# 28th August Backtracking

**Q1: Given an integer array arr and an integer k, return true if it is possible to divide the vector into k non-empty subsets with equal sum.**

java

Copy code

```
public static boolean canDivideArray(int[] arr, int k) {
    int sum = 0;
    for (int num : arr) {
        sum += num;
    }

    if (sum % k != 0) {
        return false;
    }

    int targetSum = sum / k;
    boolean[] used = new boolean[arr.length];

    return canPartition(arr, 0, k, targetSum, used);
}

private static boolean canPartition(int[] arr, int startIndex, int k, int targetSum,
boolean[] used) {
    if (k == 0) {
        return true; // All subsets have been formed.
    }

    if (targetSum == 0) {
        return canPartition(arr, 0, k - 1, targetSum, used); // Start forming the next subset.
    }

    for (int i = startIndex; i < arr.length; i++) {
        if (!used[i] && arr[i] <= targetSum) {
            used[i] = true;
            if (canPartition(arr, i + 1, k, targetSum - arr[i], used)) {
                return true;
            }
            used[i] = false;
        }
    }

    return false;
}
```

**Q2: Given an integer array arr, print all the possible permutations of the given array (containing non-repeating elements).**

java

Copy code

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

public static List<List<Integer>> permute(int[] arr) {
    List<List<Integer>> result = new ArrayList<>();
    List<Integer> current = new ArrayList<>();
    boolean[] used = new boolean[arr.length];

    generatePermutations(arr, current, result, used);

    return result;
}

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

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

**Q3: Given a collection of numbers nums, possibly containing duplicates, return all possible unique permutations.**

```
java
Copy code
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

public static List<List<Integer>> permuteUnique(int[] nums) {
    List<List<Integer>> result = new ArrayList<>();
    Arrays.sort(nums); // Sort the array to handle duplicates.

    generatePermutations(nums, new ArrayList<>(), result, new boolean[nums.length]);

    return result;
}

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

    for (int i = 0; i < nums.length; i++) {
        if (used[i] || (i > 0 && nums[i] == nums[i - 1] && !used[i - 1])) {
            continue; // Skip duplicates.
        }

        current.add(nums[i]);
        used[i] = true;
        generatePermutations(nums, current, result, used);
        current.remove(current.size() - 1);
        used[i] = false;
    }
}
```

**Q4: Check if the product of some subset of an array is equal to the target value.**

java

Copy code

```
public static boolean isSubsetProductEqualToTarget(int[] arr, int target) {
    return isSubsetProductEqualToTarget(arr, target, 0, 1);
}

private static boolean isSubsetProductEqualToTarget(int[] arr, int target, int index, int
currentProduct) {
    if (currentProduct == target) {
        return true;
    }

    if (index >= arr.length || currentProduct > target) {
        return false;
    }

    // Include the current element in the product.
    if (isSubsetProductEqualToTarget(arr, target, index + 1, currentProduct * arr[index])) {
        return true;
    }

    // Exclude the current element from the product.
    return isSubsetProductEqualToTarget(arr, target, index + 1, currentProduct);
}
```

*Q5: The n-queens puzzle - Return the number of distinct solutions.*

java

Copy code

```java
public static int solveNQueens(int n) {
    int[] columnPlacement = new int[n];
    return countNQueensSolutions(n, 0, columnPlacement);
}

private static int countNQueensSolutions(int n, int row, int[] columnPlacement) {
    if (row == n) {
        return 1; // Found a valid solution.
    }

    int count = 0;

    for (int col = 0; col < n; col++) {
        if (isValidPlacement(columnPlacement, row, col)) {
            columnPlacement[row] = col;
            count += countNQueensSolutions(n, row + 1, columnPlacement);
        }
    }

    return count;
}

private static boolean isValidPlacement(int[] columnPlacement, int row, int col) {
    for (int i = 0; i < row; i++) {
        if (columnPlacement[i] == col || Math.abs(columnPlacement[i] - col) == Math.abs(i - row)) {
            return false;
        }
    }
    return true;
}
```