**############################################################**

**Smallest divisible number**

Given a number N, find an integer denoting the smallest number evenly divisible by each number from 1 to n.

**Example 1:**

**Input:**

N = 3

**Output:** 6

**Explanation:** 6 is the smallest number

divisible by 1,2,3.

**Example 2:**

**Input:**

N = 6

**Output:** 60

**Explanation:** 60 is the smallest number

divisible by 1,2,3,4,5,6.

**Your Task:**  
You dont need to read input or print anything. Complete the function **getSmallestDivNum()**which takes N as input parameter and returns the smallest number evenly divisible by each number from 1 to n.

**Expected Time Complexity:** O(N)  
**Expected Auxiliary Space:** O(1)

**Constraints:**  
1 ≤ N ≤ 25

long long gcd(long int a, long long b)

{

if(b==0)

return a;

return gcd(b,a%b);

}

long long getSmallestDivNum(long long n){

long long a=1;

for(long long i=2;i<n+1;i++)

a=(a\*i)/gcd(a,i);

return a;

}

**############################################################**

**Quadratic Equation Roots**

Given a quadratic equation in the form **ax2 + bx + c**. Find its roots.  
  
**Example 1:**

**Input:**

a = 1

b = -2

c = 1

**Output:** 1 1

**Explanation:**

Roots of equation x2-2x+1 are 1 and 1.

**Example 2:**

**Input:**

a = 1

b = -7

c = 12

**Output:** 4 3

**Explanation:** Roots of equation

x2 - 7x + 12 are 4 and 3.

**Your Task:**  
You don't need to read input or print anything. Your task is to complete the function**quadraticRoots()**which takes a, b, c as input parameters and returns a list of two integers containing the floor value of its roots in decreasing order. If roots are imaginary, the returning list should contain only 1 integer ie -1.   
**Note:**If roots are imaginary, the generated output will display "Imaginary".

**Expected Time Complexity:** O(1)  
**Expected Auxiliary Space**: O(1)

**Constraints:**  
-103 <= a,b,c <= 103

vector<int> quadraticRoots(int a, int b, int c) {

long long int mod=100000007;

int d;

int p,q;

d=(b\*b)-(4\*a\*c);

if(d<0)

return {-1};

else

{ p=(floor)((-1\*b-sqrt(d))/(2\*a));

q=(floor)((-1\*b+sqrt(d))/(2\*a));

if(q>p)

return {q,p};

else

return {p,q};

}

}

**############################################################**

**Digits In Factorial**

Given an integer **N**. Find the number of digits that appear in its factorial.   
Factorial is defined as, factorial(n) = 1\*2\*3\*4……..\*N and factorial(0) = 1.

**Example 1:**

**Input:** N = 5

**Output:** 3

**Explanation:** Factorial of 5 is 120.

Number of digits in 120 is 3 (1, 2, and 0)

**Example 2:**

**Input:** N = 120

**Output:** 199

**Explanation:** The number of digits in

120! is 199

**Your Task:**  
You don't need to read input or print anything. Your task is to complete the function**digitsInFactorial()**that takes **N** as input parameter and returns **number of digits**in factorial of **N**.

**Expected Time Complexity** : O(N)  
**Expected Auxilliary Space** : O(1)

**Constraints:**  
1 ≤ N ≤ 105

int digitsInFactorial(int n)

{

double d=0;

while(n>0)

{

d+=log10(n);

n--;

}

return d+1;

}

**############################################################**

**GP Term**

Given the first 2 terms **A and B**of a Geometric Series. The task is to find the**Nth term**of the series.

**Example 1:**

**Input:**

A = 2

B = 3

N = 1

**Output:** 2

**Explanation:** The first term is already

given in the input as 2

**Example 2:**

**Input:**

A = 1

B = 2

N = 2

**Output:** 2

**Explanation:** Common ratio = 2,

Hence second term is 2.

