**DATA STRUCTURES AND ALGORITHMS**

**Exercise 1: Inventory Management System**

**1. Why are data structures and algorithms essential?**

**- Efficient data structures (like lists or maps) help in fast product lookup, inventory updates, and stock tracking.**

**- Algorithms help to maintain performance as the inventory grows (e.g., avoid slow loops when searching for products).**

**Suitable data structures:**

**- List<Product>: For small inventories with simple operations.**

**- Dictionary<int, Product>: For quick lookup using productId.**

using System;

using System.Collections.Generic;

namespace InventoryManagementSystem

{

// Class to represent a product

public class Product

{

public int ProductId;

public string ProductName;

public int Quantity;

public double Price;

public Product(int id, string name, int qty, double price)

{

ProductId = id;

ProductName = name;

Quantity = qty;

Price = price;

}

public void Display()

{

Console.WriteLine($"ID: {ProductId}, Name: {ProductName}, Qty: {Quantity}, Price: Rs {Price}");

}

}

class Program

{

// Using Dictionary for fast access using ProductId

static Dictionary<int, Product> inventory = new Dictionary<int, Product>();

static void AddProduct()

{

Console.Write("Enter Product ID: ");

int id = Convert.ToInt32(Console.ReadLine());

if (inventory.ContainsKey(id))

{

Console.WriteLine("Product with this ID already exists.");

return;

}

Console.Write("Enter Product Name: ");

string name = Console.ReadLine();

Console.Write("Enter Quantity: ");

int qty = Convert.ToInt32(Console.ReadLine());

Console.Write("Enter Price: ");

double price = Convert.ToDouble(Console.ReadLine());

Product p = new Product(id, name, qty, price);

inventory.Add(id, p);

Console.WriteLine("Product added successfully.");

}

static void UpdateProduct()

{

Console.Write("Enter Product ID to update: ");

int id = Convert.ToInt32(Console.ReadLine());

if (!inventory.ContainsKey(id))

{

Console.WriteLine("Product not found.");

return;

}

Console.Write("Enter New Quantity: ");

inventory[id].Quantity = Convert.ToInt32(Console.ReadLine());

Console.Write("Enter New Price: ");

inventory[id].Price = Convert.ToDouble(Console.ReadLine());

Console.WriteLine("Product updated successfully.");

}

static void DeleteProduct()

{

Console.Write("Enter Product ID to delete: ");

int id = Convert.ToInt32(Console.ReadLine());

if (inventory.Remove(id))

Console.WriteLine("Product deleted.");

else

Console.WriteLine("Product not found.");

}

static void ShowAllProducts()

{

Console.WriteLine("\nInventory:");

foreach (var item in inventory.Values)

{

item.Display();

}

}

static void Main(string[] args)

{

while (true)

{

Console.WriteLine("\nInventory Menu:");

Console.WriteLine("1. Add Product");

Console.WriteLine("2. Update Product");

Console.WriteLine("3. Delete Product");

Console.WriteLine("4. Show All Products");

Console.WriteLine("5. Exit");

Console.Write("Enter choice: ");

int choice = Convert.ToInt32(Console.ReadLine());

if (choice == 1)

AddProduct();

else if (choice == 2)

UpdateProduct();

else if (choice == 3)

DeleteProduct();

else if (choice == 4)

ShowAllProducts();

else if (choice == 5)

break;

else

Console.WriteLine("Invalid choice.");

}

}

}

}

**Time Complexity:**

**- Add/Update/Delete in Dictionary: O(1)**

**- Optimization: Use Dictionary for constant-time lookups, and validate inputs before updates.**

**E-commerce Platform Search Function**

* Linear Search: O(n), good for small/unsorted
* Binary Search: O(log n), better if data is sorted

using System;

namespace EcommerceSearchExample

{

// Step 2: Product class

public class Product

{

public int ProductId;

public string ProductName;

public string Category;

public Product(int id, string name, string cat)

{

ProductId = id;

ProductName = name;

Category = cat;

}

public void Display()

{

Console.WriteLine($"ID: {ProductId}, Name: {ProductName}, Category: {Category}");

}

}

class Program

{

static int LinearSearch(Product[] products, string searchName)

{

for (int i = 0; i < products.Length; i++)

{

if (products[i].ProductName.Equals(searchName, StringComparison.OrdinalIgnoreCase))

return i;

}

return -1;

}

static int BinarySearch(Product[] products, string searchName)

{

int left = 0;

int right = products.Length - 1;

while (left <= right)

{

int mid = (left + right) / 2;

int result = string.Compare(products[mid].ProductName, searchName, true);

if (result == 0)

return mid;

else if (result < 0)

left = mid + 1;

else

right = mid - 1;

}

return -1;

}

static void Main(string[] args)

