1. COMMON ELEMENTS IN THREE ARRAY

ALGORITHM:

1. Inside the outer loop, initialize another loop j to iterate through the elements of array b.
2. Compare the element at index i in array a with the element at index j in array b.
3. If a common element is found (a[i] == b[j]), enter a nested loop using index k to iterate through the elements of array c.
4. Compare the common element with the elements in array c.
5. If a match is found with an element in array c (a[i] == c[k]), print the common element and add a comma , after it.
6. Break out of the inner loop (for array c) to avoid printing the same element multiple times.
7. Break out of the middle loop (for array b) to avoid searching further in array b.
8. Continue checking the next element in array a.

```
package coding;

class Sort {
   int a[] = {2, 23, 3, 5, 6, 7};
   int b[] = {3, 2, 5, 9, 10, 7, 23};
   int c[] = {23, 5, 21, 2, 3, 17, 6, 3, 2, 1};

   public void threes() {
      for (int i = 0; i < a.length; ++i) {
         for (int j = 0; j < b.length; ++j) {
            if (a[i] == b[j]) {
               for (int k = 0; k < c.length; ++k) {
                  if (a[i] == c[k]) {
                     System.out.print(a[i] + ",");
                     break; // Stop searching for this common element in
array 'c'
```

```
                    }
                }
                break; // Stop searching for this common element in
array 'b'
            }
        }
    }
    System.out.println(); // Print a newline after the common
elements
  }
}

public class Three {
  public static void main(String[] args) {
    Sort s = new Sort();
    System.out.print("COMMON ELEMENTS IN THREE
ARRAYS: ");
    s.threes();
  }
}
```

OUTPUT:

COMMON ELEMENTS IN THREE ARRAYS:
2 23 3 5

2. DELTE SPECIFIED INDEX POSITION

Algorithm:

1. Define the array arr and specify the delete variable, which represents the position to delete.
2. Check if delete is a valid position (between 0 and arr.length - 1). If it's valid, proceed; otherwise, display an error message.
3. Create a new array newArr with a length of arr.length - 1.

4. Use System.arraycopy to copy elements from the original array to the new array, excluding the element at the specified position.
5. Print both the original and modified arrays to show the effect of deleting the element at the specified position.

```
package coding;

public class Delete {

	public static void main(String[] args) {

	 int[] arr = {1, 2, 3, 4, 5};

			int delete = 2;

			if (delete < 0 || delete >= arr.length) {
			    System.out.println("Invalid position. Element cannot be deleted.");
			    return;
			}

			int newSize = arr.length - 1;


			int[] newArr = new int[newSize];

			int newIndex = 0;
			for (int i = 0; i < arr.length; i++) {
			    if (i != delete) {
			        newArr[newIndex] = arr[i];
			        newIndex++;
			    }
			}
			System.out.println("Original Array: " + java.util.Arrays.toString(arr));
```

```
                System.out.println("Array after deleting element at position " + delete + ": " + java.util.Arrays.toString(newArr));
            }
        }
```

Output:
Original Array: [1, 2, 3, 4, 5]
Array after deleting element at position 2:
[1, 2, 4, 5]

3. MERGE ARRAY

Algorithm:

1. Initialize three pointers, i, j, and k, to 0. i and j are used to traverse arrays a and b, respectively, and k is used to index array m.
2. While both i and j are within their respective array bounds (i < a.length and j < b.length):
3. Compare a[i] and b[j].
4. If a[i] is smaller, copy it to m[k], increment i, and increment k.
5. Otherwise, copy b[j] to m[k], increment j, and increment k.
6. After the loop, if there are remaining elements in array a, copy them to m:
7. While i is within the bounds of array a (i < a.length), copy a[i] to m[k], increment i, and increment k.
8. Similarly, if there are remaining elements in array b, copy them to m:
9. While j is within the bounds of array b (j < b.length), copy b[j] to m[k], increment j, and increment k.
10. The array m now contains the merged and sorted elements from arrays a and b

```java
package coding;

class MergeSort {
    int a[] = {1, 3, 5, 7, 9};
    int b[] = {2, 4, 8, 10, 11, 12, 16};

    int a1 = a.length;
    int b1 = b.length;

    int n = a1 + b1;
    int m[] = new int[n];
    int i, j, k;

    public void merge() {
        System.out.println("MERGE TWO SORTED ARRAY INTO
THIRD SORTED ARRAY:");

        i = 0;
        j = 0;
        k = 0;

        while (i < a1 && j < b1) {
            if (a[i] < b[j]) {
                m[k] = a[i];
                i++;
            } else {
                m[k] = b[j];
                j++;
            }
            k++;
        }

        // Copy remaining elements from array 'a' (if any)
        while (i < a1) {
            m[k] = a[i];
```

```
            i++;
            k++;
        }

        // Copy remaining elements from array 'b' (if any)
        while (j < b1) {
            m[k] = b[j];
            j++;
            k++;
        }

        // Print the merged array
        for (k = 0; k < n; ++k) {
            System.out.print( m[k]+ " ");
        }
    }
}

public class Merge {
    public static void main(String[] args) {
        MergeSort m = new MergeSort();
        m.merge();
    }
}
```

OUTPUT:
MERGE TWO SORTED ARRAY INTO THIRD SORTED ARRAY:
1 2 3 4 5 7 8 9 10 11 12 16