**Exercise 1: Inventory Management System**

|  |
| --- |
| class Product  {  public int ProductID;  public string ProductName;  public int ProductQuantity;  public double Price;  }  class InventoryManagement  {  public Dictionary<int, Product> Inventory = new Dictionary<int, Product>();  public void add(int productID, Product product)  {  Inventory.Add(productID, product);  }  public void update(int productID)  {  Product temp = Inventory.GetValueOrDefault(productID);  Console.WriteLine($"{temp.ProductID} {temp.ProductName} {temp.ProductQuantity} {temp.Price}");  }  public void delete(int productID)  {  Inventory.Remove(productID);  }  }  public class InventoryManagementSystem  {  public static void Main(string[] args)  {  Product product = new Product();  product.ProductID = 1;  product.ProductName = "Milk";  product.ProductQuantity = 20;  product.Price = 23;  Product product2 = new Product();  product2.ProductID = 2;  product2.ProductName = "Water";  product2.ProductQuantity = 100;  product2.Price = 20;  InventoryManagement inventoryManagement = new InventoryManagement();  inventoryManagement.add(1, product);  inventoryManagement.add(2, product2);  inventoryManagement.update(2);  inventoryManagement.update(1);  inventoryManagement.delete(1);  }  } |

A screenshot of a computer

AI-generated content may be incorrect.

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

|  |
| --- |
| class Product  {  public int Id { get; set; }  public string Name { get; set; }  public string Category { get; set; }  }  class EcommeracePlatformSearch  {  public static void Main(string[] args)  {  //Linear Search  Product[] arr= new Product[4] ;  arr[0]= new Product { Id = 1, Name = "Brush", Category = "Essential" };  arr[1] = new Product { Id = 3, Name = "Paste", Category = "Essential" };  arr[2] = new Product { Id = 2, Name = "Comb", Category = "Essential" };  arr[3] = new Product { Id = 4, Name = "Soap", Category = "Essential" };  int requiredID = 2;  int count = 1;  for(int i =0;i<=3;i++)  {  if(arr[i].Id == requiredID)  {  Console.WriteLine($"Found at {i+1} place in the array and took {count} steps");  }  count++;  }  //Binary Search  Product[] arr2 = new Product[4];  arr2[0] = new Product { Id = 1, Name = "Brush", Category = "Essential" };  arr2[1] = new Product { Id = 2, Name = "Comb", Category = "Essential" };  arr2[2] = new Product { Id = 3, Name = "Paste", Category = "Essential" };  arr2[3] = new Product { Id = 4, Name = "Soap", Category = "Essential" };  int requiredID2 = 2;  int count2 = 1;  int low = 0;  int high = arr2.Length - 1;  while (low <= high)  {  int mid = (low + high) / 2;  if (arr2[mid].Id == requiredID2)  {  Console.WriteLine($"Found at position {mid + 1} in the array and took {count2} steps");  break;  }  else if (arr2[mid].Id < requiredID2)  {  low = mid + 1;  }  else  {  high = mid - 1;  }  count2++;  }  /\*In Linear Search it took 3 Steps to find a element in a array that has 4 elements  for eg we take an array of 1000 elements to calculate Time or Space Complexity we take 3 cases  case 1- Best Case that is the required element is at 1st position so the it requires only one check O(n) = O(1)  case 2- Average Case that is the element is in the 500th position now it requires 500 checks O(n) = O(500)  case 3- Worst Case that is the element is in the last position now it takes 1000 checks O(n) = O(1000)  \*/  /\*In Binary Search it took 1 Steps to find a element in an array that has 4 elements  for eg we take an array of 1000 elements (sorted) to calculate Time or Space Complexity we take 3 cases  case 1- Best Case that is the required element is at the middle position, so it requires only one check O(log n) = O(1)  case 2- Average Case that is the element is in any random position, now it takes around log₂(1000) = 10 checks O(log n) = O(10)  case 3- Worst Case that is the element is at the last possible split, still it takes around log₂(1000) = 10 checks O(log n) = O(10)  \*/  //My Conclusion is to use Binary Search for Larger data and Better performance  }  } |

A screenshot of a computer program

AI-generated content may be incorrect.

