PRACTICLE FILE

ON

Analysis of Algorithms and Data Structures

Amity University, Uttar Pradesh

In partial fulfilment of the requirements for the award of the degree of

Bachelor of Computer Applications

SUBMITTED BY

Divya Khurana
A1004823220
BCA
(2023-2026)

SUBMITTED TO

Dr Garima Bohra

Q1. Use the selection sort to put the number 3, 2, 4, 1, 5 into increasing order. Illustrate the output returned in each pass clearly. Also, write the pseudo
algorithm to it.

Statement: We have to arrange numbers 3, 2, 4, 1, 5 in increasing order using Selection Sort.

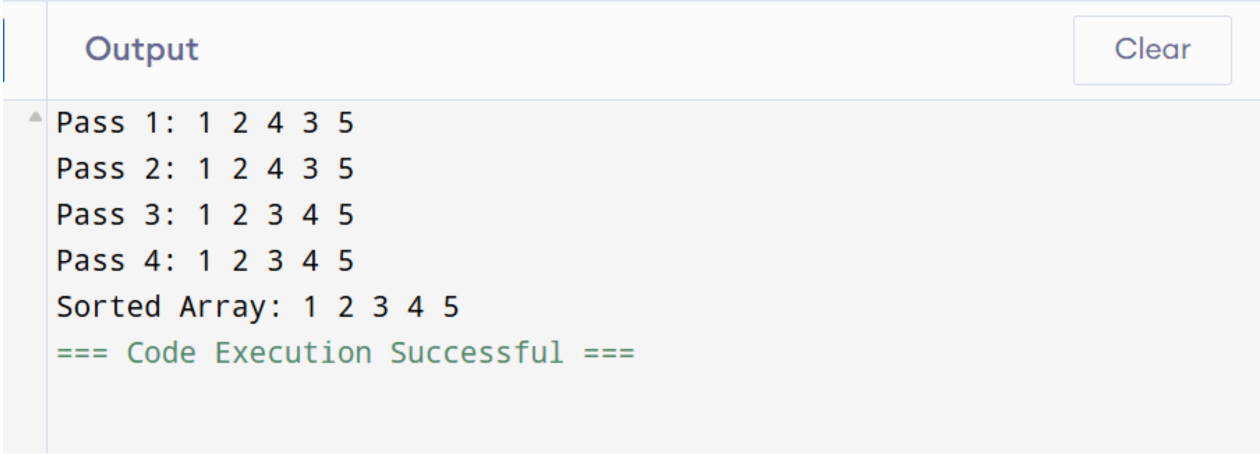
Algorithm:

```
SelectionSort(A, n):
 for i = 0 to n-2 do
    minIndex = i
    for j = i+1 to n-1 do
       if A[j] < A[minIndex] then
          minIndex = j
    swap A[i] with A[minIndex]
```

Code:

```
public class SelectionSortDemo {
   public static void main(String[] args) {
      int[] arr = {3, 2, 4, 1, 5};
      int n = arr.length;
      for (int i = 0; i < n - 1; i++) {
         int minIndex = i;
         for (int j = i + 1; j < n; j++) {
            if (arr[j] < arr[minIndex]) {
               minIndex = j;
            }
         }
         int temp = arr[i];
         arr[i] = arr[minIndex];
         arr[minIndex] = temp;
         System.out.print("Pass " + (i + 1) + ": ");
         for (int num : arr) {
            System.out.print(num + " ");
         }
         System.out.println();
      }
      System.out.print("Sorted Array: ");
      for (int num : arr) {
         System.out.print(num + " ");
      }
   }
```

Output:



```
Pass 1: 1 2 4 3 5
Pass 2: 1 2 4 3 5
Pass 3: 1 2 3 4 5
Pass 4: 1 2 3 4 5
Sorted Array: 1 2 3 4 5
=== Code Execution Successful ===
```

Q2. Write a program to sort the given array using MergeSort.

Statement:

We need to sort the array [3, 2, 4, 1, 5] using Merge Sort, which is a divide-and-conquer algorithm. It divides the array into halves, recursively sorts them, and then merges them back together.

