**{1}TWO SUM**

class Solution {

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

int size=nums.length;

for(int i=0;i<size;i++){

for(int j=0;j<size;j++){

if(i==j){

continue;

}

int iElement=nums[i];

int jElement=nums[j];

int sum=iElement+jElement;

if(sum==target){

int[]answer=new int[2];

answer[0]=i;

answer[1]=j;

return answer;

}

}

}

int[]randomReturn=new int[2];

return randomReturn;

}

}

answer:--

Example 1:

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

Output: [0,1]

Explanation: Because nums[0] + nums[1] == 9, we return [0, 1].

Example 2:

Input: nums = [3,2,4], target = 6

Output: [1,2]

Example 3:

Input: nums = [3,3], target = 6

Output: [0,1]

**{2} Palindrome Number**

Given an integer x, return true if x is a

palindrome

, and false otherwise.

class Solution {

public boolean isPalindrome(int x) {

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

return false;

}

int rno=0;

while(x>rno){

rno=rno\*10+x%10;

x/=10;

}

return x==rno||x==rno/10;

}

}

ANSWER:==

Example 1:

Input: x = 121

Output: true

Explanation: 121 reads as 121 from left to right and from right to left.

Example 2:

Input: x = -121

Output: false

Explanation: From left to right, it reads -121. From right to left, it becomes 121-. Therefore it is not a palindrome.

Example 3:

Input: x = 10

Output: false

Explanation: Reads 01 from right to left. Therefore it is not a palindrome.

**{3}. Roman to Integer**

Roman numerals are represented by seven different symbols: I, V, X, L, C, D and M.

Symbol Value

I 1

V 5

X 10

L 50

C 100

D 500

M 1000

class Solution {

public int romanToInt(String s) {

Map<Character,Integer>map=new HashMap<>();

map.put('I',1);

map.put('V',5);

map.put('X',10);

map.put('L',50);

map.put('C',100);

map.put('D',500);

map.put('M',1000);

int result=map.get(s.charAt(s.length()-1));

for(int i=s.length()-2;i>=0;i--){

if(map.get(s.charAt(i))<map.get(s.charAt(i+1))){

result-=map.get(s.charAt(i));

} else{

result+=map.get(s.charAt(i));

}

}

return result;

}

}

ANSER:==

Example 1:

Input: s = "III"

Output: 3

Explanation: III = 3.

Example 2:

Input: s = "LVIII"

Output: 58

Explanation: L = 50, V= 5, III = 3.

Example 3:

Input: s = "MCMXCIV"

Output: 1994

Explanation: M = 1000, CM = 900, XC = 90 and IV = 4.

**{4}. Longest Common Prefix**

Write a function to find the longest common prefix string amongst an array of strings.

If there is no common prefix, return an empty string "".

class Solution {

public String longestCommonPrefix(String[] strs) {

if(strs.length==0)return "";

String prefix=strs[0];

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

while(strs[i].indexOf(prefix)!=0){

prefix=prefix.substring(0,prefix.length()-1);

}

}

return prefix;

}

}

ANSWER:==

Example 1:

Input: strs = ["flower","flow","flight"]

Output: "fl"

Example 2:

Input: strs = ["dog","racecar","car"]

Output: ""

Explanation: There is no common prefix among the input strings.

**{5}. Plus One**

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.

class Solution {

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

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

if(digits[i]==9){

digits[i]=0;

}

else{

digits[i]++;

return digits;

}

}

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

digits[0]=1;

return digits;

}

}

ANSWER:==

Example 1:

Input: digits = [1,2,3]

Output: [1,2,4]

Explanation: The array represents the integer 123.

Incrementing by one gives 123 + 1 = 124.

Thus, the result should be [1,2,4].

Example 2:

Input: digits = [4,3,2,1]

Output: [4,3,2,2]

Explanation: The array represents the integer 4321.

Incrementing by one gives 4321 + 1 = 4322.

Thus, the result should be [4,3,2,2].

Example 3:

Input: digits = [9]

Output: [1,0]

Explanation: The array represents the integer 9.

Incrementing by one gives 9 + 1 = 10.

Thus, the result should be [1,0].

