**DATA STRUCTURES AND ALGORITHMS**

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

1. **Asymptotic Notation & Search Scenarios**

**Big O notation** describes how an algorithm’s runtime (or memory use) grows as the input size *n* increases, ignoring machine‑specific constants.

* **Best case:** input ordering that makes the algorithm finish in the fewest steps.
* **Average case:** expected cost over all possible inputs of size *n*.
* **Worst case:** upper bound—guarantees the algorithm never exceeds this cost.

For search:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| | **Algorithm** | **Best** | **Average** | **Worst** | **Notes** | | --- | --- | --- | --- | --- | | **Linear search** | **O(1)** (match at index 0) | O(n) | **O(n)** | Works on any array, no pre‑sort needed. | | **Binary search** | **O(1)** (match at middle) | O(log n) | **O(log n)** | Requires *sorted* array; divides range in half each step. | |

**CODE**

***File name: Product.java***

package search;

public class Product {

    private final int productId;

    private final String productName;

    private final String category;

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

        this.productId = productId;

        this.productName = productName;

        this.category = category;

    }

    public int getProductId()      { return productId; }

    public String getProductName() { return productName; }

    public String getCategory()    { return category; }

    @Override public String toString() {

        return productId + " | " + productName + " | " + category;

    }

}

***File name: ProductSearch.java***

package search;

import java.util.Arrays;

import java.util.Comparator;

public class ProductSearch {

    /\* ---------- LINEAR SEARCH ---------- \*/

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

        for (Product p : products) {

            if (p.getProductId() == id) return p;    // O(n) worst‑case

        }

        return null;

    }

    /\* ---------- BINARY SEARCH ---------- \*/

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

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

        while (low <= high) {

            int mid = low + (high - low) / 2;        // avoid overflow

            int midId = sortedById[mid].getProductId();

            if (midId == id)         return sortedById[mid];

            else if (midId < id)     low  = mid + 1;

            else                     high = mid - 1;

        }

        return null;                                 // O(log n) worst‑case

    }

    /\* ---------- HELPER TO SORT ---------- \*/

    public static Product[] copyAndSortById(Product[] unsorted) {

        Product[] clone = Arrays.copyOf(unsorted, unsorted.length);

        Arrays.sort(clone, Comparator.comparingInt(Product::getProductId));

        return clone;

    }

}

***File name: SearchDemo.java***

package search;

public class SearchDemo {

    public static void main(String[] args) {

        Product[] catalogue = {

            new Product(105, "Bluetooth Speaker", "Electronics"),

            new Product(102, "Sneakers",          "Footwear"),

            new Product(501, "Coffee Mug",        "Kitchen"),

            new Product(210, "Yoga Mat",          "Fitness")

        };

        /\* Linear search on the original unsorted array \*/

        Product p1 = ProductSearch.linearSearch(catalogue, 210);

        System.out.println("Linear result:  " + p1);

        /\* Binary search on a copy that is sorted by id \*/

        Product[] sorted = ProductSearch.copyAndSortById(catalogue);

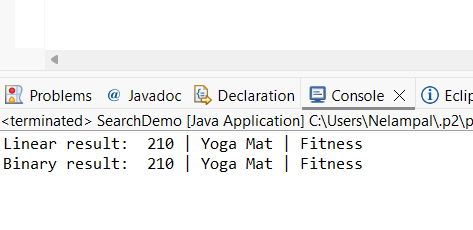
        Product p2 = ProductSearch.binarySearch(sorted, 210);

        System.out.println("Binary result:  " + p2);

    }

}

**OUTPUT**



**Exercise 7: Financial Forecasting**

**Recursion** is when a method calls itself to solve smaller instances of the same problem.  
It simplifies problems like tree traversal, factorials, or financial compounding.

**CODE**

***File name: FutureForecast.java***

package forecast;

public class FinancialForecast {

// Recursive forecast function

public static double forecast(double presentValue, double rate, int years) {

if (years == 0) return presentValue; // Base case

return *forecast*(presentValue, rate, years - 1) \* (1 + rate); // Recursive step

}

public static void main(String[] args) {

double present = 1000.0;

double growthRate = 0.08; // 8%

int periods = 5;

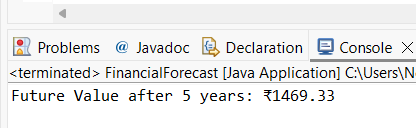
double futureValue = *forecast*(present, growthRate, periods);

System.*out*.printf("Future Value after %d years: ₹%.2f%n", periods, futureValue);

}

}

**OUTPUT**

****

**ANALYSIS**

**TIME COMPLEXITY:**

* T(n) = T(n - 1) + O(1) → O(n)  
  Since we make one recursive call per year.

