1. **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.**

Ans.

```
package Skills;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.Scanner;

public class PartitionArray
{


    public static boolean canPartitionKSubsets(int[] nums, int k)
    {
        int totalSum = Arrays.stream(nums).sum();

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

        int targetSum = totalSum / k;

        boolean[] visited = new boolean[nums.length];

        List<List<Integer>> subsets = new ArrayList<>();

        for (int i = 0; i < k; i++)
        {
            subsets.add(new ArrayList<>());
        }

        return canPartition(nums, visited, 0, k, 0, targetSum, subsets);
    }


    private static boolean canPartition(int[] nums, boolean[] visited, int startIndex,
int k, int currentSum, int targetSum, List<List<Integer>> subsets)
    {
        if (k == 0) {
            System.out.println("The subsets are:");
            for (List<Integer> subset : subsets)
            {
                System.out.println(subset);
            }
            return true;
        }

        if (currentSum == targetSum)
        {
            return canPartition(nums, visited, 0, k - 1, 0, targetSum, subsets);
        }

        for (int i = startIndex; i < nums.length; i++)
        {
            if (!visited[i])
            {
                visited[i] = true;
                subsets.get(k - 1).add(nums[i]);
                if (canPartition(nums, visited, i + 1, k, currentSum + nums[i], targetSum,
subsets))
                {
                    return true;
                }
                visited[i] = false;
                subsets.get(k - 1).remove(subsets.get(k - 1).size() - 1);
            }
        }

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


        System.out.print("Enter the size of the arary: ");
        int size = sc.nextInt();
        int[] nums = new int[size];
        System.out.print("Enter the values of the array: ");
        for(int i=0 ; i<size ; i++)
        {
            nums[i] = sc.nextInt();
        }
        System.out.print("Enter the value of K: ");
        int k = sc.nextInt();

        boolean result = canPartitionKSubsets(nums, k);
        System.out.println("Is it possible to divide the array into " + k + " subsets
with equal sum? " + result);

        sc.close();
    }
}
```

OUTPUT:

```
PS V:\ATT_JAVA\Skills> cd "v:\ATT_JAVA\Skills\" ; if ($?) { javac PartitionArray.java
} ; if ($?) { java PartitionArray }

Enter the size of the arary: 4
Enter the values of the array: 1 3 2 2
Enter the value of K: 2
The subsets are:
[2, 2]
[1, 3]
Is it possible to divide the array into 2 subsets with equal sum? true
```

- **Calculate Total Sum**: Calculate the total sum of the array elements. If the total sum is not divisible by kkk, return false since we can't partition the array into kkk subsets with equal sum.
- **Target Sum**: Calculate the target sum for each subset by dividing the total sum by kkk.
- **Backtracking**:

    - **Base Case 1**: If kkk is 0, all subsets have been successfully formed. Print the subsets and return true.
    - **Base Case 2**: If the current subset sum equals the target sum, recursively partition the remaining array into k−1k - 1k−1 subsets.
    - **Recursion**: For each element in the array that has not been visited:
        - Mark the element as visited and add it to the current subset.
        - Recursively call the function with updated parameters.
        - If the recursive call returns true, return true.
        - If it fails, backtrack by unmarking the element and removing it from the subset.

- **Main Function**:

    - Initialize the input array (`nums`) and the value of kkk.

- Call the `canPartitionKSubsets` function to check and print if it's possible to divide the array into kkk subsets with equal sum.

