1, Given the list of array return array in which each element is the product of other element except ith element (try to do it without division operation)

input: [1,2,3,4]

output:[24,12,8,6]

```
public class ProductExceptSelf {
    public static int[] productExceptSelf(int[] nums) {
        int n = nums.length;
        int[] result = new int[n];

        // Calculate product of elements to the left of each element
        int leftProduct = 1;
        for (int i = 0; i < n; i++) {
            result[i] = leftProduct;
            leftProduct *= nums[i];
        }

        // Calculate product of elements to the right of each element
        int rightProduct = 1;
        for (int i = n - 1; i >= 0; i--) {
            result[i] *= rightProduct;
            rightProduct *= nums[i];
        }

        return result;
    }

    public static void main(String[] args) {
        int[] input = {1, 2, 3, 4};
        int[] output = productExceptSelf(input);

        // Print the result
        System.out.print("[");
        for (int i = 0; i < output.length; i++) {
            System.out.print(output[i]);
            if (i < output.length - 1) {
                System.out.print(", ");
            }
        }
        System.out.println("]");
    }
}
```

2. Medium: Given an array list return all possible permutations Input: nums = [1,4,3]

Output: [[1,4,3],[1,3,4],[4,1,3],[4,3,1],[3,1,4],[3,4,1]]

```
import java.util.ArrayList;
import java.util.List;


public class Permutations {
    public static List<List<Integer>> permute(int[] nums) {
        List<List<Integer>> result = new ArrayList<>();
        List<Integer> currentPermutation = new ArrayList<>();
        boolean[] used = new boolean[nums.length];

        generatePermutations(nums, used, currentPermutation, result);

        return result;
    }
    private static void generatePermutations(int[] nums, boolean[] used, List<Integer> currentPermutation, List<List<Integer>> result) {
        if (currentPermutation.size() == nums.length) {
            result.add(new ArrayList<>(currentPermutation));
            return;
        }

        for (int i = 0; i < nums.length; i++) {
            if (!used[i]) {
                used[i] = true;
                currentPermutation.add(nums[i]);
                generatePermutations(nums, used, currentPermutation, result);
                used[i] = false;
                currentPermutation.remove(currentPermutation.size() - 1);
            }
        }
    }

    public static void main(String[] args) {
        int[] nums = {1, 4, 3};
        List<List<Integer>> permutations = permute(nums);

        // Print the result
        System.out.println(permutations);
    }
}
```

3. import java.util.ArrayList;

```
import java.util.HashSet;
import java.util.List;
import java.util.Set;

class TrieNode {
    TrieNode[] children;
    boolean isEnd;

    public TrieNode() {
        this.children = new TrieNode[26];
        this.isEnd = false;
    }
}

public class ClubbedWords {
    public static List<String> findAllClubbedWords(String[] words) {
        List<String> result = new ArrayList<>();
        Set<String> wordSet = new HashSet<>();
        TrieNode root = new TrieNode();
        // Build trie with all words

        for (String word : words) {

            insertWord(root, word);

            wordSet.add(word);

        }



        // Check for clubbed words

        for (String word : words) {

            wordSet.remove(word); // Remove the current word to prevent it from being
considered as a clubbed word

            if (canFormClubbedWord(word, root, wordSet)) {

                result.add(word);

            }

            wordSet.add(word); // Add the current word back for the next iteration

        }



        return result;

    }



    private static void insertWord(TrieNode root, String word) {

        TrieNode node = root;

        for (char c : word.toCharArray()) {

            int index = c - 'a';
            if (node.children[index] == null) {

                node.children[index] = new TrieNode();

            }

            node = node.children[index];

        }

        node.isEnd = true;

    }


    private static boolean canFormClubbedWord(String word, TrieNode root,
Set<String> wordSet) {

        int n = word.length();

        if (n == 0) {

            return false;

        }


        boolean[] dp = new boolean[n + 1];

        dp[0] = true;


        for (int i = 1; i <= n; i++) {

            TrieNode node = root;

            for (int j = i - 1; j >= 0; j--) {

                int index = word.charAt(j) - 'a';

                if (node.children[index] == null) {

                    break;
                }

                node = node.children[index];
                if (node.isEnd && dp[j]) {
                    dp[i] = true;
                    break;
                }
            }
        }

        return dp[n];
    }

    public static void main(String[] args) {
        String[] words = {"mat", "mate", "matbellmates", "bell", "bellmatesbell",
"butterribbon", "butter", "ribbon"};
        List<String> clubbedWords = findAllClubbedWords(words);

        // Print the result
        System.out.println(clubbedWords);
    }
}
```