Backtracking

Assignment Questions

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

Ans:

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

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

        int target = sum / k;
        boolean[] visited = new boolean[arr.length];
        Arrays.sort(arr);

        return backtrack(arr, visited, k, target, 0, 0);
    }

    private boolean backtrack(int[] arr, boolean[] visited, int k, int target, int
startIndex, int currentSum) {
        if (k == 0) {
            return true;
        }
```

```
        if (currentSum == target) {
            return backtrack(arr, visited, k - 1, target, 0, 0);
        }

        for (int i = arr.length - 1; i >= startIndex; i--) {
            if (!visited[i] && currentSum + arr[i] <= target) {
                visited[i] = true;
                if (backtrack(arr, visited, k, target, i + 1, currentSum + arr[i])) {
                    return true;
                }
                visited[i] = false;
            }
        }
```

Q2. Given an integer array arr, print all the possible permutations of the given array.
Note : The array will only contain non repeating elements.
Input: arr = [1, 2, 3]
Output:[[1,2,3] , [1,3,2] , [2,1,3] , [2,3,1] , [3,1,2] , [3,2,1]]

Ans:

```
public class Permutations {
    public void permute(int[] arr) {
        permuteHelper(arr, 0, arr.length - 1);
    }

    private void permuteHelper(int[] arr, int left, int right) {
        if (left == right) {
            printArray(arr);
        } else {
            for (int i = left; i <= right; i++) {
                swap(arr, left, i);
                permuteHelper(arr, left + 1, right);
```

```
                swap(arr, left, i); // backtrack
            }
        }
    }

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

    private void printArray(int[] arr) {
        System.out.println(Arrays.toString(arr));
    }

    public static void main(String[] args) {
        Permutations permutations = new Permutations();
        int[] arr = {1, 2, 3};
        permutations.permute(arr);
    }
}
```

Q3. Given a collection of numbers, nums, that might contain duplicates, return all possible unique permutations in any order.

Example1:
Input:nums= [1,1,2]
Output[[1,1,2], [1,2,1], [2,1,1]]

Example 2:
Input: nums = [1,2,3]
Output: [[1,2,3],[1,3,2],[2,1,3],[2,3,1],[3,1,2],[3,2,1]]

Ans:

```
public class PermutationsII {
```

```
    public List<List<Integer>> permuteUnique(int[] nums) {
        List<List<Integer>> result = new ArrayList<>();
        Arrays.sort(nums);
        backtrack(result, new ArrayList<>(), nums, new boolean[nums.length]);
        return result;
    }

    private void backtrack(List<List<Integer>> result, List<Integer> tempList,
int[] nums, boolean[] used) {
        if (tempList.size() == nums.length) {
            result.add(new ArrayList<>(tempList));
        } else {
            for (int i = 0; i < nums.length; i++) {
                if (used[i] || (i > 0 && nums[i] == nums[i - 1] && !used[i - 1])) {
                    continue;
                }
                used[i] = true;
                tempList.add(nums[i]);
                backtrack(result, tempList, nums, used);
                used[i] = false;
                tempList.remove(tempList.size() - 1);
            }
        }
    }
}
```

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

Input :n = 5 , target = 16
Array = [2 3 2 5 4]

Output :YES

Ans:

```
public class SubsetProduct {
    public String isSubsetProduct(int[] arr, int target) {
        return backtrack(arr, target, 0, 1) ? "YES" : "NO";
    }

    private boolean backtrack(int[] arr, int target, int index, long product) {
        if (product == target) {
            return true;
        }
        if (product > target || index == arr.length) {
            return false;
        }
        // Include current element
        if (backtrack(arr, target, index + 1, product * arr[index])) {
            return true;
        }
        // Exclude current element
        return backtrack(arr, target, index + 1, product);
    }
}
```

Q5. 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:

```
public class NQueens {
    public int totalNQueens(int n) {
        int[] queens = new int[n];
        return placeQueens(queens, 0);
    }

    private int placeQueens(int[] queens, int row) {
        if (row == queens.length) {
            return 1;
        }

        int count = 0;
        for (int col = 0; col < queens.length; col++) {
            if (isValid(queens, row, col)) {
                queens[row] = col;
                count += placeQueens(queens, row + 1);
            }
        }
        return count;
    }

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