**WEEK 1 : ALGORITHMS\_DATA STRUCTURES**

**MANDATORY**

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

**Understanding Big O for Search**

Big O notation helps us understand how the time an algorithm takes grows as the data size increases.

-Linear Search checks every product one by one until it finds the target or finishes the list.

-Binary Search works on a sorted list by repeatedly dividing it in half to quickly find the product.

**Best, Average, and Worst Cases**

Linear Search:

-Best case: Product is the first item (O(1)) — very fast.

-Average/Worst case: Might have to check most or all items (O(n)) — slower for large lists.

Binary Search:

-Best case: Product is in the middle on first try (O(1)).

-Average/Worst case: Reduces search area by half each time (O(log n)) — much faster on big lists.

**Which Search Method is Better ?**

-If your product list is small or unsorted, linear search is simple and effective.

-If your product list is large and sorted, binary search is much faster and efficient.

CODE:

package weekone;

public class EcommerceSearch {

static class Product {

int productId;

String productName;

String category;

Product(int id, String name, String cat) {

productId = id;

productName = name;

category = cat;

}

public String toString() {

return productId + " - " + productName + " (" + category + ")";

}

}

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

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

if (products[i].productName.equalsIgnoreCase(name)) {

return products[i];

}

}

return null;

}

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

int low = 0;

int high = products.length - 1;

while (low <= high) {

int mid = (low + high) / 2;

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

if (res == 0) return products[mid];

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

else high = mid - 1;

}

return null;

}

public static void main(String[] args) {

Product[] products = {

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

new Product(2, "Phone", "Electronics"),

new Product(3, "Shoes", "Footwear")

};

Product p1 = *linearSearch*(products, "Phone");

System.*out*.println("Linear Search: " + (p1 != null ? p1 : "Not Found"));

Product[] sortedProducts = {

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

new Product(2, "Phone", "Electronics"),

new Product(3, "Shoes", "Footwear")

};

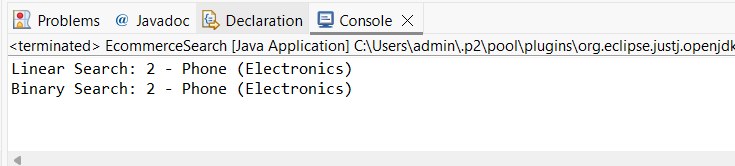
Product p2 = *binarySearch*(sortedProducts, "Phone");

System.*out*.println("Binary Search: " + (p2 != null ? p2 : "Not Found"));

}

}

OUTPUT:



**Exercise 7: Financial Forecasting**

**Understanding Recursive Algorithms**  
Recursion happens when a function calls itself to solve smaller parts of the same problem until it hits a stopping point, called the base case. This method is useful for problems that involve repeating steps or nested structures.

In the given code, the predictFutureValue function calculates the future value by calling itself repeatedly. Each time, it updates the current value by multiplying it by (1 + growthRate) and reduces the number of years left by one. This continues until there are no years left (years == 0), which stops the recursion. Using recursion here keeps the code clean and easy to follow without needing loops.

**Time Complexity:**  
The function makes one recursive call per year until it reaches zero years, so the time complexity is linear — O(n), where n is the number of years. Each call does a small, constant amount of work, so the total work grows proportionally with the number of years.

**Optimization:**  
Because the function only calls itself once each time and doesn’t redo any calculations, it works efficiently for small to medium values of n. But for very large n, an iterative solution (using a loop) might be better to avoid the extra overhead from recursive calls and potential issues like stack overflow.

Also, if the problem involved repeated calculations of the same results (like in Fibonacci numbers), memoization would help. But here, since each step depends only on the immediate previous result and no values are recalculated, memoization isn’t needed.

CODE:

package weekone;

public class FinancialForecasting {

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

if (years == 0) {

return currentValue;

} else {

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

}

}

public static void main(String[] args) {

double presentValue = 1000.0;

double growthRate = 0.05;

int years = 3;

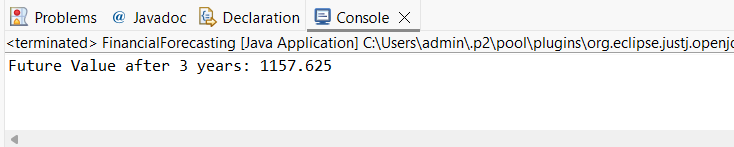
double futureValue = *predictFutureValue*(presentValue, growthRate, years);

System.*out*.println("Future Value after " + years + " years: " + futureValue);

}

}

OUTPUT:



**ADDITIONAL EX**

**Exercise 1: Inventory Management System**

package weekone;

import java.util.\*;

public class InventoryManagementSystem {

static 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;

}

public String toString() {

return productId + " - " + productName + " Qty: " + quantity + " Price: Rs." + price;

}

}

static class Inventory {

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

public void addProduct(Product product) {

productMap.put(product.productId, product);

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

}

public void updateProduct(String productId, int newQuantity, double newPrice) {

Product product = productMap.get(productId);

if (product != null) {

product.quantity = newQuantity;

product.price = newPrice;

