WEEK 1-Exercise 2: E-commerce Platform Search Function

**Product.java**

public 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 String toString() {

return "[" + productId + "] " + productName + " - " + category;

}

}

**SearchAlgorithms.java**

import java.util.\*;

public class SearchAlgorithms {

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

for (Product product : products) {

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

return product;

}

}

return null;

}

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;

}

}

**Main.java**

import java.util.Arrays;

import java.util.Comparator;

public class Main {

public static void main(String[] args) {

Product[] products = {

new Product(101, "Shoes", "Footwear"),

new Product(102, "Laptop", "Electronics"),

new Product(103, "Book", "Stationery"),

new Product(104, "Watch", "Accessories"),

new Product(105, "Phone", "Electronics")

};

String target = "Watch";

Product foundLinear = SearchAlgorithms.linearSearch(products, target);

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

// Sort for Binary Search

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

Product foundBinary = SearchAlgorithms.binarySearch(products, target);

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

System.out.println("\nAnalysis:");

System.out.println("Linear Search Time Complexity: O(n)");

System.out.println("Binary Search Time Complexity: O(log n)");

System.out.println("Binary search is faster but needs sorted data.");

}

}

**Analysis**

Big O Analysis

Linear Search: O(n)

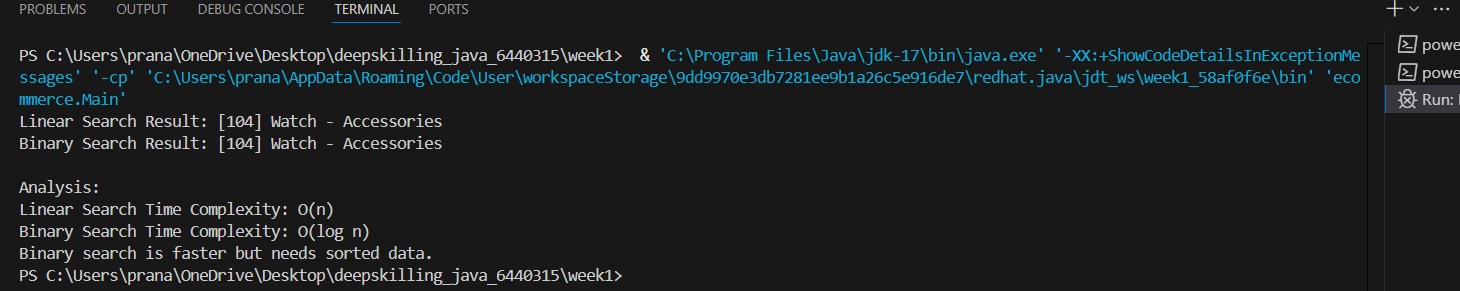
- Best Case: O(1)

- Average/Worst Case: O(n)

Binary Search: O(log n)

- Best Case: O(1)

- Average/Worst Case: O(log n)



**WEEK 1- Exercise 7: Financial Forecasting**

**FinancialForecast.java**

public class FinancialForecast {

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

if (years == 0) {

return amount;

}

return forecast(amount \* (1 + rate), rate, years - 1);

}

public static void main(String[] args) {

double initialAmount = 10000.0;

double growthRate = 0.05; // 5%

int years = 5;

double futureValue = forecast(initialAmount, growthRate, years);

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

}

}

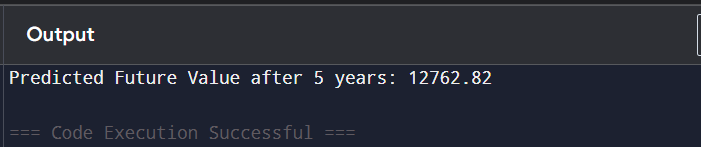
**Time Complexity**

The recursive method makes **one recursive call per year**:

* So, **Time Complexity = O(n)** where n = years

**Optimization Tip**

* If the calculation became complex (e.g., with caching or multiple factors), use **memoization** or **dynamic programming** to avoid redundant calculations.
* For this simple exponential growth, recursion is clear and performant.



