**Q1. Running Sum of 1d Array Given an array nums. We define a running sum of an array as runningSum[i] = sum(nums[0]…nums[i]). Return the running sum of nums.**

class Solution {

    public int[] runningSum(int[] nums) {

        for(int i=1;i<nums.length;i++)

        {

            nums[i]=nums[i-1]+nums[i];

        }

        return nums;

    }

}

**Q2. Number of Good Pairs Given an array of integers nums, return the number of good pairs. A pair (i, j) is called good if nums[i] == nums[j] and i < j.**

class Solution {

    public int numIdenticalPairs(int[] nums) {

        int a=0;

        for(int i=0;i<nums.length;i++)

        {

            int b=nums[i];

            for(int j=i+1;j<nums.length;j++)

            {

                if(nums[i]==nums[j])

                {

                    a++;

                }

            }

        }

        return a;

    }

}

**Q3. Find Greatest Common Divisor of Array Given an integer array nums, return the greatest common divisor of the smallest number and largest number in nums. The greatest common divisor of two numbers is the largest positive integer that evenly divides both numbers.**

class Solution {

    public int findGCD(int[] nums) {

        Arrays.sort(nums);

        int max=nums[nums.length-1];

        int min=nums[0];

        int Temp;

             while(min != 0){

                        Temp = min;

                        min = max % min;

                        max = Temp;

                  }

        return max;

    }

}

**Q4. Unique Number of Occurrences Given an array of integers arr, return true if the number of occurrences of each value in the array is unique or false otherwise.**

class Solution {

    public boolean uniqueOccurrences(int[] arr) {

        Map<Integer, Integer> a = new HashMap();

        for (int i : arr)  a.put(i, a.getOrDefault(i, 0) + 1);

        Set<Integer> uv = new HashSet<>(a.values());

        return uv.size() == a.size();

    }

}

}

**Q5. Divide Array Into Equal Pairs You are given an integer array nums consisting of 2 \* n integers. You need to divide nums into n pairs such that: Each element belongs to exactly one pair. The elements present in a pair are equal. Return true if nums can be divided into n pairs, otherwise return false.**

class Solution {

    public boolean divideArray(int[] nums) {

        Arrays.sort(nums);

        for(int i=0;i<nums.length-1;i++)

        {

            if(nums[i]==nums[i+1])

            {

                i++;

                continue;

            }

           else

                return false;

        }

        return true;

    }

}

**Q6. Find the Duplicate Number Given an array of integers nums containing n + 1 integers where each integer is in the range [1, n] inclusive. There is only one repeated number in nums, return this repeated number. You must solve the problem without modifying the array nums and uses only constant extra space.**

class Solution {

    public int findDuplicate(int[] nums) {

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

        for(int i:nums){

            if(arr[i]) return i;

            arr[i] = true;

        }

        return -1;

    }

}

**Q7. Find All Duplicates in an Array Given an integer array nums of length n where all the integers of nums are in the range [1, n] and each integer appears once or twice, return an array of all the integers that appears twice. You must write an algorithm that runs in O(n) time and uses only constant extra space.**

class Solution {

    public List<Integer> findDuplicates(int[] nums) {

       int[] table = new int[nums.length+1];

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

       for(int num : nums){

           table[num]++;

       }

       for(int i = 0 ; i < table.length ; i++){

           if(table[i] >= 2){

               res.add(i);

           }

       }

      return res;

    }

}

**Q8. Find Peak Element A peak elemenA peak element is an element that is strictly greater than its neighbors. Given a 0-indexed integer array nums, find a peak element, and return its index. If the array contains multiple peaks, return the index to any of the peaks. You may imagine that nums[-1] = nums[n] = -∞. In other words, an element is always considered to be strictly greater than a neighbor that is outside the array. You must write an algorithm that runs in O(log n) time. t is an element that is strictly greater than**

class Solution {

    public int findPeakElement(int[] nums) {

        for(int i=nums.length-1;i>=1;i--){

            if(nums[i-1]<nums[i]){

                return i;

            }

        }

        return 0;

    }

}