Backtracking Assignment Questions

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.**
   Input: arr = [1,3,2,2] k = 2
   Output: true
   Explanation: 1 + 3 and 2+2 are two subsets with equal sum.

Ans :-

```java
import java.util.*;
public class Assignment_Q_1 {
    public static void main(String[]args){
        Scanner sc = new Scanner(System.in);

        System.out.println("Enter the number k");
        int k = sc.nextInt();
        System.out.println("Enter the size of array");
        int n = sc.nextInt();
        int arr [] = new int [n];
        System.out.println("Enter the array");
        int sum = 0;
```

```
        for(int i=0; i<n; i++){
            arr[i] = sc.nextInt();
            sum += arr[i];
        }


        if(sum % k == 0){
            System.out.println("True");
        }
        else{
            System.out.println("False");
        }

    }
}
```

Output :-

Enter the number k
2
Enter the size of array
4
Enter the array
1 3 2 2
True

2. **Given an integer array arr, print all the possible permutations of the given array.**
   Note: The array will only contain non repeating elements.

   Input 1: arr = [1, 2, 3]
   Output1: [[1,2,3], [1,3,2], [2,1,3], [2,3,1], [3,1,2], [3,2,1]]

   Ans :-

```java
import java.util.*;
public class Assignment_Q_2 {
    public static String swaping(String str, int i, int j){

        char c [] = str.toCharArray();
        char temp = c [i];
        c [i] = c[j];
        c [j] = temp;

        return String.valueOf(c);
    }

    public static void permutation(String str, int l , int r){
        if(l == r){
            System.out.println(str);
        }
        else{
            for(int i = l; i<=r; i++){
                str = swaping(str, l, i);
                permutation(str, l + 1, r);

                str = swaping(str, l, i);
            }
        }
    }
    public static void main(String[]args){
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the array which permutation do you
want");
        String str = sc.nextLine();

        System.out.println("All the possible combination of the
permutation are : ");

        int n = str.length();
```

```
        permutation(str, 0, n-1);
    }

}
```

Output :-

Enter the array which permutation do you want
123
All the possible combination of the permutation are :
123
132
213
231
321
312

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

   Example 1:
   Input: nums = [1,1,2]
   Output:
   [[1,1,2], [1,2,1], [2,1,1]]

   Ans :-

```
import java.util.*;

public class Assignment_Q_3 {
    public static String swaping(String str, int i, int j) {
        char[] c = str.toCharArray();
        char temp = c[i];
        c[i] = c[j];
```

```
        c[j] = temp;
        return String.valueOf(c);
    }

    public static void permutation(String str, int l, int r, Set<String>
result) {
        if (l == r) {
            if (!result.contains(str)) {
                result.add(str);
                System.out.println(str);
            }
        } else {
            for (int i = l; i <= r; i++) {
                str = swaping(str, l, i);
                permutation(str, l + 1, r, result);
                str = swaping(str, l, i);
            }
        }
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the array which permutation do you
want");
        String str = sc.nextLine();

        System.out.println("All the possible combinations of the
permutation are:");

        int n = str.length();

        Set<String> result = new HashSet<>();
        permutation(str, 0, n - 1, result);
    }
}
```

Output :-

Enter the array which permutation do you want
112
All the possible combinations of the permutation are:
112
121
211

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

   Input: n = 5, target = 16
   Array = [23254]
   Here the target will be equal to 2x2x4 = 16
   Output: YES

   Ans :-

```
import java.util.*;
public class Assignment_Q_4 {
    public static void main(String[]args){
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the size of the array");
        int n = sc.nextInt();
        int []ar = new int [n];
        System.out.println("Enter the target");
        int target = sc.nextInt();
        System.out.println("Enter the elements present in the array");
        int mul = 1;
        for(int i=0; i<n; i++){
            ar[i] = sc.nextInt();
            mul *= ar[i];
        }

        if(mul % target == 0){
```

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

Output :-

Enter the size of the array
5
Enter the target
16
Enter the elements present in the array
2 3 2 5 4
YES

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.**

   Input: n = 4
   Output: 2
   Explanation: There are two distinct solutions to the 4-queens puzzle as shown.
   Input: n = 1
   Output: 1

   Ans :-

```
import java.util.*;
```

```java
public class Assignment_Q_5 {

    int n;
    int solutionCount;

    Assignment_Q_5(int n) {
        this.n = n;
        this.solutionCount = 0;
    }

    public int findSolutions() {
        int[][] sol = new int[n][n];
        queenProblemUtil(0, sol);
        return solutionCount;
    }

    public boolean queenProblemUtil(int col, int[][] sol) {
        if (col >= n) {
            // Found a solution
            solutionCount++;
            return true;  // Continue to find other solutions
        }

        boolean hasSolution = false;

        for (int row = 0; row < n; row++) {
            if (isSafeToPlace(row, col, sol)) {
                sol[row][col] = 1;

                if (queenProblemUtil(col + 1, sol)) {
                    hasSolution = true;
                }

                // Backtrack
                sol[row][col] = 0;
            }
        }
```

```
        return hasSolution;
    }

    public boolean isSafeToPlace(int row, int col, int[][] sol) {
        int i, j;

        // Check this row on left side
        for (i = 0; i < col; i++) {
            if (sol[row][i] == 1) {
                return false;
            }
        }

        // Check upper diagonal on left side
        for (i = row, j = col; i >= 0 && j >= 0; i--, j--) {
            if (sol[i][j] == 1) {
                return false;
            }
        }

        // Check lower diagonal on left side
        for (i = row, j = col; i < n && j >= 0; i++, j--) {
            if (sol[i][j] == 1) {
                return false;
            }
        }

        return true;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the value of n");
        int n = sc.nextInt();
        Assignment_Q_5 prob = new Assignment_Q_5(n);
        int result = prob.findSolutions();
        System.out.println(result);
```

```
    }
}
```

Output :-

Enter the value of n
4
2