**Exercise 3: Sorting Customer Orders**

|  |
| --- |
| using System;  class Order  {  public int orderID;  public string cutomerName;  public double totalPrice;  }  class Quicksort  {  public static int QuickSortWithSteps(Order[] arr, int low, int high, ref int steps)  {  if (low < high)  {  int pivotIndex = Partitioner(arr, low, high, ref steps);  QuickSortWithSteps(arr, low, pivotIndex - 1, ref steps);  QuickSortWithSteps(arr, pivotIndex + 1, high, ref steps);  }  return steps;  }  public static int Partitioner(Order[] arr, int low, int high, ref int steps)  {  int pivot = arr[high].orderID;  steps++; // for pivot assignment  int i = low - 1;  for (int j = low; j < high; j++)  {  steps++; // comparison  if (arr[j].orderID < pivot)  {  i++;  swap(arr, i, j);  steps += 3; // swap counts as 3 steps  }  }  swap(arr, i + 1, high);  steps += 3; // final swap  return i + 1;  }  public static void swap(Order[] arr, int i, int j)  {  Order temp = arr[i];  arr[i] = arr[j];  arr[j] = temp;  }  public static void QuickSort(Order[] arr, int low, int high)  {  int dummy = 0;  QuickSortWithSteps(arr, low, high, ref dummy);  }  }  class BubbleSort  {  public static int Bubblesort(Order[] arr,int n)  {  int i, j, temp,count=0;  Boolean swapped;  for(i=0; i < n; i++)  {  swapped = false;  for (j = 0; j < n - i - 1; j++)  {  count++;  if (arr[j].orderID > arr[j + 1].orderID)  {  temp = arr[j].orderID;  arr[j].orderID = arr[j + 1].orderID;  arr[j + 1].orderID = temp;  swapped = true;  }  }  if (swapped == false) {  break;  }  }  return count;  }  }  class SortingCustomerOrders  {  public static void Main(string[] args)  {  Order[] orders = new Order[5];  orders[0] = new Order { orderID = 105, cutomerName = "Mike", totalPrice = 280.20 };  orders[1] = new Order { orderID = 101, cutomerName = "Shake", totalPrice = 250.0 };  orders[2] = new Order { orderID = 104, cutomerName = "David", totalPrice = 400 };  orders[3] = new Order { orderID = 102, cutomerName = "Harvey", totalPrice = 452.30 };  orders[4] = new Order { orderID = 103, cutomerName = "Dani", totalPrice = 300.75 };  int steps = 0;  int count = Quicksort.QuickSortWithSteps(orders, 0, 4, ref steps);  Console.WriteLine("Sorted Orders by OrderID In Quick Sort:");  Console.WriteLine($"Done it in {count} steps");  foreach (var order in orders)  {  Console.WriteLine($"OrderID: {order.orderID}, Name: {order.cutomerName}, Total: {order.totalPrice}");  }  Order[] orders2 = new Order[5];  orders2[1] = new Order { orderID = 105, cutomerName = "Mike", totalPrice = 280.20 };  orders2[0] = new Order { orderID = 101, cutomerName = "Shake", totalPrice = 250.0 };  orders2[4] = new Order { orderID = 104, cutomerName = "David", totalPrice = 400 };  orders2[3] = new Order { orderID = 102, cutomerName = "Harvey", totalPrice = 452.30 };  orders2[2] = new Order { orderID = 103, cutomerName = "Dani", totalPrice = 300.75 };  int count2 = BubbleSort.Bubblesort(orders2, 5);  Console.WriteLine("Sorted Orders by OrderID In Bubble Sort:");  Console.WriteLine($"Done it in {count2} steps");  foreach (var order in orders)  {  Console.WriteLine($"OrderID: {order.orderID}, Name: {order.cutomerName}, Total: {order.totalPrice}");  }  }  }  /\*Here Bubble Sort take only 9 steps but whereas Quick Sort takes 27 steps because  the bubble sort time complexity is O(n^2) so for 5 elements it takes 9 steps  but the time complexity of quick sort is O(n log n) so here it takes 27 steps  but for longer array like 1000 elements bubble sort takes 1million steps to complete the sorting  whereas in an 1000 element array in quick sort it only takes around 40k - 50k steps  \*/  //Quick sort is preffered because of its performance against larger samples |