Algorithm:

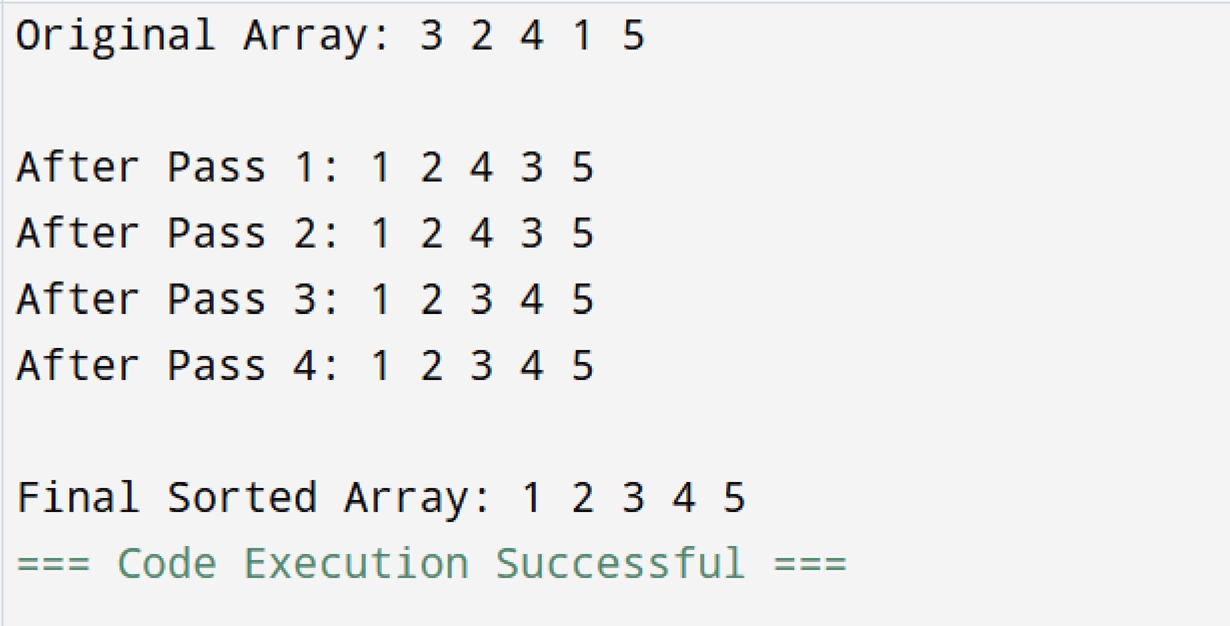
```
MergeSort(A, left, right):
  if left < right then
    mid = (left + right) / 2
    MergeSort(A, left, mid)
    MergeSort(A, mid+1, right)
    Merge(A, left, mid, right)

Merge (A, left, mid, right):
  Create two temporary arrays L and R
  Copy data into L and R
  Merge them back into A[left..right] in sorted order
```

Code:

```
public class SelectionSortSteps {
  public static void main(String[] args) {
    int[] arr = {3, 2, 4, 1, 5};
    int n = arr.length;

    System.out.print("Original Array: ");
    for (int num : arr) System.out.print(num + " ");
    System.out.println("\n");

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

```
                if (arr[j] < arr[minIndex])
                    minIndex = j;
            }
            int temp = arr[i];
            arr[i] = arr[minIndex];
            arr[minIndex] = temp;

            System.out.print("After Pass " + (i + 1) + ": ");
            for (int num : arr) System.out.print(num + " ");
            System.out.println();
        }

        System.out.print("\nFinal Sorted Array: ");
        for (int num : arr) System.out.print(num + " ");
    }
}
```

Output:



```
Original Array: 3 2 4 1 5

After Pass 1: 1 2 4 3 5
After Pass 2: 1 2 4 3 5
After Pass 3: 1 2 3 4 5
After Pass 4: 1 2 3 4 5

