**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:

* Big O notation is used to analyse the performance of an algorithm based on the change in size of the input.
* Big O notation helps in analysing the algorithms by comparing with different algorithms, finding the efficiency for large applications such as an E-commerce platform and understand the efficiency in the working of the algorithm with the change in the input size.

**Product.java**

**package** search;

**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;

}

}

**SearchFunc.java**

**package** search;

**public** **class** SearchFunc {

**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** left = 0;

**int** right = products.length - 1;

**while** (left <= right) {

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

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

**if** (cmp == 0) **return** products[mid];

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

**else** right = mid - 1;

}

**return** **null**;

}

}

**Main.java**

**package** search;

**import** java.util.Arrays;

**import** java.util.Comparator;

**public** **class** Main {

**public** **static** **void** main(String[] args) {

Product[] products = {

**new** Product(1, "Mobile", "Electronics"),

**new** Product(2, "Shampoo", "Haircare"),

**new** Product(3, "Chair", "Furniture"),

**new** Product(4, "Phone", "Electronics")

};

// Linear Search

**long** startTime = System.*nanoTime*();

Product foundProduct = SearchFunc.*linearSearch*(products, "Phone");

**long** endTime = System.*nanoTime*();

**long** linearSearchTime = endTime - startTime;

**if** (foundProduct != **null**)

System.***out***.println("Linear Search Found: " + foundProduct.productName);

**else**

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

System.***out***.println("Time taken for Linear Search: " + linearSearchTime + " ns");

// Sort elements

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

// Binary Search

startTime = System.*nanoTime*();

foundProduct = SearchFunc.*binarySearch*(products, "Phone");

endTime = System.*nanoTime*();

**long** binarySearchTime = endTime - startTime;

**if** (foundProduct != **null**)

System.***out***.println("Binary Search Found: " + foundProduct.productName);

**else**

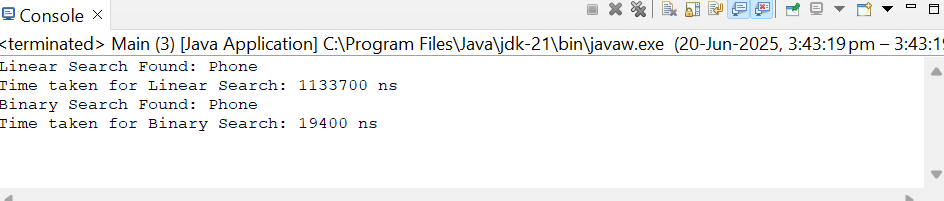
System.***out***.println("Product not found");

System.***out***.println("Time taken for Binary Search: " + binarySearchTime + " ns");

}

}

**OUTPUT:**



For an e-commerce platform, binary search is more suitable for better performance as it contains large data.

**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:

* Recursion is a method where the function calls itself repeatedly until a base case is satisfied.
* Recursion helps in breaking down complex problems into smaller sub problems making the code less complex to understand.

**FF.java**

**package** forecastTool;

**public** **class** FF {

**public** **static** **double** GrowthRate(**double** principal, **double**[] growthRates) {

**for** (**double** rate : growthRates) {

principal \*= (1 + rate);

}

**return** principal;

}

**public** **static** **double** GrowthRatesRec(**double** principal, **double**[] growthRates, **int** year) {

**if** (year == growthRates.length) {

**return** principal;

}

**return** *GrowthRatesRec*(principal \* (1 + growthRates[year]), growthRates, year + 1);

}

}

**Main.java**

**package** forecastTool;

**public** **class** Main {

**public** **static** **void** main(String[] args) {

**double** principal = 10000;

**double**[] growthRates = {0.05, 0.03, -0.02, 0.04, 0.06};

**long** startIter = System.*nanoTime*();

**double** futureValIter = FF.*GrowthRate*(principal, growthRates);

**long** endIter = System.*nanoTime*();

**long** timeIter = endIter - startIter;

**long** startRec = System.*nanoTime*();

**double** futureValRec = FF.*GrowthRatesRec*(principal, growthRates, 0);

**long** endRec = System.*nanoTime*();

**long** timeRec = endRec - startRec;

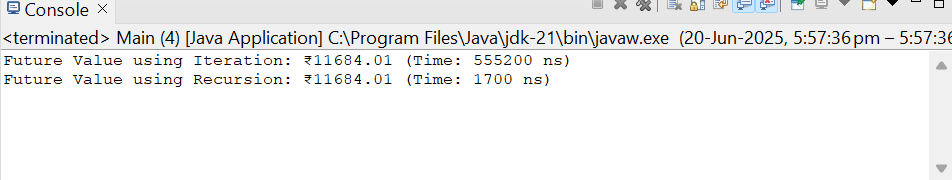
System.***out***.printf("Future Value using Iteration: ₹%.2f (Time: %d ns)%n", futureValIter, timeIter);

System.***out***.printf("Future Value using Recursion: ₹%.2f (Time: %d ns)%n", futureValRec, timeRec);

}

}

**OUTPUT:**

****

* The time complexity of recursion is O(n).
* Can optimize using tail recursion or iterative method for linear sequential problems.