**OPTIMIZATION:**

While recursion works here, it's not optimal for large n due to:

* Stack overflow risks
* Redundant operations if not managed

**Exercise 1: Inventory Management System**

**CODE**

***File name: Product.java***

package inventory;

public 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;

}

// Getters and setters

public String getProductId() { return productId; }

public String getProductName() { return productName; }

public int getQuantity() { return quantity; }

public double getPrice() { return price; }

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

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

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

@Override

public String toString() {

return "Product[ID=" + productId + ", Name=" + productName + ", Qty=" + quantity + ", Price=₹" + price + "]";

}

}

***File name: InventoryManager.java***

package inventory;

import java.util.HashMap;

import java.util.Map;

public class InventoryManager {

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

// Add a new product

public void addProduct(Product product) {

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

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

} else {

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

System.out.println("Product added: " + product);

}

}

// Update existing product

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

Product p = inventory.get(productId);

if (p != null) {

p.setProductName(name);

p.setQuantity(quantity);

p.setPrice(price);

System.out.println("Product updated: " + p);

} else {

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

}

}

// Delete product

public void deleteProduct(String productId) {

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

System.out.println("Product deleted: " + productId);

} else {

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

}

}

// Display all products

public void showInventory() {

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

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

System.out.println(p);

}

}

}

***File name: InventoryTest.java***

package inventory;

public class InventoryTest {

public static void main(String[] args) {

InventoryManager manager = new InventoryManager();

// Add products

manager.addProduct(new Product("P001", "Keyboard", 50, 899.99));

manager.addProduct(new Product("P002", "Mouse", 100, 499.50));

manager.addProduct(new Product("P003", "Monitor", 30, 8999.00));

manager.showInventory();

// Update a product

manager.updateProduct("P002", "Wireless Mouse", 80, 799.00);

// Delete a product

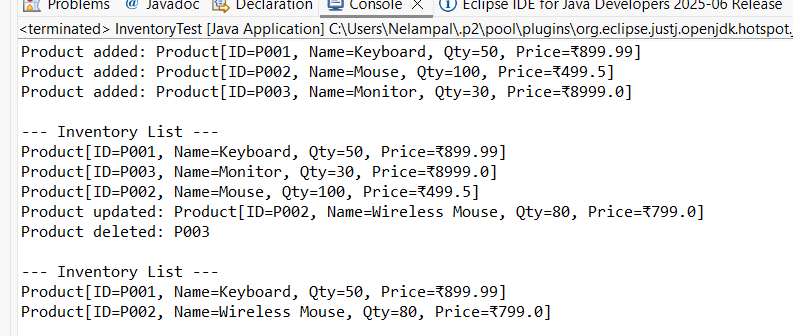
manager.deleteProduct("P003");

manager.showInventory();

}

}

**OUTPUT**



**Exercise 3: Sorting Customer Orders**

**CODE**

***File name: Order.java***

package sort;

public class Order {

private final String orderId;

private final String customerName;

private final double totalPrice;

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

this.orderId = orderId;

this.customerName = customerName;

this.totalPrice = totalPrice;

}

public double getTotalPrice() {

return totalPrice;

}

public String toString() {

return "Order[ID=" + orderId + ", Customer=" + customerName + ", Total=₹" + totalPrice + "]";

}

}

***File: Sorter.java***

package sort;

public class Sorter {

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

Order temp = orders[j];

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

orders[j + 1] = temp;

swapped = true;

}

}

if (!swapped) break;

}

}

// 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);

}

}

// Partition for Quick Sort

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;

}

}

***File: SortTest.java***

package sort;

public class SortTest {

public static void main(String[] args) {

Order[] orders = {

new Order("O101", "Kiruthika", 1200.0),

new Order("O102", "Meena", 800.0),

new Order("O103", "Ravi", 1500.0),

new Order("O104", "Anand", 500.0)

};

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

printOrders(orders);

// Bubble Sort

Order[] bubbleSorted = orders.clone();

Sorter.bubbleSort(bubbleSorted);

System.out.println("\n Sorted by Bubble Sort:");

printOrders(bubbleSorted);

// Quick Sort

Order[] quickSorted = orders.clone();

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

System.out.println("\n Sorted by Quick Sort:");

printOrders(quickSorted);

