Backtracking Assignment

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.

Ans :-

```java
import java.util.*;

import java.util.Scanner;
public class Backtracking{

   public static boolean helper(int []arr , int []vis , int si , int curr_sum
, int target , int k)
    {
        if(k==1)return true;
        if(curr_sum>target) return false;
        if(curr_sum==target)return helper(arr,vis,0,0,target,k-1);

        for(int i=si;i<arr.length;i++){
            if(vis[i]==-1){
                vis[i]=1;
                if(helper(arr,vis,i+1,curr_sum+arr[i],target,k) == true)return
true;
                vis[i]=-1;
            }
        }
        return false;
    }
    public static boolean canPartition(int []arr, int k) {
        int n=arr.length;
        int []vis = new int[n];
        for(int i=0;i<n;i++)vis[i] = -1;
        int sum=0;
        for(int i=0;i<n;i++)sum+=arr[i];

        if(sum%k!=0)return false;
        return helper(arr,vis,0,0,sum/k,k);
    }
    public static void main(String[] args){
        int []arr = {1 , 2 , 2 , 3};
        int n = 4;
        int k = 2;

        if(canPartition(arr , k) == true){
            System.out.println("yes it is possible to partition the array.");
        }
        else System.out.println("no it is not possible to partition.");
    }
}
```

Q2. Given an integer array arr, print all the possible permutations of the given array.

Ans ;

```java
import java.util.*;

import java.util.Scanner;
public class Backtracking_1{

   public static void permute(int[] nums) {
        List<List<Integer>> result = new ArrayList<>();
        if (nums == null || nums.length == 0) {
          return;
        }

        permutationsHelper(result, nums, 0);

        for(List<Integer> list : result){
            for(Integer item : list){
                System.out.print(item + "  ");
            }
            System.out.println("");
        }
    }

    private static void permutationsHelper(List<List<Integer>> result, int[]
nums, int start) {
        if (start == nums.length - 1) {
            List<Integer> list = new ArrayList<>();
            for (int n : nums) {
                list.add(n);
            }
            result.add(list);
            return;
        }
        for (int i = start; i < nums.length; i++) {
            swap(nums, start, i);
            permutationsHelper(result, nums, start + 1);
            swap(nums, start, i);
        }
    }

    private static void swap(int[] nums, int x, int y) {
        int t = nums[x];
        nums[x] = nums[y];
        nums[y] = t;
    }
    public static void main(String[] args){
        int []arr = {1 , 4 , 2 , 3};
        System.out.println("The possible permutations are : ");
```

```
        permute(arr);
    }
}
```

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

Ans :
```
import java.util.*;

import java.util.Scanner;
public class Backtracking_2{
   public static void permuteUnique(int[] nums) {
        List<List<Integer>> result = new ArrayList<List<Integer>>();
        if(nums==null || nums.length==0) return ;
        boolean[] used = new boolean[nums.length];
        List<Integer> list = new ArrayList<Integer>();
        Arrays.sort(nums);
        go(nums, used, list, result);

        for(List<Integer> li : result){
            for(Integer item : li){
                System.out.print(item + "  ");
            }
            System.out.println("");
        }
    }

    public static void go(int[] nums, boolean[] used, List<Integer> list,
List<List<Integer>> res){
        if(list.size()==nums.length){
            res.add(new ArrayList<Integer>(list));
            return;
        }
        for(int i=0;i<nums.length;i++){
            if(used[i]) continue;
            if(i>0 &&nums[i-1]==nums[i] && !used[i-1]) continue;
            used[i]=true;
            list.add(nums[i]);
            go(nums,used,list,res);
            used[i]=false;
            list.remove(list.size()-1);
        }
    }

    public static void main(String[] args){
        int []arr = {1 , 4 , 4 , 3};
        System.out.println("The possible permutations are : ");
```

```
        permuteUnique(arr);
    }
}
```

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

```
Ans import java.util.*;



public class Backtracking_3 {
   static int n;
   public static boolean solve(int n, int target, int a[], int i, int product)
{
       if (i == n) return (product == target);
       boolean answer = false;

       product *= a[i];

       answer |= solve(n, target, a, i + 1, product);

       product /= a[i];

       answer |= solve(n, target, a, i + 1, product);

       return answer;
   }
   public static void main(String[] args) {

       Scanner sc = new Scanner(System.in);
       System.out.println("Enter the number of elements you want and the value
of target respectively : ");

       int n = sc.nextInt(), target = sc.nextInt();
       int a[] = new int[n];
       System.out.println("Enter the array elements");
       for (int i = 0; i < n; i++) {
           a[i] = sc.nextInt();
       }
       System.out.println((solve(n, target, a, 0, 1) ? "YES" : "NO"));
   }
}
```

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.

Ans ;-

```java
import java.util.*;


public class Backtracking_4 {
    public static int totalNQueens(int n) {
        char board[][] = new char[n][n];
        for(char i[] : board)
            Arrays.fill(i, '.');
        return go(0, board);
    }
    public static int go(int col, char board[][]){
        if(col == board.length) return 1;
        int count = 0;
        for(int row = 0; row < board.length; row++){
            if(isSafe(board, row, col)){
                board[row][col] = 'Q';
                count += go(col + 1, board);
                board[row][col] = '.';
            }
        }
        return count;
    }
    public static boolean isSafe(char board[][], int row, int col){
        int dupRow = row;
        int dupCol = col;

        while(row >= 0 && col >= 0){
            if(board[row][col] == 'Q') return false;
            row--;
            col--;
        }

        row = dupRow;
        col = dupCol;
        while(col >= 0){
            if(board[row][col] == 'Q') return false;
            col--;
        }

        row = dupRow;
        col = dupCol;
        while(col >= 0 && row < board.length){
            if(board[row][col] == 'Q') return false;
            row++;
            col--;
        }
        return true;
    }
```

```
    public static void main(String[] args) {
        int n = 4;
       System.out.println("The desired answer is : " + totalNQueens(n));
    }
}
```