Week1-Algorithms and Data Structures

**Exercise 1: Inventory Management System**

**Scenario:**

You are developing an inventory management system for a warehouse. Efficient data storage and retrieval are crucial.

**Steps:**

1. **Understand the Problem:**
   * Explain why data structures and algorithms are essential in handling large inventories.
   * Discuss the types of data structures suitable for this problem.
2. **Setup:**
   * Create a new project for the inventory management system.
3. **Implementation:**
   * Define a class Product with attributes like **productId**, **productName**, **quantity**, and **price**.
   * Choose an appropriate data structure to store the products (e.g., ArrayList, HashMap).
   * Implement methods to add, update, and delete products from the inventory.
4. **Analysis:**
   * Analyze the time complexity of each operation (add, update, delete) in your chosen data structure.
   * Discuss how you can optimize these operations.

Code:

1. Creating the Product class having features productid,productName,quantity,price.

Product.java

*// @ this is just the Product class having the product features defined in it....//*

*public* *class* Product {

*// we have the atributes like productId ,productName, quantity,price*

*private* *String* productId;

*private* *String* productName;

*private* *int* quantity;

*private* *double* price;

*public* Product(*String* productId,*String* productName, *int* quantity,*double* price){

*this*.*productId*=productId;

*this*.*productName*=productName;

*this*.*quantity*=quantity;

*this*.*price*=price;

   }

*// # all the getter methods*

*public* *String* getProductName() {

    return productName;

   }

*public* *int* getQuantity() {

    return quantity;

   }

*public* *double* getPrice() {

    return price;

   }

*public* *String* getProductId() {

    return productId;

   }

*// # all the setter methods*

*public* *void* setProductId(*String* productId) {

*this*.*productId* = productId;

   }

*public* *void* setProductName(*String* productName) {

*this*.*productName* = productName;

   }

*public* *void* setQuantity(*int* quantity) {

*this*.*quantity* = quantity;

   }

*public* *void* setPrice(*double* price) {

*this*.*price* = price;

   }

  @*Override*

*public* *String* toString(){

      return *String*.format("Product[\nId-%s\nName-%s\nQuantity-%d units\nPrice-$%.3f\n]",productId,productName,quantity,price);

   }

}

1. Creating the Inventory class having the HashMap data structure which stores the Products in the list.

Inventory.java

*// # this will be the inventory class which will store the Products and have functionalities like adding,deleting and updating them in the inventory.*

*import* *java.util.HashMap*;

*public* *class* Inventory {

*private* *static* *final* *HashMap*<*String*,*Product*> list;

*static*{

        list=new *HashMap*<*String*,*Product*>();

    }

*// \* now we will write the behavioral functions of the Inventory like add,delete,update..*

*// @ add function which will add products to our HashMap list*

*public*  *void* addProduct(*Product* product) {

        if(!*list*.containsValue(product)){

*list*.put(*product*.getProductId(), product);

*System*.*out*.println(*String*.format("the product %s has been added to the inventory..",*product*.getProductName()));

        }

        else{

*System*.*out*.println(*String*.format("the product %s already exists in the inventory..", *product*.getProductName()));

        }

    }

*// @ delete function which will delete products from our HashMap list*

*public*  *void* deleteProduct(*String* productId) {

        if(*list*.containsKey(productId)){

*list*.remove(productId);

*System*.*out*.println(*String*.format("the pr0duct %s has been deleted from the inventory",*list*.get(productId).getProductName()));

        }

        else{

*System*.*out*.println(*String*.format("the product with id %s is not in the inventory",productId));

        }}

*// @ update function which will update the features of a product in out list*

*public* *void* updateProduct(*String* productId,*String* productName,*int* quantity,*double* price)

    {

*Product* product = null;

        if(*list*.containsKey(productId)){

             product=*list*.get(productId);

        }

        if(product==null){

             return;

        }

        if(*product*.getProductName()!=null){

*product*.setProductName(productName);

        }

*product*.setQuantity(quantity);

*product*.setPrice(price);

*System*.*out*.println(*String*.format("the product %s has been updated.",productName));

    }

*// ! display method to display the whole list to see the inventory..*

*public* *void* displayInventory()

    {

*System*.*out*.println("the inventory is: ");

        for(*Product* p:*list*.values()){

*System*.*out*.println(p);

*System*.*out*.println();

        }}}

1. Creating the InventoryManagementSystem class implementing the Product and Inventory classes.

InventoryManagementSystem.java

*public* *class* InventoryManagementSystem {

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

*Product* p1=new Product("1001","Reynolds",90,32.78);

*Product* p2=new Product("1002","Pepsi",80,25.00);

*Product* p3=new Product("1003","Coca Cola",100,30.00);

*Inventory* inventory=new Inventory();

*inventory*.addProduct(p1);

*inventory*.addProduct(p2);

*inventory*.addProduct(p3);

*inventory*.displayInventory();

    }

Output:



**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.

Code:

1. Code for the linear search of products :

 // # Search using linear search for the Inventory

    public Product searchLinear(*int* *productId*){

        if(inventory.containsKey(*productId*)){

            System.out.println(String.format("Id-%d Name-%s present in the inventory...\n",*productId*,inventory.get(*productId*).getProductName()));

            return inventory.get(*productId*);

        }

        return null;

    }

1. Code for the binary search of products:

// Search the inventory using binary search

public *void* searchBinary(*int* *productId*){

        ArrayList<Product> list=**new** ArrayList<>(inventory.values());

        Collections.sort(list);

*int* s=0;

*int* l=list.size();

        while(s<l){

*int* mid=(s+l)/2;

*int* productIdmid=list.get(mid).getProductId();// # this is the middle product id

            if(*productId*>productIdmid){

                s=mid+1;

            }

            else if(*productId*<productIdmid){

                l=mid-1;

            }

            else if(productIdmid==*productId*){

                System.out.println(String.format("Id-%d Name-%s present in the inventory...\n",*productId*,list.get(mid).getProductName()));

                 return;

            }}}

1. The InventoryManagementSystem code:
2. public class InventoryManagementSystem {
3. public static *void* main(String *args*[]){
4. Product p1=**new** Product(1001,"Reynolds",90,32.78);
5. Product p2=**new** Product(1002,"Pepsi",80,25.00);
6. Product p3=**new** Product(1003,"Coca Cola",100,30.00);
7. Product p4=**new** Product(1000,"Bisleri",200,30.00);
9. Inventory inventory=**new** Inventory();
10. inventory.addProduct(p1);
11. inventory.addProduct(p2);
12. inventory.addProduct(p3);
13. inventory.addProduct(p4);
15. //  inventory.displayInventory();
16. Product p=inventory.searchLinear(1002);
17. System.out.println(p);
18. inventory.searchBinary(1003);
19. }
20. }

Output:

