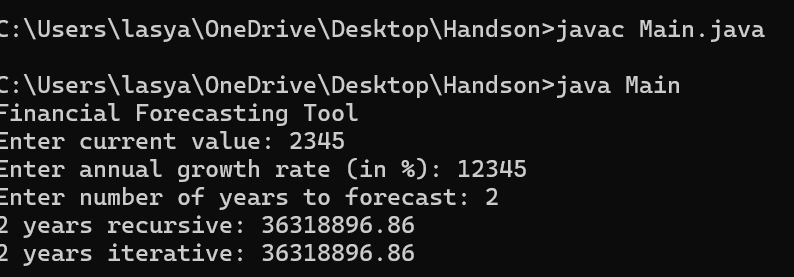
double optimizedValue = calculateFutureValueIterative(currentValue, growthRate, years);

System.out.printf("%d years iterative: %.2f\n", years, optimizedValue);

scanner.close();

}

}



time complexity of your recursive algorithm- O(n)