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

} else {

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

}

}

public void deleteProduct(String productId) {

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

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

} else {

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

}

}

public void displayInventory() {

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

System.*out*.println(p);

}

}

}

public static void main(String[] args) {

Inventory inventory = new Inventory();

Product p1 = new Product("P101", "Keyboard", 50, 799);

Product p2 = new Product("P102", "Mouse", 100, 499);

Product p3 = new Product("P103", "Monitor", 20, 5499);

inventory.addProduct(p1);

inventory.addProduct(p2);

inventory.addProduct(p3);

System.*out*.println("\nInitial Inventory:");

inventory.displayInventory();

inventory.updateProduct("P102", 80, 459);

inventory.deleteProduct("P103");

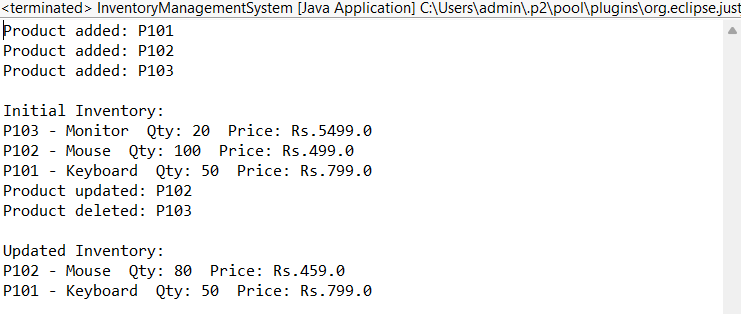
System.*out*.println("\nUpdated Inventory:");

inventory.displayInventory();

}

}

OUTPUT:



**Exercise 3: Sorting Customer Orders**

package weekone;

public class SortingCustomerOrders {

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 + " " + customerName + " Rs." + 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);

}

}

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 printOrders(Order[] orders) {

for (Order order : orders) {

System.*out*.println(order);

}

}

public static void main(String[] args) {

Order[] orders = {

new Order("O101", "Aiswarya", 450.0),

new Order("O102", "Subhi", 1200.0),

new Order("O103", "Sam", 999.0),

new Order("O104", "Harshu", 199.0),

new Order("O105", "Rukku", 1990.0)

};

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

*printOrders*(orders);

*bubbleSort*(orders);

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

*printOrders*(orders);

orders = new Order[] {

new Order("O101", "Aiswarya", 450.0),

new Order("O102", "Subhi", 1200.0),

new Order("O103", "Sam", 999.0),

new Order("O104", "Harshu", 199.0),

new Order("O105", "Rukku", 1990.0)

};

*quickSort*(orders, 0, orders.length - 1);

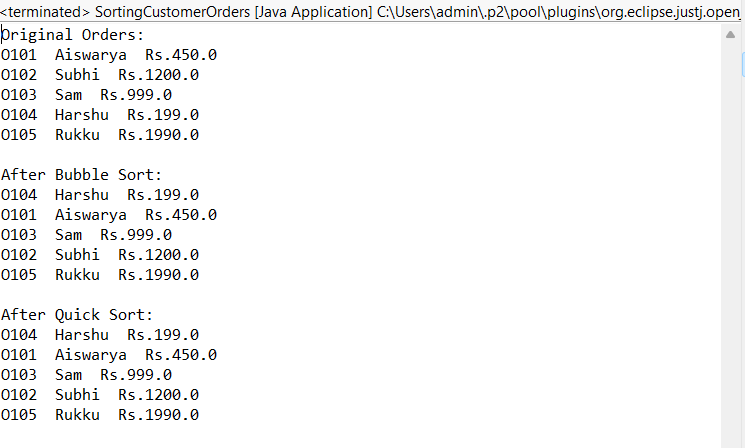
System.*out*.println("\nAfter Quick Sort:");

*printOrders*(orders);

}

}

OUTPUT:



**Exercise 4: Employee Management System**

package weekone;

import java.util.\*;

public class EmployeeManagementSystem {

static 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 String toString() {

return employeeId + " " + name + " " + position + " Rs." + salary;

}

}

static class EmployeeManager {

Employee[] employees;

int count = 0;

public EmployeeManager(int size) {

employees = new Employee[size];

}

public void addEmployee(Employee emp) {

if (count < employees.length) {

employees[count] = emp;

count++;

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

} else {

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

}

}

public void searchEmployee(int id) {

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

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

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

return;

}

}

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

}

public 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 - 1] = null;

count--;

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

return;

}

}

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

}

public void listEmployees() {

if (count == 0) {

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

} else {

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

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

}

}

}

}

public static void main(String[] args) {

EmployeeManager manager = new EmployeeManager(5);

manager.addEmployee(new Employee(1, "Aiswarya", "Developer", 60000));

manager.addEmployee(new Employee(2, "Kiki", "Designer", 55000));

manager.addEmployee(new Employee(3, "Saranya", "Tester", 50000));

System.*out*.println("\nAll Employees:");

manager.listEmployees();

System.*out*.println("\nSearching Employee with ID 2:");

manager.searchEmployee(2);

