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

1. Explain Big O notation and how it helps in analyzing algorithms.

* **Definition:** Big O notation is a mathematical notation that describes the upper bound of the time complexity of an algorithm. It helps us understand how the runtime of an algorithm grows as the size of the input increases.
* **Purpose:** It provides a high-level understanding of the algorithm's efficiency and performance in terms of time and space.

1. Describe the best, average, and worst-case scenarios for search operations.

* **Best Case:** The minimum time an algorithm takes to complete. For example, in a linear search, the best case is when the target element is the first element in the array O(1).
* **Average Case:** The expected time an algorithm takes to complete, assuming the input is random. For linear search, it’s O(n/2), which simplifies to O(n).
* **Worst Case:** The maximum time an algorithm takes to complete. For linear search, the worst case is when the target element is the last element or not present at all O(n).

1. Create a class **Product** with attributes for searching, such as **productId, productName**, and **category**.

import java.util.ArrayList;  
  
public class Product {  
 private int productId;  
 private String productName;  
 private String category;  
  
 public Product( int productId, String productName, String category) {  
 this.productName = productName;  
 this.productId = productId;  
 this.category = category;  
 }  
  
 public int getProductId() {  
 return productId;  
 }  
  
 public void setProductId(int productId) {  
 this.productId = productId;  
 }  
  
 public String getCategory() {  
 return category;  
 }  
  
 public void setCategory(String category) {  
 this.category = category;  
 }  
  
 public String getProductName() {  
 return productName;  
 }  
  
 public void setProductName(String productName) {  
 this.productName = productName;  
 }  
  
 public String toString(){  
 return this.productId + " " + this.productName + " " + this.category;  
 }  
}

1. Implement linear search and binary search algorithms.

public class Search {

public Product linearSearch(Product[] products,int id){

for(Product product : products){

if(product.getProductId() == (id))

return product;

}

return null;

}

public Product binarySearch(Product[] sortedArray,int id){

int left = 0;

int right = sortedArray.length - 1;

int mid = -1;

while (left <= right){

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

int currId = sortedArray[mid].getProductId();

if(currId == id)

return sortedArray[mid];

else if(currId < id)

left = mid + 1;

else

right = mid - 1;

}

return null;

}

}

1. Store products in an array for linear search and a sorted array for binary search.

public class Main {  
 public static void main(String[] args) {  
  
 Search search = new Search();  
  
 Product[] productsArray = new Product[3];  
  
 productsArray[0] = new Product(101,"Poco X3","Mobile");  
 productsArray[1] = new Product(102,"Lenovo Thinkbook 15","Laptop");  
 productsArray[2] = new Product(100,"Mi Wired Earphones","Earphones");  
  
 Product[] sortedProductsArray = new Product[3];  
  
 sortedProductsArray[0] = new Product(100,"Poco X3","Mobile");  
 sortedProductsArray[1] = new Product(101,"Lenovo Thinkbook 15","Laptop");  
 sortedProductsArray[2] = new Product(102,"Mi Wired Earphones","Earphones");  
  
 System.*out*.println("Search Using Linear Search");  
 System.*out*.println(search.linearSearch(productsArray,101));  
  
 System.*out*.println("Search Using Binary Search");  
 System.*out*.println(search.binarySearch(sortedProductsArray,102));  
  
 }  
}

1. Time Complexity

* **Linear Search:**
  + **Best Case:** O(1)
  + **Average Case:** O(n)
  + **Worst Case:** O(n)
* **Binary Search:**
  + **Best Case:** O(1)
  + **Average Case:** O(log⁡n)
  + **Worst Case:** O(log⁡n)

1. Comparison and Suitability

* **Linear Search:**
  + Simple and does not require the array to be sorted.
  + Suitable for small datasets or unsorted data.
* **Binary Search:**
  + Requires the array to be sorted.
  + Much faster for large datasets due to its logarithmic time complexity.
  + More suitable for an e-commerce platform where the product list can be pre-sorted and fast search performance is critical.