{

Product[] productList = {

new Product(1, "Laptop", "Electronics"),

new Product(2, "Shoes", "Footwear"),

new Product(3, "Mobile", "Electronics"),

new Product(4, "T-Shirt", "Clothing"),

new Product(5, "Watch", "Accessories")

};

Console.WriteLine("Choose Search Type:");

Console.WriteLine("1. Linear Search");

Console.WriteLine("2. Binary Search (sorted by name)");

Console.Write("Enter option: ");

int choice = Convert.ToInt32(Console.ReadLine());

Console.Write("Enter Product Name to search: ");

string name = Console.ReadLine();

int index = -1;

if (choice == 1)

{

index = LinearSearch(productList, name);

}

else if (choice == 2)

{

Array.Sort(productList, (a, b) => string.Compare(a.ProductName, b.ProductName, true));

index = BinarySearch(productList, name);

}

else

{

Console.WriteLine("Invalid choice.");

return;

}

if (index != -1)

productList[index].Display();

else

Console.WriteLine("Product not found.");

}

}

}

**Exercise 3: Sorting Customer Orders**

**1. Sorting Algorithms:**

* **Bubble Sort: O(n^2), basic and slow**
* **Insertion Sort: O(n^2), better than bubble for small inputs**
* **Quick Sort: O(n log n), very efficient in most cases**
* **Merge Sort: O(n log n), stable and consistent**

using System;

namespace OrderSortingExample

{

public class Order

{

public int OrderId;

public string CustomerName;

public double TotalPrice;

public Order(int id, string name, double price)

{

OrderId = id;

CustomerName = name;

TotalPrice = price;

}

public void Display()

{

Console.WriteLine($"Order ID: {OrderId}, Customer: {CustomerName}, Total: Rs {TotalPrice}");

}

}

class Program

{

// Bubble Sort

static void BubbleSort(Order[] orders)

{

int n = orders.Length;

for (int i = 0; i < n - 1; i++)

{

for (int j = 0; j < n - i - 1; j++)

{

if (orders[j].TotalPrice > orders[j + 1].TotalPrice)

{

// swap

var temp = orders[j];

orders[j] = orders[j + 1];

orders[j + 1] = temp;

}

}

}

}

// Quick Sort

static void QuickSort(Order[] orders, int low, int high)

{

if (low < high)

{

int pivotIndex = Partition(orders, low, high);

QuickSort(orders, low, pivotIndex - 1);

QuickSort(orders, pivotIndex + 1, high);

}

}

static int Partition(Order[] orders, int low, int high)

{

double pivot = orders[high].TotalPrice;

int i = low - 1;

for (int j = low; j < high; j++)

{

if (orders[j].TotalPrice < pivot)

{

i++;

// swap

var temp = orders[i];

orders[i] = orders[j];

orders[j] = temp;

}

}

// swap pivot

var temp2 = orders[i + 1];

orders[i + 1] = orders[high];

orders[high] = temp2;

return i + 1;

}

static void Main(string[] args)

{

// Sample orders

Order[] orders = {

new Order(101, "Amit", 4500),

new Order(102, "Neha", 1200),

new Order(103, "Rahul", 8200),

new Order(104, "Pooja", 3000)

};

Console.WriteLine("Choose Sorting Method:");

Console.WriteLine("1. Bubble Sort");

Console.WriteLine("2. Quick Sort");

Console.Write("Enter option: ");

int choice = Convert.ToInt32(Console.ReadLine());

if (choice == 1)

{

BubbleSort(orders);

Console.WriteLine("\nSorted using Bubble Sort:");

}

else if (choice == 2)

{

QuickSort(orders, 0, orders.Length - 1);

Console.WriteLine("\nSorted using Quick Sort:");

}

else

{

Console.WriteLine("Invalid choice.");

return;

}

foreach (var order in orders)

{

order.Display();

}

}

}

}

* Bubble Sort: O(n^2), slow for large data
* Quick Sort: O(n log n) average, faster
* Preferred: Quick Sort for e-commerce due to speed
* **Exercise 4: Employee Management System**
* **Array Representation:**
* **Arrays are stored in contiguous memory blocks, which allow fast access via indexing.**
* **Advantage: Fast access (O(1)), minimal overhead**

using System;

namespace EmployeeManagementSystem

{

// Employee class

public class Employee

{

public int EmployeeId;

public string Name;

public string Position;

public double Salary;

public Employee(int id, string name, string position, double salary)

{

EmployeeId = id;

Name = name;

Position = position;

Salary = salary;

}

public void Display()

{

Console.WriteLine($"ID: {EmployeeId}, Name: {Name}, Position: {Position}, Salary: Rs {Salary}");

}

}

class Program

{

// Store employees in an array

static Employee[] employees = new Employee[100]; // Max 100 employees

static int count = 0; // Number of employees currently

static void AddEmployee()

