Java Assignments (array)

1. Find second highest values in array without using predefined functions.

[2,1,3,4,5,6,7,89,3,4]   Result : 7

```
public class SecondHighestFinder {
    public static void main(String[] args) {
        int[] array = {2, 1, 3, 4, 5, 6, 7, 89, 3, 4};
        System.out.println("Second highest value is: " + findSecondHighest(array));
    }
    public static int findSecondHighest(int[] array) {
        int highest = Integer.MIN_VALUE;
        int secondHighest = Integer.MIN_VALUE;
        for (int num : array) {
            if (num > highest) {
                secondHighest = highest;
                highest = num;
            } else if (num > secondHighest && num < highest) {
                secondHighest = num;
            }
        }
        if (secondHighest == Integer.MIN_VALUE) {
            throw new IllegalArgumentException("Array must contain at least two distinct values");
        }

        return secondHighest;
    }
}
```

2. Find product of given 2 A,B matrices 2 X 2, Product matrix C. Print C matrix elements

```
2 3     2 4     10 11
1 2     2 1     6   6
```

```
public class MatrixMultiplication {
    public static void main(String[] args) {
        int[][] A = {
            {2, 3},
            {1, 2}
        };
        int[][] B = {
            {2, 4},
            {2, 1}
        };
        int[][] C = multiplyMatrices(A, B);
        System.out.println("Matrix C:");
        for (int i = 0; i < C.length; i++) {
            for (int j = 0; j < C[i].length; j++) {
                System.out.print(C[i][j] + " ");
            }
            System.out.println();
        }
    }
    public static int[][] multiplyMatrices(int[][] A, int[][] B) {
        int size = 2; // Since both matrices are 2x2
        int[][] C = new int[size][size];
        for (int i = 0; i < size; i++) {
            for (int j = 0; j < size; j++) {
                C[i][j] = 0;
                for (int k = 0; k < size; k++) {
```

```
                C[i][j] += A[i][k] * B[k][j];
            }
        }
    }
    return C;
  }
}
```

3. Find repeated element with high frequency an array , print it.

[3,4,5,2,3,4,6,7,8,5,3,6,7,8,3]

Result 3

```
public class MostFrequentElement {
  public static void main(String[] args) {
    int[] array = {3, 4, 5, 2, 3, 4, 6, 7, 8, 5, 3, 6, 7, 8, 3};
    int result = findMostFrequentElement(array);
    System.out.println("Element with highest frequency: " + result);
  }
  public static int findMostFrequentElement(int[] array) {
    int n = array.length;
    int mostFrequent = array[0];
    int highestFrequency = 0;
    for (int i = 0; i < n; i++) {
      int currentElement = array[i];
      int frequency = 0;
      for (int j = 0; j < n; j++) {
        if (array[j] == currentElement) {
          frequency++;
        }
      }
      if (frequency > highestFrequency) {
```

```
                highestFrequency = frequency;
                mostFrequent = currentElement;
            }
        }
        return mostFrequent;
    }
}
```

Java assignments (String)

1. Find first last repeating character in a given string.

Input : language

Output :g

```
public class RepeatingCharacter {
    public static void main(String[] args) {
        String input = "language";
        char firstRepeating = findFirstRepeatingCharacter(input);
        char lastRepeating = findLastRepeatingCharacter(input);

        System.out.println("First repeating character: " + firstRepeating);
        System.out.println("Last repeating character: " + lastRepeating);
    }
    public static char findFirstRepeatingCharacter(String input) {
        int length = input.length();
        for (int i = 0; i < length; i++) {
            char ch = input.charAt(i);
            for (int j = i + 1; j < length; j++) {
                if (input.charAt(j) == ch) {
                    return ch;
                }
            }
        }
        return '\0';
    }
    public static char findLastRepeatingCharacter(String input) {
        int length = input.length();
        int[] lastIndex = new int[256];
        for (int i = 0; i < 256; i++) {
            lastIndex[i] = -1;
        }
        for (int i = 0; i < length; i++) {
            lastIndex[input.charAt(i)] = i;
        }
        for (int i = length - 1; i >= 0; i--) {
            char ch = input.charAt(i);
            if (lastIndex[ch] != i) {
                return ch;
            }
        }
        return '\0';
    }
}
```

2. Reverse words in a given string without using predefined functions.

Input: *i love programming very much*

Output   : much very programming love i

```
public class ReverseWords {
    public static void main(String[] args) {
        String input = "i love programming very much";
        String reversed = reverseWords(input);
        System.out.println("Reversed words: " + reversed);
    }

    public static String reverseWords(String input) {
        String[] words = extractWords(input);
        reverse(words);
        return buildString(words);
    }


    private static String[] extractWords(String input) {
        String[] words = new String[input.length() / 2 + 1];
        int count = 0;
        int start = 0;
        for (int i = 0; i <= input.length(); i++) {
            if (i == input.length() || input.charAt(i) == ' ') {
                words[count++] = input.substring(start, i);
                start = i + 1;
            }
        }
        return java.util.Arrays.copyOf(words, count);
    }
    private static void reverse(String[] words) {
        int left = 0;
        int right = words.length - 1;
        while (left < right) {
            String temp = words[left];
            words[left] = words[right];
            words[right] = temp;
            left++;
            right--;
        }
    }
    private static String buildString(String[] words) {
        StringBuilder sb = new StringBuilder();
        for (int i = 0; i < words.length; i++) {
            if (i > 0) {
                sb.append(' ');
            }
            sb.append(words[i]);
        }
        return sb.toString();
    }
}
```

3. Find count of distinct subsequences in a string. Input “var” v a r

The seven distinct subsequences are “”, “v”, “a”, “r”, “va”, “av”,”ar”,”ra”,”vr”,”rv”,”var”,”rva”,”rav”,”avr”,”arv”,….

```
import java.util.HashSet;
import java.util.Set;
public class DistinctSubsequences {
    public static void main(String[] args) {
        String input = "var";
        int count = countDistinct(input);
        System.out.println("Number of distinct subsequences: " + count);
    }
    public static int countDistinct(String input) {
        Set<String> subsequences = new HashSet<>();
        generateSubsequences(input, "", 0, subsequences);
        return subsequences.size();
    }
    private static void generateSubsequences(String input, String current, int index, Set<String> 
subsequences) {
        if (index == input.length()) {
            subsequences.add(current);
            return;}
        generateSubsequences(input, current + input.charAt(index), index + 1, subsequences);
        generateSubsequences(input, current, index + 1, subsequences);
    }}
```