**{6}Add Binary**

Given two binary strings a and b, return their sum as a binary string.

class Solution {

public String addBinary(String a, String b) {

StringBuilder sb=new StringBuilder();

int i=a.length()-1;

int j=b.length()-1;

int carry=0;

while(i>=0||j>=0){

int sum=carry;

if(i>=0)

{

sum+=a.charAt(i)-'0';

}

if(j>=0)

{

sum+=b.charAt(j)-'0';

}

sb.append(sum%2);

carry=sum/2;

i--;

j--;

}

if(carry!=0)

{

sb.append(carry);

}

return sb.reverse().toString();

}

}

ANSWER:==

Example 1:

Input: a = "11", b = "1"

Output: "100"

Example 2:

Input: a = "1010", b = "1011"

Output: "10101"

**{7}Happy Number**

Write an algorithm to determine if a number n is happy.

A happy number is a number defined by the following process:

Starting with any positive integer, replace the number by the sum of the squares of its digits.

Repeat the process until the number equals 1 (where it will stay), or it loops endlessly in a cycle which does not include 1.

Those numbers for which this process ends in 1 are happy.

Return true if n is a happy number, and false if not.

class Solution {

public boolean isHappy(int n) {

HashSet<Integer>hs=new HashSet<>();

while(!hs.contains(n)){

hs.add(n);

int sqrSum=0;

while(n!=0){

int rem=n%10;

sqrSum+=rem\*rem;

n=n/10;

}

if(sqrSum==1)

{

return true;

}

else{

n=sqrSum;

}

}

return false;

}

}

AANSWER:==

Example 1:

Input: n = 19

Output: true

Explanation:

12 + 92 = 82

82 + 22 = 68

62 + 82 = 100

12 + 02 + 02 = 1

Example 2:

Input: n = 2

Output: false

**{8}Excel Sheet Column Number**

Given a string columnTitle that represents the column title as appears in an Excel sheet, return its corresponding column number.

For example:

A -> 1

B -> 2

C -> 3

...

Z -> 26

AA -> 27

AB -> 28

...

class Solution {

public int titleToNumber(String columnTitle) {

char[]c=columnTitle.toCharArray();

int result=0;

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

int num=c[i]-65+1;

result=result\*26+num;

}

return result;

}

}

ANSWER:==

Example 1:

Input: columnTitle = "A"

Output: 1

Example 2:

Input: columnTitle = "AB"

Output: 28

Example 3:

Input: columnTitle = "ZY"

Output: 701

**{9} Excel Sheet Column Title**

Given an integer columnNumber, return its corresponding column title as it appears in an Excel sheet.

For example:

A -> 1

B -> 2

C -> 3

...

Z -> 26

AA -> 27

AB -> 28

...

class Solution {

public String convertToTitle(int columnNumber) {

StringBuilder sb=new StringBuilder();

while(columnNumber!=0){

columnNumber--;

sb.append((char)('A'+columnNumber%26));

columnNumber /=26;

}

sb.reverse();

return sb.toString();

}

}

ANSWER:==

Example 1:

Input: columnNumber = 1

Output: "A"

Example 2:

Input: columnNumber = 28

Output: "AB"

Example 3:

Input: columnNumber = 701

Output: "ZY"

**{9}Sqrt(x)**

Given a non-negative integer x, return the square root of x rounded down to the nearest integer. The returned integer should be non-negative as well.

You must not use any built-in exponent function or operator.

For example, do not use pow(x, 0.5) in c++ or x \*\* 0.5 in python.

class Solution {

public int mySqrt(int x) {

int start=1, end=x,floorRes=0;

while(start<=end){

int mid=start+(end-start)/2;

if(mid<=x/mid){

floorRes=mid;

start=mid+1;

}

else{

end=mid-1;

}

}

return floorRes;

}

}

ANSWER:==

Example 1:

Input: x = 4

Output: 2

Explanation: The square root of 4 is 2, so we return 2.

Example 2:

Input: x = 8

Output: 2

Explanation: The square root of 8 is 2.82842..., and since we round it down to the nearest integer, 2 is returned.

**{10}Climbing Stairs**

You are climbing a staircase. It takes n steps to reach the top.

Each time you can either climb 1 or 2 steps. In how many distinct ways can you climb to the top?

class Solution {

public int climbStairs(int n) {

if(n<=2){

return n;

}

int arr[] = new int[n+1];

arr[0] = 0;

arr[1] = 1;

arr[2] = 2;

for(int i=3;i<arr.length;i++)

{

arr[i] = arr[i-1] + arr[i-2];

}

return arr[n];

}

}

ANSWER:==

Example 1:

Input: n = 2

Output: 2

Explanation: There are two ways to climb to the top.

1. 1 step + 1 step

2. 2 steps

Example 2:

Input: n = 3

Output: 3

Explanation: There are three ways to climb to the top.

1. 1 step + 1 step + 1 step

2. 1 step + 2 steps

3. 2 steps + 1 step