**WEEK 1: ALGORITHMS \_ DATA STRUCTURES**

**Exercise 1: Inventory Management System**

**CODE:**

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;

}

}

public class Main {

static HashMap<Integer, Product> inventory = new HashMap<>();

public static void addProduct(Product p) {

inventory.put(p.productId, p);

}

public static void updateProduct(int id, int newQty, double newPrice) {

if (inventory.containsKey(id)) {

Product p = inventory.get(id);

p.quantity = newQty;

p.price = newPrice;

}

}

public static void deleteProduct(int id) {

inventory.remove(id);

}

public static void main(String[] args) {

addProduct(new Product(1, "Keyboard", 10, 499.99));

addProduct(new Product(2, "Mouse", 15, 299.99));

updateProduct(1, 20, 479.99);

deleteProduct(2);

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

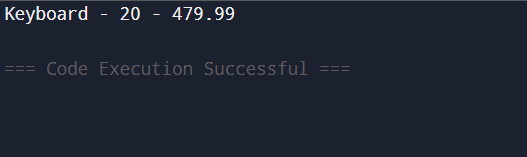
System.out.println(p.productName + " - " + p.quantity + " - " + p.price);

}

}

}

**OUTPUT:**



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

**CODE:**

import java.util.\*;

class Product {

int productId;

String productName;

String category;

public Product(int id, String name, String category) {

this.productId = id;

this.productName = name;

this.category = category;

}

}

public class Main {

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

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

if (products[i].productName.equals(name)) return i;

}

return -1;

}

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

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

while (low <= high) {

int mid = (low + high) / 2;

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

if (cmp == 0) return mid;

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

else high = mid - 1;

}

return -1;

}

public static void main(String[] args) {

Product[] products = {

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

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

new Product(3, "Mouse", "Electronics")

};

int result1 = linearSearch(products, "Mouse");

System.out.println("Linear Search Found at: " + result1);

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

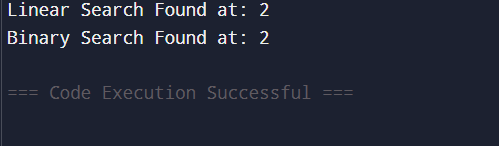
int result2 = binarySearch(products, "Mouse");

System.out.println("Binary Search Found at: " + result2);

}

}

**OUTPUT:**



**Exercise 3: Sorting Customer Orders**

**CODE:**

class Order {

int orderId;

String customerName;

double totalPrice;

public Order(int id, String name, double price) {

this.orderId = id;

this.customerName = name;

this.totalPrice = price;

}

}

public class Main {

public static void bubbleSort(Order[] orders) {

for (int i = 0; i < orders.length - 1; i++) {

for (int j = 0; j < orders.length - 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[] arr, int low, int high) {

if (low < high) {

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

quickSort(arr, low, pi - 1);

quickSort(arr, pi + 1, high);

}

public 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, "Swathi", 250.50),

new Order(2, "Dinakaran", 499.99),

new Order(3, "Anitha", 125.75) };

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

for (Order o : orders) {

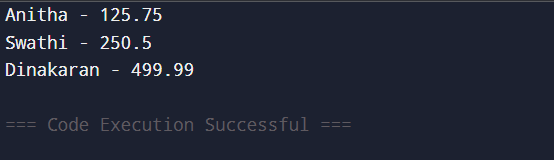
System.out.println(o.customerName + " - " + o.totalPrice);

}

}

}

**OUTPUT:**



**Exercise 4: Employee Management System**

**CODE:**

class Employee {

int employeeId;

String name;

String position;

double salary;

public Employee(int id, String name, String position, double salary) {

this.employeeId = id;

this.name = name;

this.position = position;

this.salary = salary;

}

}

public class Main {

static Employee[] employees = new Employee[5];

static int count = 0;

public static void addEmployee(Employee e) {

if (count < employees.length) {

employees[count++] = e;

}

}

public static void searchEmployee(int id) {

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

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

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

return;