{

if (count >= employees.Length)

{

Console.WriteLine("Employee list is full.");

return;

}

Console.Write("Enter ID: ");

int id = Convert.ToInt32(Console.ReadLine());

Console.Write("Enter Name: ");

string name = Console.ReadLine();

Console.Write("Enter Position: ");

string pos = Console.ReadLine();

Console.Write("Enter Salary: ");

double salary = Convert.ToDouble(Console.ReadLine());

employees[count++] = new Employee(id, name, pos, salary);

Console.WriteLine("Employee added.");

}

static void SearchEmployee()

{

Console.Write("Enter Employee ID to search: ");

int id = Convert.ToInt32(Console.ReadLine());

for (int i = 0; i < count; i++)

{

if (employees[i].EmployeeId == id)

{

employees[i].Display();

return;

}

}

Console.WriteLine("Employee not found.");

}

static void TraverseEmployees()

{

Console.WriteLine("\nEmployee List:");

for (int i = 0; i < count; i++)

{

employees[i].Display();

}

}

static void DeleteEmployee()

{

Console.Write("Enter Employee ID to delete: ");

int id = Convert.ToInt32(Console.ReadLine());

int index = -1;

for (int i = 0; i < count; i++)

{

if (employees[i].EmployeeId == id)

{

index = i;

break;

}

}

if (index == -1)

{

Console.WriteLine("Employee not found.");

return;

}

for (int i = index; i < count - 1; i++)

{

employees[i] = employees[i + 1]; // shift left

}

count--;

Console.WriteLine("Employee deleted.");

}

static void Main(string[] args)

{

while (true)

{

Console.WriteLine("\nEmployee Menu:");

Console.WriteLine("1. Add Employee");

Console.WriteLine("2. Search Employee");

Console.WriteLine("3. Traverse Employees");

Console.WriteLine("4. Delete Employee");

Console.WriteLine("5. Exit");

Console.Write("Enter choice: ");

int choice = Convert.ToInt32(Console.ReadLine());

if (choice == 1)

AddEmployee();

else if (choice == 2)

SearchEmployee();

else if (choice == 3)

TraverseEmployees();

else if (choice == 4)

DeleteEmployee();

else if (choice == 5)

break;

else

Console.WriteLine("Invalid choice.");

}

}

}

}

Time Complexity:

- Add: O(1), Delete/Search: O(n), Traverse: O(n)

- Limitation: Fixed size, resizing is manual, slow deletion

**Exercise 5: Task Management System**

* **Linked Lists:**
* **Singly: Each node points to the next**
* **Doubly: Points to both next and previous**
* **Advantage: Easier insertion/deletion at dynamic positions**

using System;

namespace TaskManagementSystem

{

// Step 2: Task class

public class Task

{

public int TaskId;

public string TaskName;

public string Status;

public Task(int id, string name, string status)

{

TaskId = id;

TaskName = name;

Status = status;

}

public void Display()

{

Console.WriteLine($"ID: {TaskId}, Name: {TaskName}, Status: {Status}");

}

}

// Linked List Node

public class TaskNode

{

public Task TaskData;

public TaskNode Next;

public TaskNode(Task task)

{

TaskData = task;

Next = null;

}

}

class Program

{

static TaskNode head = null;

// Add task at end

static void AddTask()

{

Console.Write("Enter Task ID: ");

int id = Convert.ToInt32(Console.ReadLine());

Console.Write("Enter Task Name: ");

string name = Console.ReadLine();

Console.Write("Enter Task Status (Pending/Done): ");

string status = Console.ReadLine();

Task task = new Task(id, name, status);

TaskNode newNode = new TaskNode(task);

if (head == null)

{

head = newNode;

}

else

{

TaskNode temp = head;

while (temp.Next != null)

temp = temp.Next;

temp.Next = newNode;

}

Console.WriteLine("Task added successfully.");

}

// Search task by ID

static void SearchTask()

{

Console.Write("Enter Task ID to search: ");

int id = Convert.ToInt32(Console.ReadLine());

TaskNode temp = head;

while (temp != null)

{

if (temp.TaskData.TaskId == id)

{

temp.TaskData.Display();

return;

}

temp = temp.Next;

}

Console.WriteLine("Task not found.");

}

// Traverse all tasks

static void TraverseTasks()

{

Console.WriteLine("\nAll Tasks:");

TaskNode temp = head;

if (temp == null)

{

Console.WriteLine("No tasks found.");

return;

}

while (temp != null)

{

temp.TaskData.Display();

temp = temp.Next;

}

}

// Delete task by ID

static void DeleteTask()

{

Console.Write("Enter Task ID to delete: ");

int id = Convert.ToInt32(Console.ReadLine());

if (head == null)