}

public static void printOrders(Order[] orders) {

for (Order o : orders) {

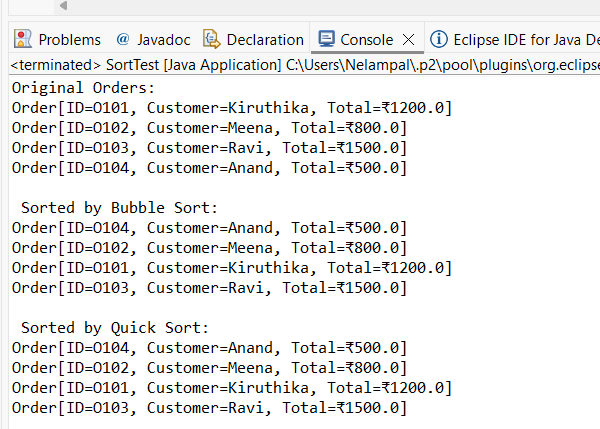
System.out.println(o);

}

}

}

**OUTPUT**



**Exercise 4: Employee Management System**

**CODE**

***File name: Employee.java***

package employee;

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; }

@Override

public String toString() {

return "Employee[ID=" + employeeId + ", Name=" + name + ", Position=" + position + ", Salary=₹" + salary + "]";

}

}

***File name: EmployeeManager.java***

package employee;

public class EmployeeManager {

private final Employee[] employees;

private int count;

public EmployeeManager(int capacity) {

employees = new Employee[capacity];

count = 0;

}

// Add employee

public void addEmployee(Employee emp) {

if (count >= employees.length) {

System.out.println("Cannot add: Employee array is full.");

return;

}

employees[count++] = emp;

System.out.println("Added: " + emp);

}

// Search employee by ID

public Employee searchEmployee(String empId) {

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

if (employees[i].getEmployeeId().equals(empId)) {

return employees[i];

}

}

return null;

}

// Delete employee by ID

public void deleteEmployee(String empId) {

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

if (employees[i].getEmployeeId().equals(empId)) {

// Shift elements left to fill the gap

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

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

}

employees[--count] = null; // remove duplicate at end

System.out.println("Deleted employee with ID: " + empId);

return;

}

}

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

}

// Traverse all employees

public void displayAll() {

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

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

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

}

}

}

***File name: EmployeeTest.java***

package employee;

public class EmployeeTest {

public static void main(String[] args) {

EmployeeManager manager = new EmployeeManager(5); // max 5 employees

manager.addEmployee(new Employee("E001", "Kiruthika", "Developer", 55000));

manager.addEmployee(new Employee("E002", "Ravi", "Manager", 75000));

manager.addEmployee(new Employee("E003", "Meena", "Tester", 45000));

manager.displayAll();

// Search

Employee e = manager.searchEmployee("E002");

System.out.println(e != null ? "Found: " + e : "Not found.");

// Delete

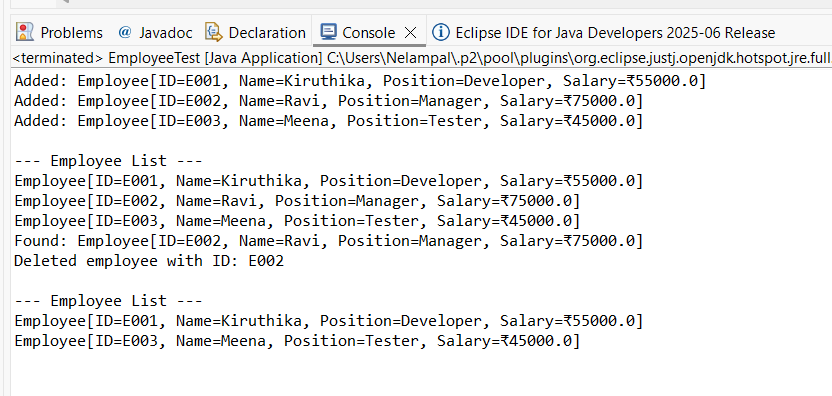
manager.deleteEmployee("E002");

manager.displayAll();

}

}

**OUTPUT**



**Exercise 5: Task Management System**

**CODE**

***File name: Task.java***

package task;

public class Task {

private String taskId;

private String taskName;

private String status;

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

this.taskId = taskId;

this.taskName = taskName;

this.status = status;

}

public String getTaskId() { return taskId; }

@Override

public String toString() {

return "Task[ID=" + taskId + ", Name=" + taskName + ", Status=" + status + "]";

}

}

***File: TaskNode.java***

package task;

public class TaskNode {

Task task;

TaskNode next;

public TaskNode(Task task) {

this.task = task;

this.next = null;

