Data Structure and Algorithms (JAVA)



**2nd Lab**

**Semester: Spring 2025**

**Software Engineering**

**Faculty of Information Technology, UCP Lahore, Pakistan**

Table of Contents

[Objective 3](#_Toc193023826)

[Session 1: Common Operations and Complexity Analysis 4](#_Toc193023827)

[1.1 Insert Operation 4](#_Toc193023828)

[1.2 Delete Operation 5](#_Toc193023829)

[Session 2: Array Operations 6](#_Toc193023830)

[2.1 Traversing an Array 6](#_Toc193023831)

[2.2 Searching in an Array 7](#_Toc193023832)

[Session 3: Static & Dynamic Arrays 7](#_Toc193023833)

[3.1 Static Arrays 7](#_Toc193023834)

[3.2 Dynamic Arrays 7](#_Toc193023835)

[4. ArrayList: 9](#_Toc193023836)

[4.1 Insertion in ArrayList 9](#_Toc193023837)

[4.2 Deletion in ArrayList 9](#_Toc193023838)

[4.3 Searching in an ArrayList 10](#_Toc193023839)

[Basic Practice Task 11](#_Toc193023840)

[Scenario-Based Case Studies 11](#_Toc193023841)

[Case Study 1: Number List Operations 11](#_Toc193023842)

[Case Study 2: Student Records System 11](#_Toc193023843)

[Conclusion 11](#_Toc193023844)

**Lab Manual: Data Structures and Algorithms using Java**

### **Objective**

Students will gain hands-on experience in performing basic and advanced operations on arrays, understanding their time complexities, and implementing static and dynamic arrays.

By the end of this lab, students will be able to:

* **Understand the fundamentals of lists and arrays** and their role in Java collections.
* **Perform common array operations**, such as **traversal, searching, insertion, and deletion**.
* **Analyze time complexity** for different operations on arrays and lists.
* **Differentiate between static and dynamic arrays**, understanding their advantages and use cases.
* **Implement array-based operations using object-oriented programming principles** without using static methods.
* **Apply learned concepts to solve real-world problems** through structured exercises and case studies.

This lab serves as the foundation for **efficient data structure usage** in problem-solving and further exploration of Java collections.

## **Session 1: Common Operations and Complexity Analysis**

### **1.1 Insert Operation**

#### **Code Example: Inserting an Element in an Array:**

import java.util.Arrays;  
class ArrayHandler {  
 private int[] arr;  
 private int size;  
  
 public ArrayHandler(int[] inputArr) {  
 this.arr = inputArr;  
 this.size = inputArr.length;  
 }  
 public void insertAtPosition(int element, int pos) {  
 int[] newArr = new int[size + 1];  
 for (int i = 0, j = 0; i < newArr.length; i++) {  
 if (i == pos) {  
 newArr[i] = element;  
 } else {  
 newArr[i] = arr[j++];  
 }  
 }  
 arr = newArr;  
 size++;  
 }  
 public void printArray() {  
 System.*out*.println(Arrays.*toString*(arr));  
 }  
}  
class InsertElement {  
 public static void main(String[] args) {  
 int[] initialArray = {1, 2, 4, 5};  
 ArrayHandler handler = new ArrayHandler(initialArray);  
 handler.insertAtPosition(3, 2);  
 handler.printArray();  
 }  
}

**Time Complexity:** O(n) (due to shifting elements)

### **1.2 Delete Operation**

#### **Code Example: Deleting an Element from an Array**

import java.util.Arrays;  
class ArrayDeletion {  
 private int[] arr;  
 private int size;  
 public ArrayDeletion(int[] inputArr) {  
 this.arr = inputArr;  
 this.size = inputArr.length;  
 }  
 public void deleteAtPosition(int pos) {  
 int[] newArr = new int[size - 1];  
 for (int i = 0, j = 0; i < size; i++) {  
 if (i != pos) {  
 newArr[j++] = arr[i];  
 }  
 }  
 arr = newArr;  
 size--;  
 }  
 public void printArray() {  
 System.*out*.println(Arrays.*toString*(arr));  
 }  
}  
class DeleteElement {  
 public static void main(String[] args) {  
 int[] initialArray = {1, 2, 3, 4, 5};  
 ArrayDeletion handler = new ArrayDeletion(initialArray);  
 handler.deleteAtPosition(2);  
 handler.printArray();  
 }  
}

**Time Complexity:** O(n) (due to shifting elements)

## **Session 2: Array Operations**

### **2.1 Traversing an Array**

class ArrayTraversal {  
 private int[] arr;  
 public ArrayTraversal(int[] inputArr) {  
 this.arr = inputArr;  
 }  
 public void traverseArray() {  
 for (int num : arr) {  
 System.*out*.print(num + " ");  
 }  
 System.*out*.println();  
 }  
}  
class TraverseArray {  
 public static void main(String[] args) {  
 int[] array = {10, 20, 30, 40, 50};  
 ArrayTraversal traversal = new ArrayTraversal(array);  
 traversal.traverseArray();  
 }  
}

**Time Complexity:** O(n)

### **2.2 Searching in an Array**

class ArraySearch {  
 private int[] arr;  
 public ArraySearch(int[] inputArr) {  
 this.arr = inputArr;  
 }  
 public int searchElement(int key) {  
 for (int i = 0; i < arr.length; i++) {  
 if (arr[i] == key) {  
 return i;  
 } }  
 return -1;  
 }  
}  
class SearchArray {  
 public static void main(String[] args) {  
 int[] array = {5, 15, 25, 35, 45};  
 ArraySearch search = new ArraySearch(array);  
 int index = search.searchElement(25);  
 System.*out*.println("Element found at index: " + index);  
 }  
}

**Time Complexity:** O(n) (Linear Search)

## **Session 3: Static & Dynamic Arrays**

### **3.1 Static Arrays**

* **Fixed Size**
* Memory allocated at compile time
* Example: int arr[10];

### **3.2 Dynamic Arrays**

* **Resizable**
* Memory allocated at runtime
* Implemented using ArrayList

#### **Code Example: Using Dynam**ic Arrays

class DynamicArray {  
 private int[] arr;  
 private int size, capacity;  
 public DynamicArray(int capacity) {  
 this.capacity = capacity;  
 arr = new int[capacity];  
 }  
 public void insert(int element) {  
 if (size == capacity) resize();  
 arr[size++] = element;  
 }  
 private void resize() {  
 int[] newArr = new int[capacity \*= 2];  
 System.*arraycopy*(arr, 0, newArr, 0, size);  
 arr = newArr;  
 }  
 public void printArray() {  
 for (int i = 0; i < size; i++) System.*out*.print(arr[i] + " ");  
 System.*out*.println();  
 }  
}  
public class Main {  
 public static void main(String[] args) {  
 DynamicArray da = new DynamicArray(2);  
 da.insert(10);  
 da.insert(20);  
 da.insert(30);  
 da.printArray(); // Output: 10 20 30  
 }  
}

## **4. ArrayList:**

### **4.1 Insertion in ArrayList**

import java.util.ArrayList;

class ArrayListInsertion {

public static void main(String[] args) {

ArrayList<Integer> list = new ArrayList<>();

// Inserting elements

list.add(10); // At the end

list.add(20);

list.add(30);

list.add(1, 15); // Inserting at index 1

System.out.println("ArrayList after insertion: " + list);

}

}

**Time Complexity:**

* add(element): **O(1)** (Amortized)
* add(index, element): **O(n)** (Shifting elements)

### **4.2 Deletion in ArrayList**

import java.util.ArrayList;  
class ArrayListDeletion {  
 public static void main(String[] args) {  
 ArrayList<Integer> list = new ArrayList<>();  
 list.add(10);  
 list.add(20);  
 list.add(30);  
 list.add(40);  
 list.remove(1); // Removes element at index 1 (20)  
 list.remove(Integer.*valueOf*(30)); // Removes element 30  
 System.*out*.println("ArrayList after deletion: " + list);  
 }  
}

**Time Complexity:**

* remove(index): **O(n)** (Shifting elements)
* remove(object): **O(n)** (Finding + Removing)

### **4.3 Searching in an ArrayList**

import java.util.ArrayList;

class ArrayListSearch {

public static void main(String[] args) {

ArrayList<Integer> list = new ArrayList<>();

list.add(5);

list.add(10);

list.add(15);

list.add(20);

int index = list.indexOf(15);

boolean exists = list.contains(10);

System.out.println("Index of 15: " + index);

System.out.println("Does 10 exist? " + exists);

}

}

**Time Complexity:**

* indexOf(element): **O(n)**
* contains(element): **O(n)**

## **Basic Practice Task**

1. Implement a program to **reverse an array in place**.
2. Implement a **merge function** for two sorted arrays.
3. **Merge two ArrayList** into a single list.
4. **Update an element** in an ArrayList at a specific index.

## **Scenario-Based Case Studies**

### **Case Study 1: Number List Operations**

**Task:** Implement an **Integer List Manager** with ArrayList<Integer>.  
Features:

* Insert numbers into the list.
* Find the largest and smallest number.
* Sort the list in ascending order.

**Hint:** Use Collections.sort(list).

## **Case Study 2: Student Records System**

**Scenario:**

A **school** maintains student roll numbers using an array-based system. The system should allow the modification and retrieval of records.

**Tasks:**

1. **Insert a new student's roll number** at a specified position.
2. **Search for a roll number** to check if a student is enrolled.
3. **Delete a student's roll number** if they transfer to another school.
4. **Count how many students are currently enrolled.**
5. **Find the student with the highest roll number** (assuming roll numbers are unique and increasing

### **Conclusion**

This lab covered fundamental array operations using an **object-oriented approach**, including insertion, deletion, traversal, searching, and static vs. dynamic arrays. These concepts will be a foundation for more advanced data structures like linked lists and trees.