----ARRAYS-----

1.TWO SUMS

Given an array of integers nums and an integer target, return *indices of the two numbers such that they add up to target*.

You may assume that each input would have ***exactly* one solution**, and you may not use the *same* element twice.

You can return the answer in any order.

**Example 1:**

**Input:** nums = [2,7,11,15], target = 9

**Output:** [0,1]

class Solution {

    public  static int[] twoSum(int[] nums, int target) {

        int ans[]=new int[2];

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

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

                    if(nums[i]+nums[j]==target){

                        ans[0]=i;

                        ans[1]=j;

                    }

                }

            }

            return ans;

    }

}

2.PALINDROME NUMBER(9)

Given an integer x, return true*if*x*is a****palindrome*** *and*false*otherwise*.

**Example 1:**

**Input:** x = 121

**Output:** true

class Solution {

    public boolean isPalindrome(int num) {

        if(num<0 || (num!=0 && num%10==0) ){

         return false;

        }

         int ans=0;

        while(num>ans){

        ans=ans\*10+num%10;

        num/=10;

        }

        return (num==ans || num==ans/10);

          }

}

3.Find Numbers with Even Number of Digits(1295)

Given an array nums of integers, return how many of them contain an **even number** of digits.

**Example 1:**

**Input:** nums = [12,345,2,6,7896]

**Output:** 2

class Solution {

    public int findNumbers(int[] arr) {

        int count=0;

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

            if(evendigits(arr[i])){

                count++;

            }

        }

        return count;

    }

     static boolean evendigits(int num){

        int ans=digits(num);

        if(ans%2==0){

            return true;

        }

        return false;

    }

 static  int digits(int num){

        int count=0;

        while(num>0){

            count++;

            num/=10;

        }

        return count;

    }

}

4.Richest Customer Wealth(1672)

You are given an m x n integer grid accounts where accounts[i][j] is the amount of money the i​​​​​​​​​​​th​​​​ customer has in the j​​​​​​​​​​​th​​​​ bank. Return the ***wealth*** that the richest customer has.

A customer's **wealth** is the amount of money they have in all their bank accounts. The richest customer is the customer that has the maximum **wealth**.

**Example 1:**

**Input:** accounts = [[1,2,3],[3,2,1]]

**Output:** 6

class Solution {

    public int maximumWealth(int[][] arr) {

         int max=Integer.MIN\_VALUE;

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

             int sum=0;

            for(int j=0;j<arr[i].length;j++){

                sum+=arr[i][j];

        }

         if(sum>max){

            max=sum;

        }

        }

        return max;

    }

}

5.Average Value of Even Numbers That Are Divisible by Three(2455)

Given an integer array nums of **positive** integers, return the average value of all even integers that are divisible by 3*.*

Note that the **average** of n elements is the **sum** of the n elements divided by n and **rounded down** to the nearest integer.

**Example 1:**

**Input:** nums = [1,3,6,10,12,15]

**Output:** 9

class Solution {

    public int averageValue(int[] nums) {

         int sum=0;

        int count=0;

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

             if(nums[i]%2==0 && nums[i]%3==0){

            sum+=nums[i];

            count++;

                }

             }

             if(count==0){

                 return 0;

             }

        return sum/count;

    }

}

6.Reverse Integer(7)

Given a signed 32-bit integer x, return x with its digits reversed. If reversing x causes the value to go outside the signed 32-bit integer range [-231, 231 - 1], then return 0.

**Assume the environment does not allow you to store 64-bit integers (signed or unsigned).**

**Example 1:**

**Input:** x = 123

**Output:** 321

class Solution {

    public int reverse(int x) {

        boolean ans=true;

        long rev=0;

        if(x<0){

            ans=false;

        }

        x=Math.abs(x);

        while(x>0){

            int rem=x%10;

            rev=rev\*10+rem;

            x/=10;

              if(rev>Integer.MAX\_VALUE || rev<Integer.MIN\_VALUE){

           return 0;

        }

        }

        if(ans==false){

            return -1\*(int)rev;

        }

        return (int)rev;

    }

}

7.Plus One (66)

You are given a **large integer** represented as an integer array digits, where each digits[i] is the ith digit of the integer. The digits are ordered from most significant to least significant in left-to-right order. The large integer does not contain any leading 0's.

Increment the large integer by one and return the resulting array of digits.

**Example 1:**

**Input:** digits = [1,2,3]

**Output:** [1,2,4]

class Solution {

    public int[] plusOne(int[] digits) {

       for(int i=digits.length-1;i>=0;i--){

           if(digits[i]+1<10){

           digits[i]++;

           return digits;

       }

       else{

           digits[i]=0;

       }

    }

    int[] arr=new int[digits.length+1];

        arr[0]=1;

        return arr;

    }

}

8.Build Array from Permutation(1920)

Given a **zero-based permutation** nums (**0-indexed**), build an array ans of the **same length** where ans[i] = nums[nums[i]] for each 0 <= i < nums.length and return it.

A **zero-based permutation** nums is an array of **distinct** integers from 0 to nums.length - 1 (**inclusive**).

**Example 1:**

**Input:** nums = [0,2,1,5,3,4]

**Output:** [0,1,2,4,5,3]

class Solution {

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

       int[] ans=new int[nums.length];

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

           ans[i]=nums[nums[i]];

       }

       return ans;

    }

}

9.Concatenation of Array(1929)

Given an integer array nums of length n, you want to create an array ans of length 2n where ans[i] == nums[i] and ans[i + n] == nums[i] for 0 <= i < n (**0-indexed**).

Specifically, ans is the **concatenation** of two nums arrays.

Return the array ans.

**Example 1:**

**Input:** nums = [1,2,1]

**Output:** [1,2,1,1,2,1]

class Solution {

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

       int[] ans=new int[nums.length\*2];

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

        ans[i]=ans[i+nums.length]=nums[i];

       }

       return ans;

    }

}

 10.Running Sum of 1d Array(1480)

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.

**Example 1:**

**Input:** nums = [1,2,3,4]

**Output:** [1,3,6,10]

class Solution {

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

        int sum=0;

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

            sum+=nums[i];

             nums[i]=sum;

        }

        return nums;

    }

}