2. **Given an integer array arr, print all the possible permutations of the given array.**

Ans.

```
package Skills;
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

public class Permutations
{


   public static void generatePermutations(int[] arr, int index, List<List<Integer>>
result)
   {
      if (index == arr.length)
      {
         List<Integer> permutation = new ArrayList<>();
         for (int num : arr)
         {
            permutation.add(num);
         }
         result.add(permutation);
      }
      else
      {
         for (int i = index; i < arr.length; i++)
         {
            swap(arr, index, i);
            generatePermutations(arr, index + 1, result);
            swap(arr, index, i); // backtrack
         }
      }
   }

     private static void swap(int[] arr, int i, int j)
   {
      int temp = arr[i];
      arr[i] = arr[j];
      arr[j] = temp;
```

```
    }

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

        System.out.print("Enter the size of the array: ");
        int size = scanner.nextInt();

        int[] arr = new int[size];
        System.out.println("Enter the elements of the array:");
        for (int i = 0; i < size; i++) {
            arr[i] = scanner.nextInt();
        }

        List<List<Integer>> result = new ArrayList<>();

        generatePermutations(arr, 0, result);

        System.out.println("All possible permutations are:");
        for (List<Integer> permutation : result) {
            System.out.println(permutation);
        }
    }
}
```

OUTPUT:

```
Enter the size of the array: 3
Enter the elements of the array: 1 2 3
All possible permutations are:
[1, 2, 3]
[1, 3, 2]
[2, 1, 3]
[2, 3, 1]
[3, 2, 1]
[3, 1, 2]
PS V:\ATT_JAVA\Skills>
```

- **Main Function**:
  - Use `Scanner` to read the size of the array and the elements from the user.
  - Initialize an empty list `result` to store all permutations.
  - Call `generatePermutations` to generate all permutations starting from index 0.
  - Print all the permutations stored in `result`.
- **generatePermutations Method**:
  - If `index` is equal to the length of the array, it means we have a complete permutation. Add this permutation to the result list.
  - Otherwise, for each position from `index` to the end of the array, swap the current element with the element at `index`, recursively generate permutations for the next index, and then backtrack by swapping the elements back to their original positions.
- **swap Method**:
  - Swap the elements at positions `i` and `j` in the array.

3. **Given a collection of numbers, nums, that might contain duplicates, return all possible unique permutations in any order.**

Ans.

```
package Skills;

import java.util.*;

public class UniquePermutations
{


   public static void generatePermutations(int[] nums, int index,
List<List<Integer>> result, Set<List<Integer>> seen) {
        if (index == nums.length)
        {
           List<Integer> permutation = new ArrayList<>();
           for (int num : nums)
           {
              permutation.add(num);
           }
           if (!seen.contains(permutation))
           {
              result.add(permutation);
              seen.add(permutation);
           }
        }
        else
        {
            for (int i = index; i < nums.length; i++)
            {
                swap(nums, index, i);
                generatePermutations(nums, index + 1, result, seen);
                swap(nums, index, i); // backtrack
            }
        }
    }


    private static void swap(int[] nums, int i, int j)
    {
        int temp = nums[i];
        nums[i] = nums[j];
        nums[j] = temp;
    }

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


        System.out.print("Enter the size of the array: ");
        int size = scanner.nextInt();


        int[] nums = new int[size];
        System.out.println("Enter the elements of the array:");
        for (int i = 0; i < size; i++)
        {
            nums[i] = scanner.nextInt();
        }


        List<List<Integer>> result = new ArrayList<>();
        Set<List<Integer>> seen = new HashSet<>();


        generatePermutations(nums, 0, result, seen);
        System.out.println("All possible unique permutations are:");
        for (List<Integer> permutation : result)
        {
            System.out.println(permutation);
        }
    }
  }
```

```
Enter the size of the array: 3
Enter the elements of the array:
1 1 2
All possible unique permutations are:
[1, 1, 2]
[1, 2, 1]
[2, 1, 1]
PS V:\ATT_JAVA\Skills>
```

- **Main Function**:
  - Use `Scanner` to read the size of the array and the elements from the user.
  - Initialize an empty list `result` to store all unique permutations.
  - Initialize a set `seen` to keep track of permutations that have already been added to `result`.
  - Call `generatePermutations` to generate all unique permutations starting from index 0.
  - Print all the unique permutations stored in `result`.
- **generatePermutations Method**:
  - If `index` is equal to the length of the array, it means we have a complete permutation. Convert this permutation to a list and add it to `result` if it hasn't been added before (checked using the `seen` set).
  - Otherwise, for each position from `index` to the end of the array, swap the current element with the element at `index`, recursively generate permutations for the next index, and then backtrack by swapping the elements back to their original positions.
- **swap Method**:
  - Swap the elements at positions `i` and `j` in the array.