**Your Task:**  
You don't need to read input or print anything. Your task is to complete the function **termOfGP()**that takes **A, B and N** as parameterand returns***Nth***term of **GP**. The **return value**should be **double**that would be **automatically**converted to **floor**by the **driver code**.

**Expected Time Complexity** : O(logN)  
**Expected Auxilliary Space**: O(1)

**Constraints:**  
-100 <= A <= 100  
-100 <= B <= 100  
1 <= N <= 5

double termOfGP(int a,int b,int n)

{ if(n==1)

return a;

if(n==2)

return b;

double res=a;

while(n>1)

{

res=double(res\*b/a);

n--;

}

return res;

}

**############################################################**

**Primality Test**

A prime number is a number which is only**divisible by 1 and itself.**  
Given number **N** check if it is prime or not.

**Example 1:**

**Input:**

N = 5

**Output:** Yes

**Explanation:** 5 is only divisible by 1

and itself. So, 5 is a prime number.

**Example 2:**

**Input:**

N = 4

**Output:** No

**Explanation:** 4 is divisible by 2.

So, 4 is not a prime number.

**Your Task:**  
You don't need to read input or print anything. Your task is to complete the function **isPrime()**that takes **N** as input parameterand returns **True**if **N** is prime else returns F**alse**.

**Expected Time Complexity** : O(N1/2)  
**Expected Auxilliary Space**:  O(1)

**Constraints:**  
1 <= N <= 109

bool isPrime(int n)

{ if(n==1 || n==2)

return true;

int d=2;

bool flag=true;

while(d<n)

{ if(n%d==0)

{ flag=false;

break;

}

d++;

}

return flag;

}

**############################################################**

**Exactly 3 Divisors**

Given a positive integer value**N**. The task is to find how many numbers**less than or equal to N**have numbers of divisors exactly equal to **3**.

**Example 1:**

**Input:**

N = 6

**Output:** 1

**Explanation:** The only number with

3 divisor is 4.

**Example 2:**

**Input:**

N = 10

**Output:** 2

**Explanation:** 4 and 9 have 3 divisors.

**Your Task:**  
You don't need to read input or print anything. Your task is to complete the function**exactly3Divisors()**that takes **N**as input parameter and **returns**count of numbers**less than or equal to N**with exactly **3 divisors**.

**Expected Time Complexity** : O(N1/2\* N1/4)  
**Expected Auxilliary Space**:  O(1)

**Constraints :**  
1 <= N <= 109

int isprime(int n)

{ for(int i=2;i\*i<=n;i++)

if(n%i==0)

return 0;

return 1;

}

int exactly3Divisors(int n)

{ int count=0;

for(int i=2;i\*i<=n;i++)

if(isprime(i))

count++;

return count;

}

**############################################################**

**Modular Multiplicative Inverse**

Given two integers**‘a’**and**‘m**’. The task is to find the smallest modular multiplicative inverse of **‘a’** under modulo**‘m’**.

**Example 1:**

**Input:**

a = 3

m = 11

**Output:** 4

**Explanation:** Since (4\*3) mod 11 = 1, 4

is modulo inverse of 3. One might think,

15 also as a valid output as "(15\*3)

mod 11" is also 1, but 15 is not in

ring {0, 1, 2, ... 10}, so not valid.

**Example 2:**

**Input:**

a = 10

m = 17

**Output:** 12

**Explanation:** Since (12\*10) mod 17 = 1,

12 is the modulo inverse of 10.

**Your Task:**  
You don't need to read input or print anything. Your task is to complete the function**function modInverse()**that takes **a and m as input parameters**and **returns**modular multiplicative inverse of **‘a’** under modulo**‘m’**. If the modular multiplicative inverse doesn't exist **return -1.**

**Expected Time Complexity** : O(m)  
**Expected Auxilliary Space** : O(1)

**Constraints:**  
1 <= a <= 104  
1 <= m <= 104

int modInverse(int a, int m)

{ int i=0;

while(i<m)

{

if((i\*a)%m==1)

break;

i++;

}

if(i==m)

return -1;

return i;

}

**############################################################**