A screenshot of a computer program

AI-generated content may be incorrect.

**Exercise 4: Employee Management System**

|  |
| --- |
| using System.Runtime.CompilerServices;  class Employee  {  public int employeeID;  public string name;  public string position;  public double salary;  }  class Array  {  public static void Add(Employee[] arr, Employee emp,ref int count)  {  if (count < arr.Length) {  arr[count] = emp;  count++;  }  else  {  Console.WriteLine("Array is full");  }  }  public static void Search(Employee[] arr, int emp,ref int count)  {  for (int i = 0; i < count; i++)  {  if (arr[i].employeeID == emp)  {  Console.WriteLine($"EmployeeID : {arr[i].employeeID} ; Name : {arr[i].name} ; Position : {arr[i].position} ; Salary : {arr[i].salary}");  }  }  }  public static void Traverse(Employee[] arr,ref int count)  {  for (int i = 0; i < count; i++)  {  Console.WriteLine($"EmployeeID : {arr[i].employeeID} ; Name : {arr[i].name} ; Position : {arr[i].position} ; Salary : {arr[i].salary}");  }  }  public static void Delete(Employee[] arr,int emp,ref int count)  {  Boolean deleted = false;  for (int i = 0; i < count; i++)  {  if (arr[i].employeeID == emp)  {  for (int j = i; j < count; j++)  {  arr[j] = arr[j + 1];  }  arr[count - 1] = null;  count--;  deleted = true;  }    }  if(deleted==false)  {  Console.WriteLine("Employee Not Found");  }  }  }  class EmployeeManagementSystem  {  static Employee[] employees = new Employee[10];  static int count = 0;  public static void Main(string[] args)  {  Array.Add(employees,new Employee { employeeID = 4, name = "Shake", position = "Head of Development", salary = 300000 },ref count);  Array.Add(employees, new Employee { employeeID = 5, name = "David", position = "Developer", salary = 30000 }, ref count);  Array.Add(employees, new Employee { employeeID = 2, name = "Dani", position = "Junior Developer", salary = 50000 }, ref count);  Array.Add(employees, new Employee { employeeID = 7, name = "Ross", position = "Associate", salary = 40000 }, ref count);  Array.Add(employees, new Employee { employeeID = 1, name = "Mike", position = "CEO", salary = 3000000 }, ref count);  Console.WriteLine("Searching For a Employee");  Array.Search(employees, 7,ref count);  Console.WriteLine("Employee List:");  Array.Traverse(employees,ref count);  Array.Delete(employees, 5, ref count);  Console.WriteLine("Employee List after Update:");  Array.Traverse(employees, ref count);  }  }  /\*Let's Calculate the Time Complexity of each function  for add function there is no loops so the time complexity is O(1)  for search function there is a single for loop so the time compelxity is O(n)  for traverse function also there is single for loop so the time complexity is O(n)  for delete function there is a for loop which will run one time and a for loop inside if condtion so atmost it will also execute once  so the time complexity is O(n) + O(n) = O(n)  \*/  // The Limitation of the arrray is the fixed size,deletion of a specific sample is time consuming due to shifting,need a manual pointer  // When to use array is when we know the size is fixed and easier access through index |

A screenshot of a computer

AI-generated content may be incorrect.