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

Ans.

```
import java.util.Scanner;

public class SubsetProduct
{


   public static boolean isSubsetProduct(int[] nums, int index, int currentProduct,
int target)
   {

      if (currentProduct == target)
      {
         return true;
      }


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


      if (isSubsetProduct(nums, index + 1, currentProduct * nums[index], target))
      {
         return true;
      }


      return isSubsetProduct(nums, index + 1, currentProduct, target);
   }

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


      System.out.print("Enter the size of the array: ");
      int n = scanner.nextInt();
```

```
        System.out.print("Enter the target value: ");
        int target = scanner.nextInt();


        int[] nums = new int[n];
        System.out.println("Enter the elements of the array:");
        for (int i = 0; i < n; i++)
        {
            nums[i] = scanner.nextInt();
        }


        boolean result = isSubsetProduct(nums, 0, 1, target);


        if (result) {
            System.out.println("YES");
        } else {
            System.out.println("NO");
        }
    }
}
```

```
PS V:\ATT_JAVA\Skills> cd "v:\ATT_JAVA\Skills\" ; if ($?) { javac
SubsetProduct.java } ; if ($?) { java SubsetProduct }
Enter the size of the array: 5
Enter the target value: 16
Enter the elements of the array:
2 3 2 5 4
YES
PS V:\ATT_JAVA\Skills>
```

- **Main Function**:
  - Use `Scanner` to read the size of the array, the target value, and the elements of the array from the user.
  - Call `isSubsetProduct` to check if there exists a subset whose product equals the target value.

- Print "YES" if such a subset exists, otherwise print "NO".

- **isSubsetProduct Method**:

  - **Base Case 1**: If `currentProduct` equals the target, return true.
  - **Base Case 2**: If `index` is equal to the length of the array or `currentProduct` exceeds the target, return false.
  - **Include the Current Element**: Recursively check if including the current element in the product results in the target.
  - **Exclude the Current Element**: Recursively check if excluding the current element and moving to the next element results in the target.

5. **The n-queens puzzle is the problem of placing n queens on an (n x n) chessboard such that no two queens attack each other. Given an integer n, return the number of distinct solutions to the n-queens puzzle.**

Ans.

```
import java.util.Scanner;

public class NQueens {

    private static boolean isSafe(int[][] board, int row, int col, int n) {
        // Check this column on upper side
        for (int i = 0; i < row; i++) {
            if (board[i][col] == 1) {
                return false;
            }
        }

        for (int i = row, j = col; i >= 0 && j >= 0; i--, j--) {
            if (board[i][j] == 1) {
                return false;
            }
        }

        for (int i = row, j = col; i >= 0 && j < n; i--, j++) {
            if (board[i][j] == 1) {
                return false;
            }
        }

        return true;
    }

    // Recursive utility method to solve the n-queens problem
    private static void solveNQueens(int[][] board, int row, int n, int[] count) {
        if (row == n) {
            count[0]++;
            return;
        }

        for (int col = 0; col < n; col++) {
            if (isSafe(board, row, col, n)) {
                board[row][col] = 1;
                solveNQueens(board, row + 1, n, count);
                board[row][col] = 0; // backtrack
            }
        }
    }

    public static int totalNQueens(int n) {
        int[][] board = new int[n][n];
        int[] count = new int[1];
        solveNQueens(board, 0, n, count);
        return count[0];
    }

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


        System.out.print("Enter the value of n: ");
        int n = scanner.nextInt();

        int result = totalNQueens(n);

        System.out.println("Number of distinct solutions: " + result);
    }
}
```

```
PS V:\ATT_JAVA\Skills> cd "v:\ATT_JAVA\Skills\" ; if ($?) { javac NQueens.java }
; if ($?) { java NQueens }
Enter the value of n: 4
Number of distinct solutions: 2
```