**LEETCODE PROBLEMS**

**1. Plus one**

**Example :**

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].

**Solution:**

class Solution {

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

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

if (digits[i] < 9) {

digits[i]++;

return digits;

} else {

digits[i] = 0;

}

}

// If all digits were 9 (e.g., [9,9,9]), a new array with an extra digit is needed

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

newDigits[0] = 1; // The carry becomes the new most significant digit

return newDigits;

}

}

**2. Single Number**

**Example :**

Input: nums = [2,2,1]

Output: 1

**Solution:**

class Solution {

public int singleNumber(int[] nums) {

int result = 0;

for (int num : nums) {

result = result ^ num;

}

return result;

}

}

**3. Happy Number**

**Example 1:**

Input: n = 19

Output: true

Explanation:

12 + 92 = 82

82 + 22 = 68

62 + 82 = 100

12 + 02 + 02 = 1

**Solution:**

public class HappyNumber {

public static boolean isHappy(int n) {

while (n != 1 && n != 4) {

int sum = 0;

while (n > 0) {

int digit = n % 10;

sum += digit \* digit;

n /= 10;

}

n = sum;

}

return n == 1;

}

**4. Contains Duplicate**

**Example 1:**

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

Output: true

Explanation:

The element 1 occurs at the indices 0 and 3.

**Solution:**

class Solution {

public boolean containsDuplicate(int[] nums) {

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

for (int num : nums) {

if (map.containsKey(num)) {

return true;

} else {

map.put(num, 1);

}

}

return false;

}

}

**5. Add digit**

**Example 1:**

Input: num = 38

Output: 2

Explanation: The process is

38 --> 3 + 8 --> 11

11 --> 1 + 1 --> 2

Since 2 has only one digit, return it.

**Solution:**

class Solution {

public int addDigits(int num) {

int sum = 0, digit;

while (num > 9) {

sum = 0;

while (num != 0) {

digit = num % 10;

sum += digit;

num /= 10;

}

num = sum;

}

return num;

}

}

**6.Ugly number**

**Example 1:**

Input: n = 6

Output: true

Explanation: 6 = 2 × 3

**Example 2:**

Input: n = 1

Output: true

Explanation: 1 has no prime factors.

**Solution:**

class Solution {

public boolean isUgly(int n) {

if (n <= 0) return false;

while (n % 2 == 0) {

n /= 2;

}

while (n % 3 == 0) {

n /= 3;

}

while (n % 5 == 0) {

n /= 5;

}

return n == 1;

}

}

**7.Missing number**

**Example 1:**

Input: nums = [3,0,1]

Output: 2

Explanation: n = 3 since there are 3 numbers, so all numbers are in the range [0,3]. 2 is the missing number in the range since it does not appear in nums.

**Solution:**

class Solution {

public int missingNumber(int[] nums) {

int sum=0;

int n = nums.length;

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

sum+=nums[i];

}

int original = (n\*(n+1))/2;

return (original-sum);

}

}

**8.Power of four**

**Example 1:**

Input: n = 16

Output: true

**Solution**

class Solution {

public boolean isPowerOfFour(int n) {

if(n<1) return false;

while(n%4==0){

n/=4;

}

return (n==1);

}

}