**Exercise 5: Task Management System**

|  |
| --- |
| using System;  class Task  {  public int TaskID { get; set; }  public string TaskName { get; set; }  public string Status { get; set; }  public Task(int taskID, string taskName, string status)  {  TaskID = taskID;  TaskName = taskName;  Status = status;  }  public override string ToString()  {  return $"ID: {TaskID,-5} ; Name: {TaskName,-15} ; Status: {Status}";  }  }  class Node  {  public Task Task { get; set; }  public Node Next { get; set; }  public Node(Task task)  {  Task = task;  Next = null;  }  }  class SinglyLinkedList  {  public Node Head { get; private set; }  public SinglyLinkedList()  {  Head = null;  }  public void Add(Task task)  {  Node newnode = new Node(task);  if (Head == null)  {  Head = newnode;  }  else  {  Node current = Head;  while (current.Next != null)  {  current = current.Next;  }  current.Next = newnode;  }  }  public Task Search(int id)  {  if (Head == null)  {  Console.WriteLine("The List is Empty. Cannot search.");  return null;  }  Node current = Head;  while (current != null)  {  if (current.Task.TaskID == id)  {  Console.WriteLine($"Task Found: {current.Task}");  return current.Task;  }  current = current.Next;  }  Console.WriteLine($"Task with ID {id} not found.");  return null;  }  public void Traverse()  {  if (Head == null)  {  Console.WriteLine("The List is Empty.");  return;  }  Node current = Head;  while (current != null)  {  Console.WriteLine(current.Task.ToString());  current = current.Next;  }  }  public bool Delete(int id)  {  if (Head == null)  {  Console.WriteLine("The List is Empty. Cannot delete.");  return false;  }  if (Head.Task.TaskID == id)  {  Head = Head.Next;  Console.WriteLine($"Task ID {id} deleted (was head).");  return true;  }  Node current = Head.Next;  Node previous = Head;  while (current != null)  {  if (current.Task.TaskID == id)  {  previous.Next = current.Next;  Console.WriteLine($"Task ID {id} deleted.");  return true;  }  previous = current;  current = current.Next;  }  Console.WriteLine($"Task with ID {id} not found for deletion.");  return false;  }  }  class TaskManagementSystem  {  public static void Main(string[] args)  {  SinglyLinkedList list = new SinglyLinkedList();  list.Add(new Task(1, "Plan Project", "Pending"));  list.Add(new Task(2, "Develop Module A", "Pending"));  list.Add(new Task(3, "Test Module A", "Pending"));  list.Add(new Task(4, "Deploy Module A", "Pending"));  list.Add(new Task(5, "Develop Module B", "Pending"));  list.Search(3);  Console.WriteLine("List Before Deletion");  list.Traverse();  list.Delete(2);  Console.WriteLine("List After Deletion");  list.Traverse();    }  }  /\* Now Let's calculate time complexity of each operation (assuming only worst case)  for add operation it has only one while loop so the time complexity is O(n)  for also search,delete,traverse function each contain only one loop so the time complexity of every operation is O(n)  \*/  // The Advantage of linked list compared to array is the dynamic size ,efficient deletion and more flexible(even add data at any point) |

A screenshot of a computer

AI-generated content may be incorrect.