System.*out*.println("\nDeleting Employee with ID 2:");

manager.deleteEmployee(2);

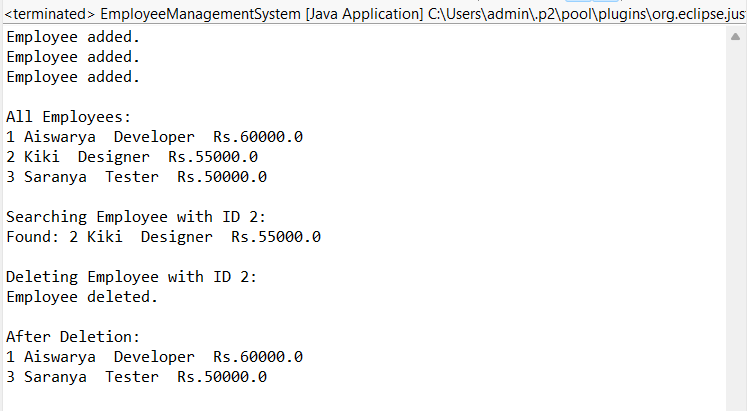
System.*out*.println("\nAfter Deletion:");

manager.listEmployees();

}

}

OUTPUT:



**Exercise 5: Task Management System**

package weekone;

public class TaskManagementSystem {

static 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 String toString() {

return taskId + " " + taskName + " " + status;

}

}

static class TaskList {

Task head;

public void addTask(int taskId, String taskName, String status) {

Task newTask = new Task(taskId, taskName, 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: " + taskName);

}

public void searchTask(int taskId) {

Task temp = head;

while (temp != null) {

if (temp.taskId == taskId) {

System.*out*.println("Found: " + temp);

return;

}

temp = temp.next;

}

System.*out*.println("Task not found with ID " + taskId);

}

public void deleteTask(int taskId) {

if (head == null) {

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

return;

}

if (head.taskId == taskId) {

head = head.next;

System.*out*.println("Task deleted with ID " + taskId);

return;

}

Task prev = head;

Task curr = head.next;

while (curr != null) {

if (curr.taskId == taskId) {

prev.next = curr.next;

System.*out*.println("Task deleted with ID " + taskId);

return;

}

prev = curr;

curr = curr.next;

}

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

}

public void showTasks() {

if (head == null) {

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

return;

}

Task temp = head;

while (temp != null) {

System.*out*.println(temp);

temp = temp.next;

}

}

}

public static void main(String[] args) {

TaskList taskList = new TaskList();

taskList.addTask(101, "Design UI", "Pending");

taskList.addTask(102, "Develop Backend", "In Progress");

taskList.addTask(103, "Testing", "Not Started");

System.*out*.println("\nAll Tasks:");

taskList.showTasks();

System.*out*.println("\nSearching for Task ID 102:");

taskList.searchTask(102);

System.*out*.println("\nDeleting Task ID 101:");

taskList.deleteTask(101);

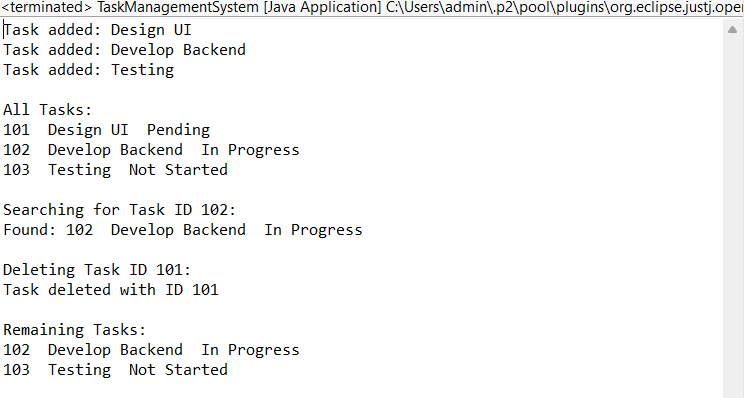
System.*out*.println("\nRemaining Tasks:");

taskList.showTasks();

}

}

OUTPUT:



**Exercise 6: Library Management System**

package weekone;

import java.util.Arrays;

import java.util.Comparator;

public class LibraryManagementSystem {

static 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 String toString() {

return bookId + " " + title + " " + author;

}

}

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

boolean found = false;

for (Book book : books) {

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

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

found = true;

}

}

if (!found) {

System.*out*.println("Book not found using linear search.");

}

}

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

int low = 0;

int high = books.length - 1;

while (low <= high) {

int mid = (low + high) / 2;

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

if (compare == 0) {

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

return;

} else if (compare < 0) {

low = mid + 1;

} else {

high = mid - 1;

}

}

System.*out*.println("Book not found using binary search.");

}

public static void main(String[] args) {

Book[] books = {

new Book(101, "Java Programming", "James"),

new Book(102, "Python Basics", "Guido"),

new Book(103, "Data Structures", "Mark"),

new Book(104, "Algorithms", "Thomas")

};

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

*linearSearch*(books, "Python Basics");

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

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

*binarySearch*(books, "Algorithms");

}

}

OUTPUT:

