DSA-2

1. Asymptotic Notation also known as the Big O Notation gives an upper bound on the growth rate of an algorithm’s running time (or space) as the input size *n* grows and it also lets us compare algorithms abstractly ignoring machine-specific constants.

Linear Search -> Best Case (O(1)), Average Case (O(n)), Worst Case (O(n))

Binary Search -> Best Case (O(1)), Average Case (O(logn)), Worst Case (O(logn))

1. //Product.java

package DSA2;

public class Product {

    private int productId;

    private String productName;

    private String category;

    public Product(int productId, String productName, String category){

       this.productId = productId;

       this.productName = productName;

       this.category = category;

    }

    public int getProductId(){

        return productId;

    }

    public String getProductName(){

        return productName;

    }

    public String getCategory(){

        return category;

    }

    public String toString(){

        return String.format("Product[%d, %s, %s]", productId, productName, category);

    }

}

1. //ProductLinearSearch.java

package DSA2;

public class ProductLinearSearch {

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

        for(Product p : products){

            if(p.getProductId() == Id){

                return p;

            }

        }

        return null;

    }

    public static void main(String[] args){

        Product[] products = {

            new Product(1, "laptop", "electronics"),

            new Product(2, "mobile", "electronics"),

            new Product(3, "t-shirt", "home"),

            new Product(4, "pen", "stationery")

        };

        Product linear = linearSearch(products, 1);

        System.out.println("linear search output -> " + linear);

    }

}

//ProductBinarySearch.java

package DSA2;

import java.util.Arrays;

import java.util.Comparator;

public class ProductBinarySearch {

    public static Product binarySearch(Product[] sortedProducts, int Id){

        int low = 0;

        int high = sortedProducts.length - 1;

        while(low <= high){

            int mid = low + (high - low)/2;

            int midId = sortedProducts[mid].getProductId();

            if(midId == Id){

                return sortedProducts[mid];

            }

            else if (midId < Id){

                low = mid + 1;

            }

            else {

                high = mid - 1;

            }

        }

        return null;

    }

    public static void main(String[] args){

        Product[] products = {

            new Product(1, "laptop", "electronics"),

            new Product(2, "mobile", "electronics"),

            new Product(3, "t-shirt", "home"),

            new Product(4, "pen", "stationery")

        };

        Product[] sorted = Arrays.copyOf(products, products.length);

        Arrays.sort(sorted, Comparator.comparingInt(Product::getProductId));

        Product binary = binarySearch(sorted, 3);

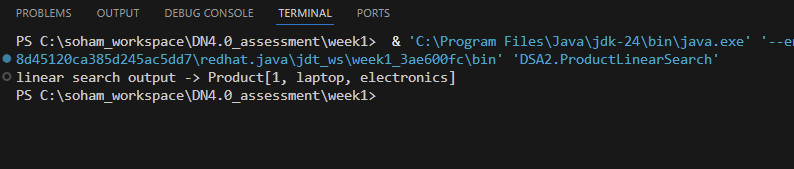
        System.out.println("binary search output -> " + binary);

    }

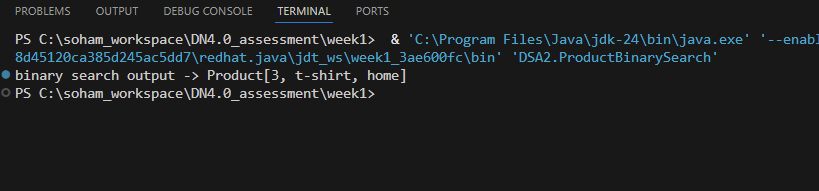
}

//Output

//Linear Search



//Binary Search



1. Time Complexity Comparison

Linear Search -> Best (O(1)), Average/Worst (O(n))

Binary Search -> Best/Average/Worst (O(logn))

Linear needs no extra space or setup but Binary Seach maintains a permanently sorted array

As the catalogue is small linear search is faster and simpler but for larger databases binary search or more complex structures can be implemented.