Final Sorted Array: 1 2 3 4 5
=== Code Execution Successful ===
```

Q3.Trace quick sort on the list L= {11, 34, 67, 78, 78, 78, 99}.What are your observations?

Statement:

We need to apply the Quick Sort algorithm on the list:
L = {11, 34, 67, 78, 78, 78, 99} and trace how the sorting takes place. Finally, we will observe the behavior of Quick Sort on a list that already contains **duplicates** and is **almost sorted**.

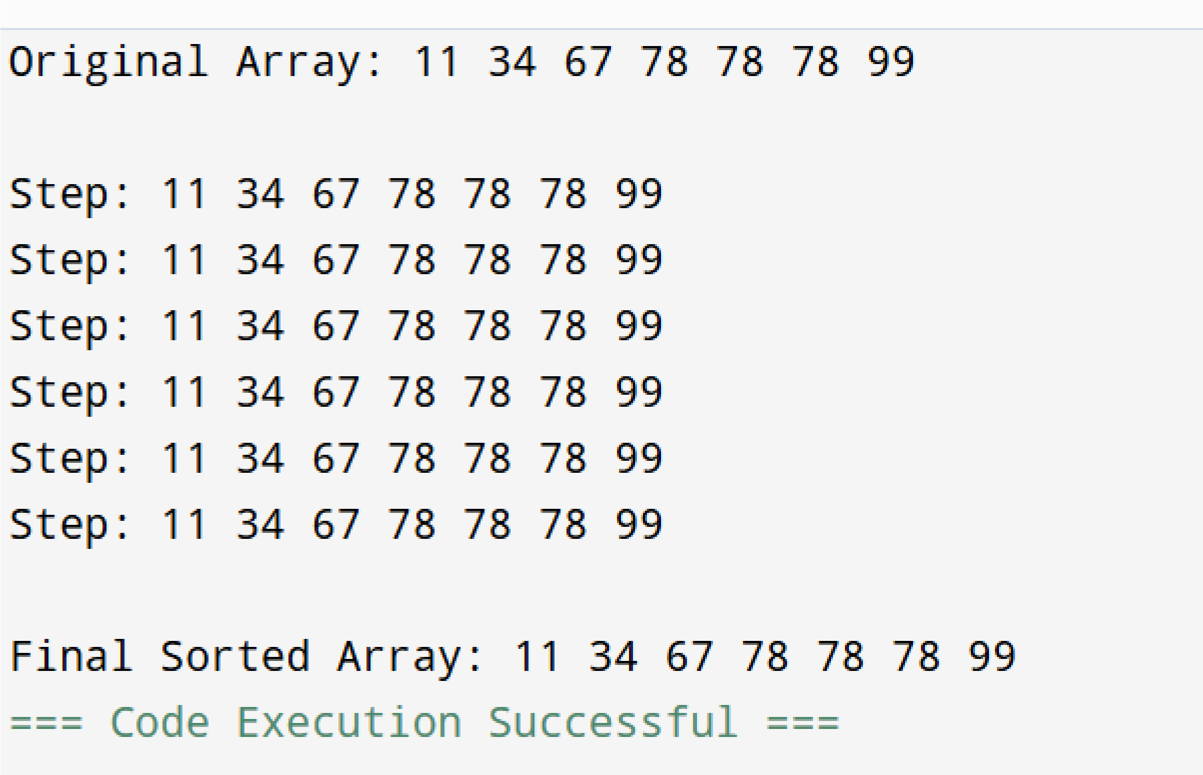
Algorithm:

```
QuickSort(arr, low, high):
    if low < high:
        pivotIndex = Partition(arr, low, high)
        QuickSort(arr, low, pivotIndex - 1)
        QuickSort(arr, pivotIndex + 1, high)

Partition(arr, low, high):
    pivot = arr[high]
    i = low - 1
    for j = low to high - 1:
        if arr[j] <= pivot:
            i = i + 1
            swap arr[i] and arr[j]
    swap arr[i+1] and arr[high]
    return i + 1
```

Code:

```
public class QuickSortTrace {

    public static int partition(int arr[], int low, int high) {
        int pivot = arr[high];
        int i = low - 1;

        for (int j = low; j < high; j++) {
            if (arr[j] <= pivot) {
                i++;
                int temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }
        }

        int temp = arr[i + 1];
        arr[i + 1] = arr[high];
        arr[high] = temp;

        return (i + 1);
    }

    public static void quickSort(int arr[], int low, int high) {
        if (low < high) {
            int pi = partition(arr, low, high);

            // Print array at each step (Tracing)
            System.out.print("Step: ");
            for (int x : arr) {
                System.out.print(x + " ");
            }
            System.out.println();

            quickSort(arr, low, pi - 1);
            quickSort(arr, pi + 1, high);
        }
    }

    public static void main(String args[]) {
        int arr[] = {11, 34, 67, 78, 78, 78, 99};
        int n = arr.length;

        System.out.print("Original Array: ");
        for (int x : arr) {
            System.out.print(x + " ");
        }
        System.out.println("\n");

        quickSort(arr, 0, n - 1);

        System.out.print("\nFinal Sorted Array: ");
        for (int x : arr) {
            System.out.print(x + " ");
        }
    }
}
```

Output:



```
Original Array: 11 34 67 78 78 78 99

Step: 11 34 67 78 78 78 99
Step: 11 34 67 78 78 78 99
Step: 11 34 67 78 78 78 99
Step: 11 34 67 78 78 78 99
Step: 11 34 67 78 78 78 99
Step: 11 34 67 78 78 78 99