}

}

***File name: TaskManager.java***

package task;

public class TaskManager {

private TaskNode head;

// Add task at the end

public void addTask(Task task) {

TaskNode newNode = new TaskNode(task);

if (head == null) {

head = newNode;

} else {

TaskNode current = head;

while (current.next != null) {

current = current.next;

}

current.next = newNode;

}

System.out.println("Added: " + task);

}

// Search task by ID

public Task searchTask(String taskId) {

TaskNode current = head;

while (current != null) {

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

return current.task;

}

current = current.next;

}

return null;

}

// Traverse and print all tasks

public void traverseTasks() {

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

TaskNode current = head;

while (current != null) {

System.out.println(current.task);

current = current.next;

}

}

// Delete task by ID

public void deleteTask(String taskId) {

if (head == null) {

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

return;

}

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

head = head.next;

System.out.println("Deleted task with ID: " + taskId);

return;

}

TaskNode current = head;

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

current = current.next;

}

if (current.next == null) {

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

} else {

current.next = current.next.next;

System.out.println("Deleted task with ID: " + taskId);

}

}

}

***File name: TaskTest.java***

package task;

public class TaskTest {

public static void main(String[] args) {

TaskManager manager = new TaskManager();

manager.addTask(new Task("T001", "Design UI", "Pending"));

manager.addTask(new Task("T002", "Write backend", "In Progress"));

manager.addTask(new Task("T003", "Test modules", "Pending"));

manager.traverseTasks();

// Search

Task found = manager.searchTask("T002");

System.out.println(found != null ? "Found: " + found : "Task not found.");

// Delete

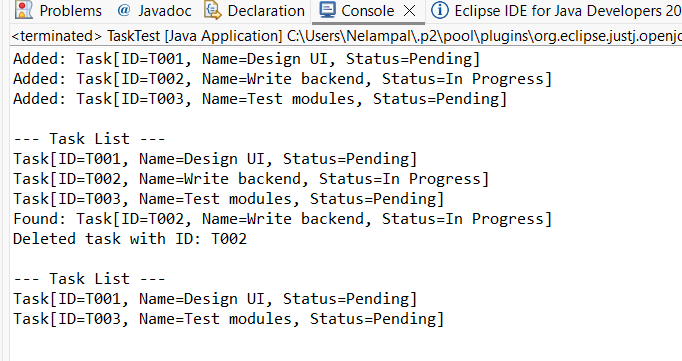
manager.deleteTask("T002");

manager.traverseTasks();

}

}

**OUTPUT**



**Exercise 6: Library Management System**

**CODE**

***File name: Book.java***

package library;

public class Book {

private String bookId;

private String title;

private String author;

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

this.bookId = bookId;

this.title = title;

this.author = author;

}

public String getTitle() {

return title.toLowerCase();

}

@Override

public String toString() {

return "Book[ID=" + bookId + ", Title=\"" + title + "\", Author=" + author + "]";

}

}

***File name: Library.java***

package library;

import java.util.Arrays;

import java.util.Comparator;

public class Library {

// Linear search by title (case insensitive)

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

for (Book book : books) {

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

return book;

}

}

return null;

}

// Binary search by title (requires sorted array)

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

Arrays.sort(books, Comparator.comparing(Book::getTitle)); // Ensure sorted

int low = 0;

int high = books.length - 1;

title = title.toLowerCase();

while (low <= high) {

int mid = (low + high) / 2;

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

if (cmp == 0) return books[mid];

else if (cmp < 0) low = mid + 1;

else high = mid - 1;

}

return null;

}

public static void displayBooks(Book[] books) {

System.out.println("\n--- Library Books ---");

for (Book b : books) {

System.out.println(b);

}

}

}

***File name: LibraryTest.java***

package library;

public class LibraryTest {

public static void main(String[] args) {

Book[] books = {

new Book("B001", "The Alchemist", "Paulo Coelho"),

new Book("B002", "Clean Code", "Robert C. Martin"),

new Book("B003", "The Hobbit", "J.R.R. Tolkien"),

new Book("B004", "Effective Java", "Joshua Bloch")

};

Library.displayBooks(books);

// Linear Search

String searchTitle = "The Hobbit";

Book result = Library.linearSearch(books, searchTitle);

System.out.println("\nLinear Search Result: " + (result != null ? result : "Book not found"));

// Binary Search

searchTitle = "Clean Code";

result = Library.binarySearch(books, searchTitle);

System.out.println("\nBinary Search Result: " + (result != null ? result : "Book not found"));

}

}

**OUTPUT**

