WEEK 1

Mandatory Hands-On

**DATA STRUCTURES AND ALGORITHMS:**

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

1. Understanding Asymptotic notations:
2. Explaining Big O notation:

* Big O notation is a mathematical way to describe the upper bound of an algorithm's running time or space usage as the input size grows.
* It tells us how fast or slow an algorithm becomes on increasing the input size.
* It helps compare the efficiency of different algorithms.
* It gives a worst-case estimate, which is helpful in planning for the worst.

1. Best, Average and Worst case scenarios for search operations:

Linear search:

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |

| **Case** | **Description** | **Time Complexity** |
| --- | --- | --- |
| **Best Case** | Element found at the **first index** | O(1) |
| **Average** | Element is in the **middle** or random place | O(n) |
| **Worst Case** | Element is at **last index** or **not found** | O(n) |

| **Case** | **Description** | **Time Complexity** |
| --- | --- | --- |
| **Best Case** | Element is at the **middle index** | O(1) |
| **Average** | Element is in one of the halves | O(log n) |
| **Worst Case** | Element not found after full divide steps | O(log n) |

Binary search:

1. Setup and Implementation:

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 + ")";

}

}

Linear Search:

public class LinearSearch{

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

for (Product product : products) {

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

return product;

}

}

return null;

}

}

Binary Search:

import java.util.Arrays;

import java.util.Comparator;

public class BinarySearch{

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

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

while (left <= right) {

int mid = left + (right - left) / 2;

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

if (cmp == 0) {

return products[mid];

} else if (cmp < 0) {

left = mid + 1;

} else {

right = mid - 1;

}

}

return null;

}

public static void sortProductsByName(Product[] products) {

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

}

}

Main Method:

public class Main {

public static void main(String[] args) {

Product[] products = {

new Product(101, "Laptop", "Electronics"),

new Product(102, "Book", "Stationery"),

new Product(103, "Phone", "Electronics"),

new Product(104, "Shoes", "Fashion")

};

// Linear search

Product result1 = LinearSearch.linearSearchByName(products, "Phone");

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

// Binary search (requires sorted array)

BinarySearch.sortProductsByName(products);

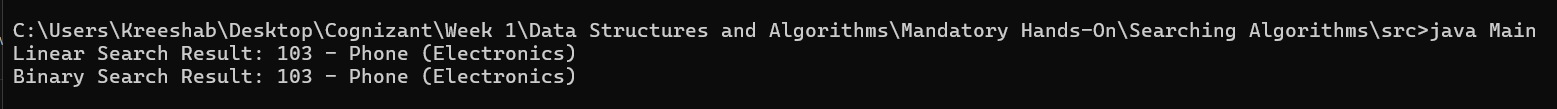
Product result2 = BinarySearch.binarySearchByName(products, "Phone");

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

}

}

OUTPUT:



1. Analysis on both type of Searches:

| **Factor** | **Linear Search** | **Binary Search** |
| --- | --- | --- |
| **Time Complexity** | O(1) Best, O(n) Average & Worst | O(1) Best, O(log n) Average & Worst |
| **Data Format** | Works on **unsorted data** | Requires **sorted data** |
| **Search Logic** | Checks every element one by one | Divides array into halves repeatedly |
| **Performance** | Slower for large data sets | Much faster for large, sorted data |
| **Extra Cost** | No setup needed | Needs sorting (O(n log n)) if unsorted |

Suitable for E-commerce website searching:

Binary Search is more suitable for an e-commerce platform — when the product data is already sorted or can be sorted once and reused.

* E-commerce platforms deal with thousands or millions of products.
* Binary search provides logarithmic time complexity (O(log n)), which is highly efficient for large datasets.
* In a real platform, data is not changing every second — products are listed, updated occasionally.
* Sort once (O(n log n)) and search many times (O(log n)), which is very optimal.
* E-commerce platforms often categorize and index products (e.g., name, brand, price).
* With pre-processing (e.g., sorting, indexing), binary search becomes ideal.