Final Sorted Array: 11 34 67 78 78 78 99
=== Code Execution Successful ===
```

Q4. Write the program for Searching Techniques and Complexity Analysis.

Statement:

This program demonstrates two searching techniques:

1. **Linear Search** – checks each element one by one (Time: O(n), Space: O(1)).
2. **Binary Search** – works on sorted arrays by dividing the search space in half (Time: O(log n), Space: O(1)).

Algorithm:

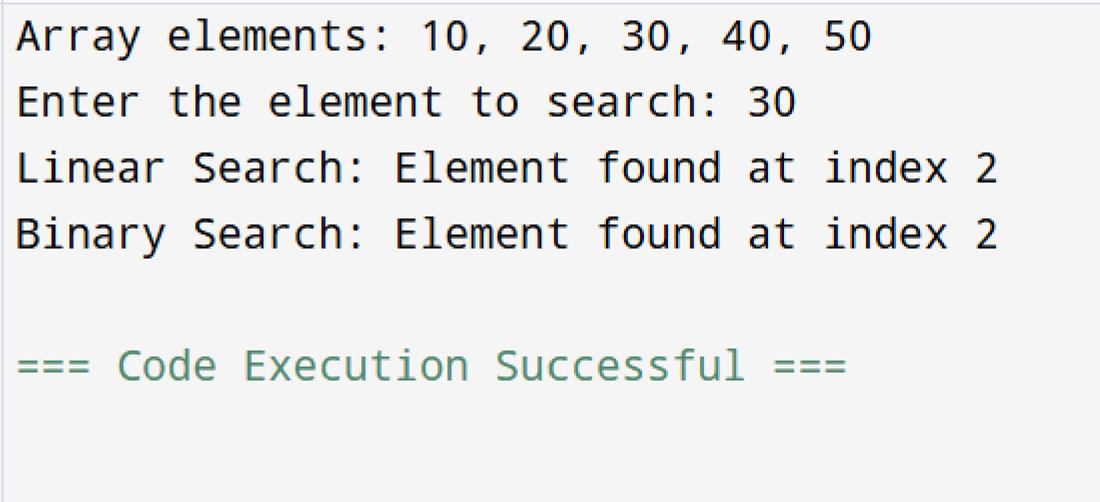
```
LinearSearch(arr, n, key):
  for i ← 0 to n-1 do
    if arr[i] = key then
      return i
  return -1

BinarySearch(arr, n, key):
  low ← 0, high ← n-1
  while low ≤ high do
    mid ← (low + high) / 2
    if arr[mid] = key then
      return mid
    else if key < arr[mid] then
      high ← mid - 1
    else
      low ← mid + 1
  return -1
```

Code:

```
import java.util.Scanner;

public class SearchingTechniques {

  // Linear Search Method
  public static int linearSearch(int[] arr, int key) {
    for (int i = 0; i < arr.length; i++) {
      if (arr[i] == key) {
        return i;
      }
    }
    return -1;
  }

  // Binary Search Method
  public static int binarySearch(int[] arr, int key) {
    int low = 0, high = arr.length - 1;

    while (low <= high) {
      int mid = (low + high) / 2;

      if (arr[mid] == key) {
        return mid;
      } else if (key < arr[mid]) {
        high = mid - 1;
      } else {
        low = mid + 1;
      }
    }
    return -1;
  }

  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);

    int[] arr = {10, 20, 30, 40, 50}; // sorted array
    System.out.println("Array elements: 10, 20, 30, 40, 50");
```

```
        System.out.print("Enter the element to search: ");
        int key = sc.nextInt();

        // Linear Search
        int linearResult = linearSearch(arr, key);
        if (linearResult == -1)
            System.out.println("Linear Search: Element not found.");
        else
            System.out.println("Linear Search: Element found at index " + linearResult);

        // Binary Search
        int binaryResult = binarySearch(arr, key);
        if (binaryResult == -1)
            System.out.println("Binary Search: Element not found.");
        else
            System.out.println("Binary Search: Element found at index " + binaryResult);

        sc.close();
    }
}
```

Output:



```
Array elements: 10, 20, 30, 40, 50
Enter the element to search: 30
Linear Search: Element found at index 2
Binary Search: Element found at index 2


