

Name: Amogh Girish Nagarkar

Superset ID: 6403503

DN-4.0 - Java FSE - Deep Skilling

Exercise 2 – E-commerce Platform Search Function

1. Understanding Asymptotic Notation

Big O notation describes how the running time (or memory use) of an algorithm grows as the input size (n) grows. It lets us compare different algorithms.

- Best case – the fastest a search can finish (example: the item is in the first position).
- Average case – the time we expect on random data.
- Worst case – the slowest time (example: item is last or not present).

2. Setup

Product.java

```
package E_commerce;

public class Product {
    String productId;
    String productName;
    String category;

    public Product(String id, String name, String category) {
        this.productId = id;
        this.productName = name;
        this.category = category;
    }

    @Override
    public String toString() {
        return productId + " | " + productName + " | " + category;
    }
}
```

3. Implementation

Main.java

```
package E_commerce;

public class Main {

    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 start = 0;
        int end = products.length - 1;

        while (start <= end) {
            int mid = (start + end) / 2;
            int result = products[mid].productName.compareToIgnoreCase(name);

            if (result == 0) {
                return products[mid];
            } else if (result < 0) {
                start = mid + 1;
            } else {
                end = mid - 1;
            }
        }

        return null;
    }

    public static void main(String[] args) {

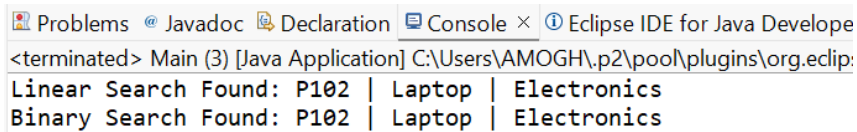
        Product[] products = {
            new Product("P101", "Fan", "Home"),
            new Product("P102", "Laptop", "Electronics"),
            new Product("P103", "Mobile", "Electronics"),
            new Product("P104", "Shirt", "Clothing"),
            new Product("P105", "Watch", "Accessories")
        };

        Product result1 = linearSearch(products, "Laptop");
        if (result1 != null) {
            System.out.println("Linear Search Found: " + result1);
        } else {
            System.out.println("Linear Search: Not found");
        }

        Product result2 = binarySearch(products, "Laptop");
        if (result2 != null) {
            System.out.println("Binary Search Found: " + result2);
        } else {
            System.out.println("Binary Search: Not found");
        }
    }
}

```

4. Output



The screenshot shows the Eclipse IDE interface with the 'Console' tab selected. The output displays the results of two searches performed on a list of products. The first search, 'Linear Search', finds the product 'P102 | Laptop | Electronics'. The second search, 'Binary Search', also finds the same product. The console output is as follows:

```

<terminated> Main (3) [Java Application] C:\Users\AMOGH\p2\pool\plugins\org.eclipse
Linear Search Found: P102 | Laptop | Electronics
Binary Search Found: P102 | Laptop | Electronics

```

5. Analysis

In linear search, the program checks each product one by one until it finds a match.

If the item is at the beginning, it is found quickly. But if it's at the end or not present, it takes longer.

So, in the best case, it's very fast ($O(1)$), but in the worst case, it takes time equal to the number of items ($O(n)$).

In binary search, the list must already be sorted.

The program keeps cutting the list in half, so it finds the item much faster, even if there are thousands or millions of products.

Its speed grows slowly even as data grows, with a time complexity of $O(\log n)$.

Which is better?

If list is small or rarely searched, linear search is easier to use.

If product list is large and if we want fast results, binary search is better (list must be sorted).