Leet Code

263. Ugly Number

An ugly number is a *positive* integer which does not have a prime factor other than 2, 3, and5.

Given an integer n, return true *if* n *is an* ***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.

Example 3:

Input: n = 14

Output: false

Explanation: 14 is not ugly since it includes the prime factor 7.

Java Solution

```
class Solution
{
public boolean isUgly(int n)
{
while(n>1)
{
if(n%2==0)
n=n/2;
else if(n%3==0)
n=n/3;
else if(n%5==0)
n=n/5;
else
return false;
}
return (n==1);
}
}
```

204. Count Primes

Given an integer n, return *the number of prime numbers that are strictly less than* n.

Example 1:

Input: n = 10

Output: 4

Explanation: There are 4 prime numbers less than 10, they are 2, 3, 5, 7.

Example 2:

Input: n = 0

Output: 0

Example 3:

Input: n = 1

Output: 0

Java Solution:-

```
class Solution {
public int countPrimes(int n) {
boolean[] prime=new boolean[n];
int  c=0;
for(int i=2;i*i<=n;i++)
{
if(!prime[i])
{
for(int j=i*i;j<n;j=j+i)
prime[j]=true;
}
}
for(int i=2;i<n;i++)
{
if(!prime[i])
c++;
}
return c;
}
}
```

202. 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.*

Example 1:

Input: n = 19

Output: true

Explanation:

$1^2 + 9^2 = 82$

$8^2 + 2^2 = 68$

$6^2 + 8^2 = 100$

$1^2 + 0^2 + 0^2 = 1$

Example 2:

Input: n = 2

Output: false

Java Solution:-

```
class Solution {
public boolean isHappy(int n) {
while(n!=4){
if(n==1)
return true;
n=sumdigits(n);
}
return false;
}
int sumdigits(int n)
{
int res=0;
while(n!=0)
{
res=res+(n%10)*(n%10);
n=n/10;
}
return res;
}
}
```

50. Pow(x, n)

Implement pow(x, n), which calculates x raised to the power n (i.e., x^n).

Example 1:

Input: x = 2.00000, n = 10

Output: 1024.00000

Example 2:

Input: x = 2.10000, n = 3

Output: 9.26100

Example 3:

Input: x = 2.00000, n = -2

Output: 0.25000

Explanation: $2^{-2} = 1/2^{2} = 1/4 = 0.25$

Java Solution:-

```
class Solution {
public double myPow(double x, int n) {
if(n<0){
n=-n;
x=1/x;
}
double res=1.0;
while(n!=0)
{
if(n%2!=0)
{
res=res*x;
}
x=x*x;
n=n/2;
}
return res;

}
}
```