**Exercise 6: Library Management System**

|  |
| --- |
| class book  {  public int bookID;  public String bookName;  public String author;  public book(int bookID, String bookName, String author)  {  this.bookID = bookID;  this.bookName = bookName;  this.author = author;  }  }  class LinearSearch  {  public void linearSearch(book[] arr,int id)  {  int count = 0;  for (int i = 0; i < arr.Length; i++) {  if (arr[i].bookID == id) {  Console.WriteLine("Match Found in Linear Search");  Console.WriteLine($"Book ID : {arr[i].bookID} Book Name : {arr[i].bookName} Author : {arr[i].author} in {count} steps");  }  count++;  }  }  }  class BinarySearch  {  public void binarySearch(book[] arr,int id)  {  int low = 0;  int high = arr.Length - 1;  int count2 = 0;  while (low <= high)  {  int mid = (low + high) / 2;  if (arr[mid].bookID == id)  {  Console.WriteLine("Match Found in Binary Search");  Console.WriteLine($"Book ID : {arr[mid].bookID} Book Name : {arr[mid].bookName} Author : {arr[mid].author} in {count2} steps");  break;  }  else if (arr[mid].bookID < id)  {  low = mid + 1;  }  else  {  high = mid - 1;  }  count2++;  }  }  }  class LibraryManagementSystem  {  public static void Main(string[] args)  {  book[] arr1 = new book[6];  arr1[0] = new book(4, "Harry Potter", "J.K Rowling");  arr1[1] = new book(6, "1984", "George Orwell");  arr1[2] = new book(2, "To Kill a Mockingbird", "Harper Lee");  arr1[3] = new book(17, "Pride and Prejudice", "Jane Austen");  arr1[4] = new book(8, "Sapiens: A Brief History of Humankind", "Yuval Noah Harari");  arr1[5] = new book(3, "The Hitchhiker's Guide to the Galaxy", "Douglas Adams");  LinearSearch linear = new LinearSearch();  linear.linearSearch(arr1, 17);  book[] arr2 = new book[6];  arr2[0] = new book(2, "To Kill a Mockingbird", "Harper Lee");  arr2[1] = new book(3, "The Hitchhiker's Guide to the Galaxy", "Douglas Adams");  arr2[2] = new book(4, "Harry Potter", "J.K Rowling");  arr2[3] = new book(6, "1984", "George Orwell");  arr2[4] = new book(8, "Sapiens: A Brief History of Humankind", "Yuval Noah Harari");  arr2[5] = new book(17, "Pride and Prejudice", "Jane Austen");  BinarySearch binary = new BinarySearch();  binary.binarySearch(arr2, 17);  }  }  /\*In Linear Search it took 3 Steps to find a element in a array that has 6 elements  for eg we take an array of 1000 elements to calculate Time or Space Complexity we take 3 cases  case 1- Best Case that is the required element is at 1st position so the it requires only one check O(n) = O(1)  case 2- Average Case that is the element is in the 500th position now it requires 500 checks O(n) = O(500)  case 3- Worst Case that is the element is in the last position now it takes 1000 checks O(n) = O(1000)  \*/  /\*In Binary Search it took 2 Steps to find a element in an array that has 6 elements  for eg we take an array of 1000 elements (sorted) to calculate Time or Space Complexity we take 3 cases  case 1- Best Case that is the required element is at the middle position, so it requires only one check O(log n) = O(1)  case 2- Average Case that is the element is in any random position, now it takes around log₂(1000) = 10 checks O(log n) = O(10)  case 3- Worst Case that is the element is at the last possible split, still it takes around log₂(1000) = 10 checks O(log n) = O(10)  \*/  //For smaller data sample and for less complexity use linear search if the data is huge just use binary search |

A screenshot of a computer

AI-generated content may be incorrect.

**Exercise 7: Financial Forecasting**

|  |
| --- |
| class FinancialForecast  {  public static int CalculateFuture(int currentAmount,float growthRate,int numberofyears)  {  if (numberofyears == 0)  {  return currentAmount;  }  else  {  currentAmount = (int)(currentAmount + (growthRate/100 \* currentAmount));  numberofyears--;  return CalculateFuture(currentAmount, growthRate, numberofyears);  }    }  public static void Main(string[] args)  {  int currentAmount = 1000;  float growthRate = 13.3f;  int numberofyears = 4;  int finalamount = CalculateFuture(currentAmount,growthRate,numberofyears);  Console.WriteLine(finalamount);  }  }  //The time complexity of this function will be O(n) because there is no loop  //To optimize it simply return the function so the memory will be used and will be cleared after computation |

A screen shot of a computer

AI-generated content may be incorrect.