# Cognizant Digital Nurture 4.0 Deep Skilling

## Algorithms Data Structures

**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 :**

import java.util.Arrays;

import java.util.Comparator;

public class ECommerceSearch {

static class Product {

int productId;

String productName;

String category;

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

this.productId = productId;

this.productName = productName;

this.category = category;

}

@Override

public String toString() {

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

}

}

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 comparison = products[mid].productName.compareToIgnoreCase(targetName);

if (comparison == 0) {

return products[mid];

} else if (comparison < 0) {

low = mid + 1;

} else {

high = mid - 1;

}

}

return null;

}

public static void main(String[] args) {

Product[] products = {

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

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

new Product(103, "Shirt", "Clothing"),

new Product(104, "Headphones", "Electronics"),

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

};

String searchTerm = "Phone";

Product result1 = linearSearch(products, searchTerm);

System.out.println("Linear Search Result: " + (result1 != null ? result1 : "Not Found"));

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

Product result2 = binarySearch(products, searchTerm);

System.out.println("Binary Search Result: " + (result2 != null ? result2 : "Not Found"));

}

}

**Output :**

Linear Search Result: [105, Phone, Electronics]

Binary Search Result: [105, Phone, Electronics]

**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 :**

public class FinancialForecast {

public static double futureValueRecursive(double initialAmount, double growthRate, int years) {

if (years == 0) {

return initialAmount;

}

return futureValueRecursive(initialAmount, growthRate, years - 1) \* (1 + growthRate);

}

public static double futureValueIterative(double initialAmount, double growthRate, int years) {

double result = initialAmount;

for (int i = 1; i <= years; i++) {

result \*= (1 + growthRate);

}

return result;

}

public static void main(String[] args) {

double initialAmount = 10000;

double annualGrowthRate = 0.08;

int years = 5;

double fvRecursive = futureValueRecursive(initialAmount, annualGrowthRate, years);

double fvIterative = futureValueIterative(initialAmount, annualGrowthRate, years);

System.out.printf("Future Value (Recursive): ?%.2f%n", fvRecursive);

System.out.printf("Future Value (Iterative): ?%.2f%n", fvIterative);

}

}

**Output :**

Future Value (Recursive): ₹14693.28

Future Value (Iterative): ₹14693.28

**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 :**

import java.util.\*;

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 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 +

", Qty: " + quantity + ", Price: $" + price;

}

}

class Inventory {

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

public void addProduct(Product product) {

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

}

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

Product p = products.get(productId);

if (p != null) {

p.setProductName(name);

p.setQuantity(qty);

p.setPrice(price);

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

} else {

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

}

}

public void deleteProduct(String productId) {

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

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

} else {

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

}

}

public void displayInventory() {

if (products.isEmpty()) {

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

} else {

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

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

System.out.println(p);

}

}

}

}

public class InventoryManagementSystem {

public static void main(String[] args) {

Inventory inventory = new Inventory();

Product p1 = new Product("P101", "Mouse", 50, 299.99);

Product p2 = new Product("P102", "Keyboard", 30, 499.49);

Product p3 = new Product("P103", "Monitor", 20, 7499.00);

inventory.addProduct(p1);

inventory.addProduct(p2);

inventory.addProduct(p3);

inventory.displayInventory();

inventory.updateProduct("P102", "Mechanical Keyboard", 25, 699.00);

inventory.deleteProduct("P101");

System.out.println("\nAfter Update and Delete:");

inventory.displayInventory();

}

}

**Output :**

Current Inventory:

ID: P101, Name: Mouse, Qty: 50, Price: $299.99

ID: P102, Name: Keyboard, Qty: 30, Price: $499.49

ID: P103, Name: Monitor, Qty: 20, Price: $7499.0

Product updated.

Product deleted.

After Update and Delete:

Current Inventory:

ID: P102, Name: Mechanical Keyboard, Qty: 25, Price: $699.0

ID: P103, Name: Monitor, Qty: 20, Price: $7499.0

**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 :**

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;

}

@Override

public String toString() {

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

}

}

class BubbleSort {

public static void sort(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].totalPrice > orders[j + 1].totalPrice) {

Order temp = orders[j];

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

orders[j + 1] = temp;

swapped = true;

}

}

if (!swapped) break;

}

}

}

class QuickSort {

public static void sort(Order[] orders, int low, int high) {

if (low < high) {

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

sort(orders, low, pi - 1);

sort(orders, pi + 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 class SortCustomerOrders {

public static void main(String[] args) {

Order[] orders = {

new Order("O1001", "Alice", 950.75),

new Order("O1002", "Bob", 245.60),

new Order("O1003", "Charlie", 1549.00),

new Order("O1004", "Diana", 500.00)

};

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

for (Order o : orders) System.out.println(o);

Order[] bubbleSortedOrders = orders.clone();

BubbleSort.sort(bubbleSortedOrders);

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

for (Order o : bubbleSortedOrders) System.out.println(o);

Order[] quickSortedOrders = orders.clone();

QuickSort.sort(quickSortedOrders, 0, quickSortedOrders.length - 1);

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

for (Order o : quickSortedOrders) System.out.println(o);

}

}

**Output :**

Original Orders:

OrderID: O1001, Customer: Alice, Total: $950.75

OrderID: O1002, Customer: Bob, Total: $245.6

OrderID: O1003, Customer: Charlie, Total: $1549.0

OrderID: O1004, Customer: Diana, Total: $500.0

Orders Sorted by Bubble Sort:

OrderID: O1002, Customer: Bob, Total: $245.6

OrderID: O1004, Customer: Diana, Total: $500.0

OrderID: O1001, Customer: Alice, Total: $950.75

OrderID: O1003, Customer: Charlie, Total: $1549.0

Orders Sorted by Quick Sort:

OrderID: O1002, Customer: Bob, Total: $245.6

OrderID: O1004, Customer: Diana, Total: $500.0

OrderID: O1001, Customer: Alice, Total: $950.75

OrderID: O1003, Customer: Charlie, Total: $1549.0

**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.

**Solution :**

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 String toString() {

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

}

}

public class EmployeeManagementSystem {

static final int MAX\_EMPLOYEES = 100;

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

static int count = 0;

public static void addEmployee(Employee emp) {

if (count < MAX\_EMPLOYEES) {

employees[count++] = emp;

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

} else {

System.out.println("Cannot add more employees. Limit reached.");

}

}

public static void searchEmployee(int id) {

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

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

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

return;

}

}

System.out.println("Employee with ID " + id + " not found.");

}

public static void traverseEmployees() {

if (count == 0) {

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

return;

}

System.out.println("All Employees:");

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

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

}

}

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;

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

return;

}

}

System.out.println("Employee with ID " + id + " not found.");

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

while (true) {

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

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

int ch = sc.nextInt();

sc.nextLine();

switch (ch) {

case 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 pos = sc.nextLine();

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

double sal = sc.nextDouble();

addEmployee(new Employee(id, name, pos, sal));

break;

case 2:

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

int sid = sc.nextInt();

searchEmployee(sid);

break;

case 3:

traverseEmployees();

break;

case 4:

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

int did = sc.nextInt();

deleteEmployee(did);

break;

case 5:

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

return;

default:

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

}

}

}

}

**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 :**

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 "Task ID: " + taskId + ", Name: " + taskName + ", Status: " + status;

}

}

class TaskList {

Task head;

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

Task newTask = new Task(id, name, status);

if (head == null) {

head = newTask;

} else {

Task current = head;

while (current.next != null) {

current = current.next;

}

current.next = newTask;

}

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

}

public void searchTask(int id) {

Task current = head;

while (current != null) {

if (current.taskId == id) {

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

return;

}

current = current.next;

}

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

}

public void displayTasks() {

if (head == null) {

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

return;

}

Task current = head;

System.out.println("All Tasks:");

while (current != null) {

System.out.println(current);

current = current.next;

}

}

public 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 current = head;

while (current.next != null) {

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

current.next = current.next.next;

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

return;

}

current = current.next;

}

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

}

}

public class TaskManagementSystem {

public static void main(String[] args) {

TaskList taskList = new TaskList();

taskList.addTask(1, "Design UI", "Pending");

taskList.addTask(2, "Implement Backend", "In Progress");

taskList.addTask(3, "Test Features", "Pending");

taskList.displayTasks();

taskList.searchTask(2);

taskList.deleteTask(2);

taskList.searchTask(2);

taskList.displayTasks();

}

}

**Output :**

Task added.

Task added.

Task added.

All Tasks:

Task ID: 1, Name: Design UI, Status: Pending

Task ID: 2, Name: Implement Backend, Status: In Progress

Task ID: 3, Name: Test Features, Status: Pending

Found: Task ID: 2, Name: Implement Backend, Status: In Progress

Task deleted.

Task not found.

All Tasks:

Task ID: 1, Name: Design UI, Status: Pending

Task ID: 3, Name: Test Features, Status: Pending

**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 :**

import java.util.\*;

class Book {

int bookId;

String title;

String author;

Book(int 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 class LibraryManagementSystem {

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

for (Book book : books) {

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

return book;

}

}

return null;

}

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

int low = 0, high = books.size() - 1;

while (low <= high) {

int mid = (low + high) / 2;

int cmp = books.get(mid).title.compareToIgnoreCase(title);

if (cmp == 0)

return books.get(mid);

else if (cmp < 0)

low = mid + 1;

else

high = mid - 1;

}

return null;

}

public static void main(String[] args) {

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

library.add(new Book(1, "The Hobbit", "J.R.R. Tolkien"));

library.add(new Book(2, "Clean Code", "Robert C. Martin"));

library.add(new Book(3, "Effective Java", "Joshua Bloch"));

library.add(new Book(4, "The Pragmatic Programmer", "Andy Hunt"));

System.out.println("== Library ==");

for (Book book : library) {

System.out.println(book);

}

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

String titleToSearch = "Clean Code";

Book foundLinear = linearSearch(library, titleToSearch);

if (foundLinear != null)

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

else

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

library.sort(Comparator.comparing(book -> book.title.toLowerCase()));

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

Book foundBinary = binarySearch(library, titleToSearch);

if (foundBinary != null)

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

else

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

}

}

**Output :**

== Library ==

Book ID: 1, Title: The Hobbit, Author: J.R.R. Tolkien

Book ID: 2, Title: Clean Code, Author: Robert C. Martin

Book ID: 3, Title: Effective Java, Author: Joshua Bloch

Book ID: 4, Title: The Pragmatic Programmer, Author: Andy Hunt

== Linear Search ==

Found: Book ID: 2, Title: Clean Code, Author: Robert C. Martin

== Binary Search ==

Found: Book ID: 2, Title: Clean Code, Author: Robert C. Martin