=== Code Execution Successful ===
```

Q5. Write a program in Java to search item '8' in the given array 12, 7, 9, 5, 16, 8, 52, 67, 90 using Linear Search.

Statement:

We are given an array: {12, 7, 9, 5, 16, 8, 52, 67, 90}.

We need to search for the item **8** using the **Linear Search** algorithm.

Algorithm:

Step 1: Start

Step 2: Initialize array = [12, 7, 9, 5, 16, 8, 52, 67, 90]

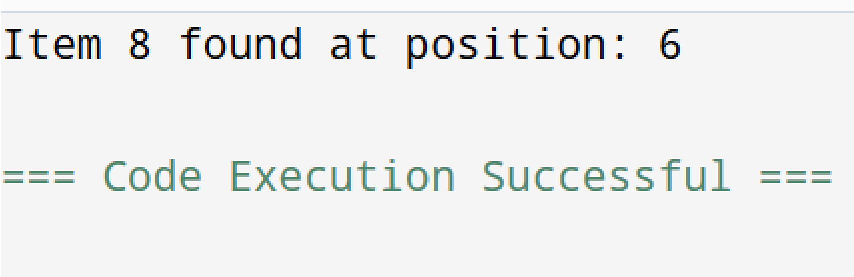
Step 3: Set item = 8

```
Step 4: For i = 0 to array.length - 1
        If array[i] == item
            Print "Item found at position i"
            Stop
Step 5: If loop ends, Print "Item not found"
Step 6: End
```

Code:

```java
public class LinearSearchExample {
    public static void main(String[] args) {
        int[] arr = {12, 7, 9, 5, 16, 8, 52, 67, 90};
        int item = 8;
        boolean found = false;

        for (int i = 0; i < arr.length; i++) {
            if (arr[i] == item) {
                System.out.println("Item " + item + " found at position: " + (i + 1));
                found = true;
                break;
            }
        }

        if (!found) {
            System.out.println("Item " + item + " not found in the array.");
        }
    }
}
```

Output



```
Item 8 found at position: 6

=== Code Execution Successful ===
```

Q6. Write a program in Java to search item ‘50’ in the given array 10, 20, 30, 40, 50, 60, 70, 80, 90, 100 using Binary search.

Statement:

We need to search for the item **50** in the array {10, 20, 30, 40, 50, 60, 70, 80, 90, 100} using **Binary Search** in Java.

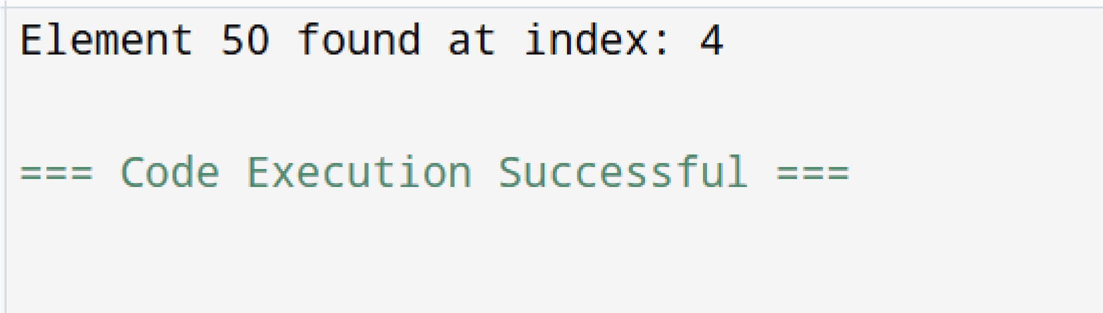
Algorithm:

```
Step 1: Start
Step 2: Initialize array = {10,20,30,40,50,60,70,80,90,100}
Step 3: Set low = 0, high = array.length - 1
Step 4: While low <= high
        mid = (low + high) / 2
        if array[mid] == key
            return mid (element found)
        else if array[mid] < key
            low = mid + 1
        else
            high = mid - 1
Step 5: If element not found, return -1
Step 6: End
```

Code:

```
public class BinarySearchExample {
    public static void main(String[] args) {
        int[] arr = {10, 20, 30, 40, 50, 60, 70, 80, 90, 100};
        int key = 50;
        int result = binarySearch(arr, key);

        if (result == -1) {
            System.out.println("Element " + key + " not found in array.");
        } else {
            System.out.println("Element " + key + " found at index: " + result);
        }
    }
    // Binary Search method
    public static int binarySearch(int[] arr, int key) {
        int low = 0, high = arr.length - 1;

        while (low <= high) {
            int mid = (low + high) / 2;

            if (arr[mid] == key) {
                return mid; // found
            } else if (arr[mid] < key) {
                low = mid + 1; // search right side
            } else {
                high = mid - 1; // search left side
            }
        }
        return -1; // not found
    }
}
```

Output:



```
Element 50 found at index: 4


=== Code Execution Successful ===
```