{

Console.WriteLine("List is empty.");

return;

}

if (head.TaskData.TaskId == id)

{

head = head.Next;

Console.WriteLine("Task deleted.");

return;

}

TaskNode current = head;

TaskNode previous = null;

while (current != null && current.TaskData.TaskId != id)

{

previous = current;

current = current.Next;

}

if (current == null)

{

Console.WriteLine("Task not found.");

return;

}

previous.Next = current.Next;

Console.WriteLine("Task deleted.");

}

static void Main(string[] args)

{

while (true)

{

Console.WriteLine("\nTask Menu:");

Console.WriteLine("1. Add Task");

Console.WriteLine("2. Search Task");

Console.WriteLine("3. Traverse Tasks");

Console.WriteLine("4. Delete Task");

Console.WriteLine("5. Exit");

Console.Write("Enter choice: ");

int choice = Convert.ToInt32(Console.ReadLine());

if (choice == 1)

AddTask();

else if (choice == 2)

SearchTask();

else if (choice == 3)

TraverseTasks();

else if (choice == 4)

DeleteTask();

else if (choice == 5)

break;

else

Console.WriteLine("Invalid choice.");

}

}

}

}

* Time Complexity:
* Add: O(n), Search/Delete: O(n), Traverse: O(n)
* Advantage: Dynamic memory use, no resizing needed

**Exercise 6: Library Management System**

using System;

namespace LibraryManagementSystem

{

// Step 2: Book class

public class Book

{

public int BookId;

public string Title;

public string Author;

public Book(int id, string title, string author)

{

BookId = id;

Title = title;

Author = author;

}

public void Display()

{

Console.WriteLine($"ID: {BookId}, Title: {Title}, Author: {Author}");

}

}

class Program

{

// Step 3a: Linear Search

static int LinearSearch(Book[] books, string title)

{

for (int i = 0; i < books.Length; i++)

{

if (books[i].Title.Equals(title, StringComparison.OrdinalIgnoreCase))

return i;

}

return -1;

}

// Step 3b: Binary Search (assumes sorted by title)

static int BinarySearch(Book[] books, string title)

{

int left = 0;

int right = books.Length - 1;

while (left <= right)

{

int mid = (left + right) / 2;

int comparison = string.Compare(books[mid].Title, title, true);

if (comparison == 0)

return mid;

else if (comparison < 0)

left = mid + 1;

else

right = mid - 1;

}

return -1;

}

static void Main(string[] args)

{

// Sample book list

Book[] books = {

new Book(1, "The Alchemist", "Paulo Coelho"),

new Book(2, "Rich Dad Poor Dad", "Robert Kiyosaki"),

new Book(3, "Wings of Fire", "A.P.J. Abdul Kalam"),

new Book(4, "1984", "George Orwell"),

new Book(5, "Ikigai", "Francesc Miralles")

};

Console.WriteLine("Choose Search Type:");

Console.WriteLine("1. Linear Search");

Console.WriteLine("2. Binary Search (sorted by title)");

Console.Write("Enter option: ");

int choice = Convert.ToInt32(Console.ReadLine());

Console.Write("Enter Book Title to search: ");

string title = Console.ReadLine();

int index = -1;

if (choice == 1)

{

index = LinearSearch(books, title);

}

else if (choice == 2)

{

// Sort array before binary search

Array.Sort(books, (a, b) => string.Compare(a.Title, b.Title, true));

index = BinarySearch(books, title);

}

else

{

Console.WriteLine("Invalid option.");

return;

}

if (index != -1)

books[index].Display();

else

Console.WriteLine("Book not found.");

}

}

}

**Exercise 7: Financial Forecasting**

using System;

namespace FinancialForecasting

{

class Program

{

static double PredictValue(double currentValue, double growthRate, int years)

{

if (years == 0)

return currentValue;

return PredictValue(currentValue \* (1 + growthRate), growthRate, years - 1);

}

static void Main(string[] args)

{

Console.Write("Enter current amount (Rs): ");

double current = Convert.ToDouble(Console.ReadLine());

Console.Write("Enter annual growth rate (e.g., 0.10 for 10%): ");

double rate = Convert.ToDouble(Console.ReadLine());

Console.Write("Enter number of years to forecast: ");

int years = Convert.ToInt32(Console.ReadLine());

double futureValue = PredictValue(current, rate, years);

Console.WriteLine($"\n📊 Future value after {years} years: Rs {futureValue:F2}");

}

}

}

* **Time Complexity:**
* Recursive Approach: O(n), where n is the number of years. Each function call processes one year.
* Optimized Approach: O(1), uses direct mathematical formula, very fast.
* **Optimization:**
* Avoid recursion stack overhead by using direct computation.
* Use memoization if intermediate results are reused.
* In this case, Math.Pow() is most efficient.