Lab Material

Data Structures

Traversing Algorithm

```
public class ArrayTraversal {
    public static void main(String[] args) {
        // Example array
        int[] LA = {10, 20, 30, 40, 50};

        // Lower bound (first index)
        int LB = 0;
        // Upper bound (last index)
        int UB = LA.length - 1;

        // Step 1: Initialize counter
        int k = LB;

        // Step 2: Repeat steps 3 & 4 while k <= UB
        while (k <= UB) {
            // Step 3: Visit element
            System.out.println("Element at index " + k + " = " + LA[k]);

            // Step 4: Increase counter
            k = k + 1;
        }

        // Step 5: Exit
```

```
        System.out.println("Traversal completed.");
    }
}
```

Insertion:

```
public class ArrayInsertion {
    public static void main(String[] args) {
        int[] LA = new int[10];  // Array with extra space
        int N = 5;               // Current number of elements
        int ITEM = 99;           // New element to insert
        int K = 3;               // Position (index) to insert at

        // Initialize array with sample values
        LA[0] = 10;
        LA[1] = 20;
        LA[2] = 30;
        LA[3] = 40;
        LA[4] = 50;

        System.out.println("Before Insertion:");
        for (int i = 0; i < N; i++) {
            System.out.print(LA[i] + " ");
        }

        // Step 1: Initialize counter
        int J = N;
```

```
        // Step 2: Repeat while J >= K
        while (J >= K) {
            // Step 3: Move element downward
            LA[J + 1] = LA[J];

            // Step 4: Decrease counter
            J = J - 1;
        }

        // Step 5: Insert element at K
        LA[K] = ITEM;

        // Step 6: Reset N
        N = N + 1;

        System.out.println("\n\nAfter Insertion of " + ITEM + " at index " + K + ":");
        for (int i = 0; i < N; i++) {
            System.out.print(LA[i] + " ");
        }
    }
}
```

Deletion:

```
public class ArrayDeletion {
    public static void main(String[] args) {
        int[] LA = {10, 20, 30, 40, 50};
        int N = 5;       // Number of elements
```

```
        int K = 2;       // Index position to delete (0-based)
        int ITEM;        // Store deleted element

        System.out.println("Before Deletion:");
        for (int i = 0; i < N; i++) {
            System.out.print(LA[i] + " ");
        }

        // Step 1: Set ITEM = LA[K]
        ITEM = LA[K];

        // Step 2: Repeat for J = K to N-1
        for (int J = K; J < N - 1; J++) {
            LA[J] = LA[J + 1];
        }

        // Step 3: Reset N
        N = N - 1;

        System.out.println("\n\nDeleted Element: " + ITEM);

        System.out.println("After Deletion:");
        for (int i = 0; i < N; i++) {
            System.out.print(LA[i] + " ");
        }
    }
}
```

Bubble Sort

```
public class BubbleSort {
    public static void main(String[] args) {
        int[] DATA = {50, 30, 20, 10, 40};
        int N = DATA.length;
        int PTR, K, temp;

        System.out.println("Before Sorting:");
        for (int i = 0; i < N; i++) {
            System.out.print(DATA[i] + " ");
        }

        // Step 1: Repeat steps 2 and 3 for K = 1 to N-1
        for (K = 1; K < N; K++) {
            PTR = 0;

            // Step 2: Repeat while PTR ≤ N-K-1
            while (PTR <= N - K - 1) {
                // a) If DATA[PTR] > DATA[PTR+1], then interchange
                if (DATA[PTR] > DATA[PTR + 1]) {
                    temp = DATA[PTR];
                    DATA[PTR] = DATA[PTR + 1];
                    DATA[PTR + 1] = temp;
                }
                // b) Set PTR := PTR + 1
                PTR = PTR + 1;
```

```
            }
        }

        System.out.println("\n\nAfter Bubble Sort:");
        for (int i = 0; i < N; i++) {
            System.out.print(DATA[i] + " ");
        }
    }
}
```

Linear Search:

```
public class LinearSearch {
    public static void main(String[] args) {
        int[] DATA = {15, 25, 35, 45, 55};
        int N = DATA.length;
        int ITEM = 45;   // Element to search
        int LOC = -1;    // Store found location
```

Linear Search

```
        for (int i = 0; i < N; i++) {
            if (DATA[i] == ITEM) {
                LOC = i; // Found
                break;
            }
        }

        // Display result
```

```
        if (LOC != -1) {
            System.out.println("Value found at Location: " + LOC);
        } else {
            System.out.println("Value not found in the array!");
        }
    }
}
```

Binary Search

```
public class BinarySearch {
    public static void main(String[] args) {
        int[] DATA = {10, 20, 30, 40, 50, 60, 70};  // Sorted array
        int LB = 0;               // Lower bound (first index)
        int UB = DATA.length - 1; // Upper bound (last index)
        int ITEM = 40;            // Element to search
        int LOC = -1;             // Location (default = not found)

        int BEG = LB;
        int END = UB;
        int MID = (BEG + END) / 2;

        // Step 2: Repeat until BEG ≤ END and not found
        while (BEG <= END && DATA[MID] != ITEM) {
            if (ITEM < DATA[MID]) {
                END = MID - 1;  // Search left half
            } else {
                BEG = MID + 1;  // Search right half
```

```
            }
            MID = (BEG + END) / 2;
        }

        // Step 3: Check result
        if (BEG <= END && DATA[MID] == ITEM) {
            LOC = MID;  // Found
            System.out.println("Value found at Location: " + LOC);
        } else {
            LOC = -1;   // Not found
            System.out.println("Value not found in the array!");
        }
    }
}
```

Exercises:

1. Linear Search Task

- Write a program that performs **linear search** to find an element in an array of strings (e.g., names of students).
- Print the index where the element is found, otherwise display *"Not Found"*.

2. **Binary Search Task**

- Write a program to perform **binary search** on a sorted array of integers.

- Modify the code to also count and display the **number of comparisons** made until the element is found (or not found).

3. **Insertion Task**
   - Write a program to insert an element at a given position in an integer array.
   - Example: Insert 25 at position 3 in {10, 20, 30, 40} → Result: {10, 20, 25, 30, 40}.
4. **Deletion Task**
   - Write a program to delete an element from a given position in an array.
   - Example: Delete element at position 2 in {5, 10, 15, 20} → Result: {5, 15, 20}.
5. **Bubble Sort Task**
   - Implement **bubble sort** on an array of integers.
   - Extend the program to display the array after **each pass/iteration**, so students can see how sorting progresses step by step.