}

}

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

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("Deleted");

return;

}

}

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

public static void traverseEmployees() {

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

System.out.println(employees[i].name + " - " + employees[i].position); }

}

public static void main(String[] args) {

addEmployee(new Employee(1, "Swathi", "Manager", 75000));

addEmployee(new Employee(2, "Dharshu", "Engineer", 60000));

traverseEmployees();

searchEmployee(2);

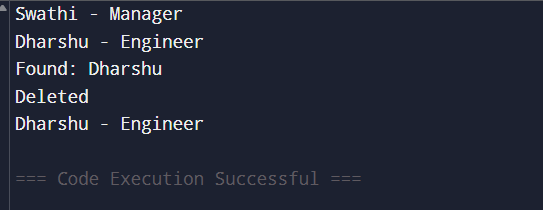
deleteEmployee(1);

traverseEmployees();

}

}

**OUTPUT:**



**Exercise 5: Task Management System**

**CODE:**

class Task {

int taskId;

String taskName;

String status;

Task next;

public Task(int id, String name, String status) {

this.taskId = id;

this.taskName = name;

this.status = status;

this.next = null;

}

}

public class Main {

static Task head = null;

public static void addTask(Task newTask) {

if (head == null) {

head = newTask;

} else {

Task temp = head;

while (temp.next != null) temp = temp.next;

temp.next = newTask;

}

}

public static void searchTask(int id) {

Task temp = head;

while (temp != null) {

if (temp.taskId == id) {

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

return;

}

temp = temp.next;

}

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

}

public static void deleteTask(int id) {

Task temp = head, prev = null;

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

prev = temp;

temp = temp.next;

}

if (temp == null) {

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

return;

}

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

else prev.next = temp.next;

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

}

public static void traverseTasks() {

Task temp = head;

while (temp != null) {

System.out.println(temp.taskName + " - " + temp.status);

temp = temp.next;

}

}

public static void main(String[] args) {

addTask(new Task(1, "Fix Bug", "Pending"));

addTask(new Task(2, "Write Docs", "In Progress"));

traverseTasks();

searchTask(1);

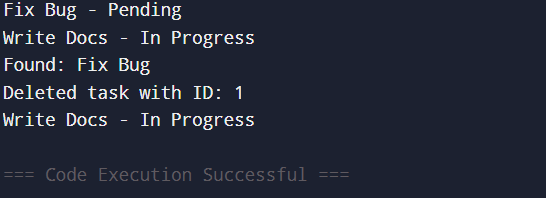
deleteTask(1);

traverseTasks();

}

}

**OUTPUT:**



**Exercise 6: Library Management System**

**CODE:**

import java.util.Arrays;

import java.util.Comparator;

class Book {

int bookId;

String title;

String author;

public Book(int id, String title, String author) {

this.bookId = id;

this.title = title;

this.author = author;

}

}

public class Main {

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

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

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

}

return -1;

}

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

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

while (low <= high) {

int mid = (low + high) / 2;

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

if (cmp == 0) return mid;

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

else high = mid - 1;

}

return -1;

}

public static void main(String[] args) {

Book[] books = {

new Book(1, "Java Basics", "James"),

new Book(2, "Python Guide", "Guido"),

new Book(3, "C++ Primer", "Bjarne")

};

int res1 = linearSearch(books, "Python Guide");

System.out.println("Linear Search Index: " + res1);

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

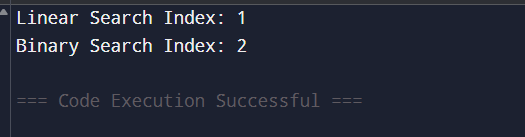
int res2 = binarySearch(books, "Python Guide");

System.out.println("Binary Search Index: " + res2);

}

}

**OUTPUT:**



**Exercise 7: Financial Forecasting**

**CODE:**

public class Main {

// Recursive forecast function

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 current = 10000;

double rate = 0.1; // 10%

int years = 5;

double future = forecast(current, rate, years);

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

}

}

**OUTPUT:**