**Exercise 1: Inventory Management System**

import java.util.\*;

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;

}

}

class Inventory {

HashMap<Integer, Product> products = new HashMap<>();

public void addProduct(Product product) {

products.put(product.productId, product);

}

public void updateProduct(Product product) {

products.put(product.productId, product);

}

public void deleteProduct(int productId) {

products.remove(productId);

}

}

public class InventorySystem {

public static void main(String[] args) {

Inventory inventory = new Inventory();

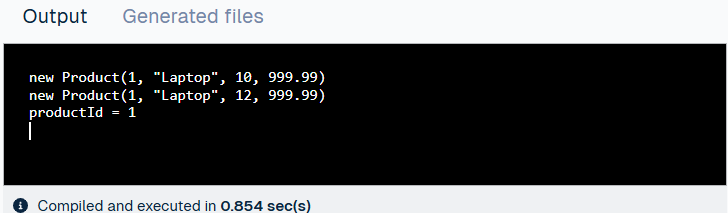
inventory.addProduct(new Product(1, "Laptop", 10, 999.99));

inventory.updateProduct(new Product(1, "Laptop", 12, 999.99));

inventory.deleteProduct(1);

}

}



**Exercise 3: Sorting Customer Orders**

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 class SortOrders {

public static void bubbleSort(Order[] arr) {

int n = arr.length;

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

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

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

Order temp = arr[j];

arr[j] = arr[j + 1];

arr[j + 1] = temp;

}

}

}

}

public static void quickSort(Order[] arr, int low, int high) {

if (low < high) {

int pi = partition(arr, low, high);

quickSort(arr, low, pi - 1);

quickSort(arr, pi + 1, high);

}

}

private static int partition(Order[] arr, int low, int high) {

double pivot = arr[high].totalPrice;

int i = low - 1;

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

if (arr[j].totalPrice < pivot) {

i++;

Order temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

}

Order temp = arr[i + 1];

arr[i + 1] = arr[high];

arr[high] = temp;

return i + 1;

}

public static void main(String[] args) {

Order[] orders = {

new Order(1, "Alice", 300),

new Order(2, "Bob", 150),

new Order(3, "Carol", 450)

};

bubbleSort(orders); // or quickSort(orders, 0, orders.length - 1);

for (Order o : orders) {

System.out.println(o.customerName + ": " + o.totalPrice);

}

}

}



**Exercise 4: Employee Management System**

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 class EmployeeSystem {

Employee[] employees = new Employee[100];

int count = 0;

public void addEmployee(Employee emp) {

employees[count++] = emp;

}

public Employee searchEmployee(int id) {

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

if (employees[i].employeeId == id) return employees[i];

}

return null;

}

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] = null;

break;

}

}

}

public void traverse() {

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

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

}

}

public static void main(String[] args) {

EmployeeSystem sys = new EmployeeSystem();

sys.addEmployee(new Employee(1, "John", "Manager", 75000));

sys.addEmployee(new Employee(2, "Jane", "Clerk", 40000));

sys.traverse();

sys.deleteEmployee(1);

sys.traverse();

}

} 

**Exercise 5: Task Management System**

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;

}

}

public class TaskList {

Task head;

public void addTask(Task task) {

task.next = head;

head = task;

}

public void deleteTask(int id) {

Task temp = head, prev = null;

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

prev = temp;

temp = temp.next;

}

if (temp != null) {

if (prev == null) head = temp.next;

else prev.next = temp.next;

}

}

public Task searchTask(int id) {

Task temp = head;

while (temp != null) {

if (temp.taskId == id) return temp;

temp = temp.next;

}

return null;

}

public void traverse() {

Task temp = head;

while (temp != null) {

System.out.println(temp.taskName);

temp = temp.next;

}

}

public static void main(String[] args) {

TaskList list = new TaskList();

list.addTask(new Task(1, "Design UI", "Pending"));

list.addTask(new Task(2, "Implement Backend", "In Progress"));

list.traverse();

list.deleteTask(1);

list.traverse();

}

}



**Exercise 6: Library Management System**

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 class LibrarySearch {

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

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

if (books[i].title.equals(title)) return i;

}

return -1;

}

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

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

while (left <= right) {

int mid = (left + right) / 2;

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

if (cmp == 0) return mid;

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

else right = mid - 1;

}

return -1;

}

public static void main(String[] args) {

Book[] books = {

new Book(1, "Java Programming", "James Gosling"),

new Book(2, "Algorithms", "Robert Sedgewick"),

new Book(3, "Clean Code", "Robert C. Martin")

};

System.out.println(linearSearch(books, "Clean Code"));

System.out.println(binarySearch(books, "Algorithms"));

}

}

