REVISION OF SIMPLE MATHEMATICS → Maths L-1

Finding number of digits in a number

Iterative Solution

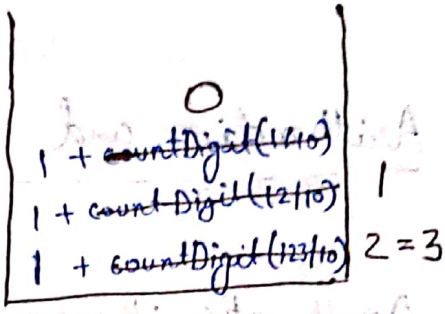
```
int countDigit(long n)
{
    int count = 0;
    while (n != 0) {
        n = n/10;
        ++ count;
    }
return count;
}
```

Explanation :- Let us suppose we have a input n is '123'. So firstly the value of count = 0 now n != 0 so n will go in the while loop and it get divided by 10 and we get 12 { 123/10 = 12 } and value of will be incremented by one. Now again as the value of n != 0 it will again go in the loop this time the value of n will be 1 { 12/10 = 1 } and again the

value of count will be incremented. This procedure will happen again and now the value of n is 0 and value of count will be incremented by 1. So now the value of n is 0 and value of count is 3. Since n=0 it will leave the while loop and the value of count will return by the function.

(2) Recursive Solution

```
int countDigit(long n)
{
    if (n==0)
        return 0;
    return 1 + countDigit(n/10);
}
```



Explanation:- In this recursive solution ~~finally~~ the program will call the function repeatedly and the fun calls will stored in stack as shown in the program. When the value of n = 0 then the function return value 0 and all the other function calls will return value.

③ Logarithmic Solution

```
int countDigit (long n)
{
    return floor (log10(n) + 1);
}
```

Explanation : Value of $\log_{10}(123)+1 = 2.089+1$ so the floor value will be 3.

Arithmetic and Geometric Progression

Arithmetic Progression: A sequence of number is said to be in an Arithmetic progression if the difference between any two consecutive terms is always the same.

Initial term: In an arithmetic progression, the first number in the series is called initial term. It is denoted by 'a'.

Common difference: The value by which consecutive term increases or decreases is called the common difference.

General formulas

- n^{th} term : $a+(n-1)d$.
- Arithmetic mean : $\frac{\text{Sum of all terms in the AP}}{\text{Number of terms in AP}}$
- Sum of 'n' terms : $0.5n(\text{first term} + \text{last term})$

$$= \frac{n}{2}\left(2a+(n-1)d\right)$$

Geometric Progression

A sequence of number is said to be in a Geometric progression. if the ratio of any two Consecutive terms is always same. In simple terms it means. that the next number is calculated by multiplying a fixed number to the previous number in series. for example 2, 4, 8, 8, 16 is a GP because ratio of any two consecutive number in the series is the same. $(4/2 = 8/4 = 16/8)$.

Initial term : In a geometeric progression, the first number is called initial term. It is denoted by a.

② Common ratio: The ratio of any two consecutive terms by taking the previous term in the denominator. It is denoted by 'r'.

General formula

If 'a' is the first term and 'r' is the common ratio

- n^{th} term of a GP: $a \times r^{n-1}$
- Geometeric mean: n^{th} root of the product of n terms in the GP.
- Sum of 'n' terms of a GP $(r<1) = \frac{[a(1-r^n)]}{[1-r]}$
- Sum of 'n' terms of a GP $(r>1) = \frac{[a(r^n-1)]}{[r-1]}$
- Sum of infinte terms of a GP $(r<1) = (a)/(1-r)$.

Quadratic Equations

~~ax^2+b~~

A quadratic equation is a second-order polynomial equation of a variable say x. The general form of a quadratic equation is given as

$$ax^2 + bx + c = 0$$

where a, b and c are real known values and a can't be zero.

Roots of an Equation : The roots of an equation are the values for which the equation satisfies the given condition. for example, the roots of equation $x^2 - 7x - 12 = 0$ are 3 and 4 respectively. If we replace the value of x by 3 and 4 individually in the equation, the equation will evaluate to zero.

A quadratic equation has two roots. The roots of a quadratic equation can be easily obtained by using the quadratic formula.

$$\text{roots} = \frac{\left(-b \pm \sqrt{(b^2 - 4ac)}\right)}{2a}$$

There arises three cases as described below while finding the roots of a quadratic equation.

If $b^2 < 4ac$, then roots are complex (not real).
for example, roots of $x^2 + x + 1 = 0$ are $-0.5 + i1.73205$ and $-0.5 - i1.73205$.

If $b^2 = 4ac$, then roots are real. and both roots are same
for example, roots of $x^2 - 2x + 1 = 0$ are 1 and 1

If $b^2 > 4ac$, then roots are real and different.
for example, roots of $x^2 - 7x - 12 = 0$ are 3 and 4.

Mean and Median

Mean is defined as the average of a given set data.
Let us consider the sequence of number 2, 4, 4, 5, 5, 7, 9
the mean of the given sequence is 5.

$$\frac{2 + 4 + 4 + 4 + 5 + 5 + 7 + 9}{8} = 5$$

formula for finding mean

$$\bar{x} = \frac{x_1 + x_2 + \ldots\ldots x_n}{n}$$

Where $x_1, x_2, x_3 \ldots x_n$ denotes the term of the given sequence and n is the count of the number present in the given sequence.

Median

Median is the value of a set of data. To determine the median value in a sequence of numbers, the numbers must first be arranged in an ascending order.

- If the count of the number in the sequence is odd, the median value is the number that is in the middle, with the same amount of numbers below and above.
- If the count of numbers in the sequence is even, then median is the average of the two middle values.

formula for finding Median

- If the count of numbers is odd: After sorting the sequence

  Median = $((N+1)/2)^{th}$ value.

  N is the number of term.

- If the count of the number is even: After sorting the sequence

  Median = Average of $(N/2)^{th}$ term and $((N/2)+1)^{th}$

Prime Numbers

A prime number is a whole number greater than 1; which is only divisble by 1 and itself.
first few prime numbers are: 2, 3, 5, 7, 11, 13 19, 23.

Some facts about prime Number

* Every prime number can be represented in the farm of $6n+1$ or $6n-1$ except 2 and 3, where n is a natural number.

* Two and three are only two consecutive ~~natu~~ natural numbers which are prime too.

LCM and HCF

Factors and Multiples: All numbers that divide a number completely i.e without ~~having~~ leaving any remainders are called factors of that number. for example, 24 is completely divisble by 1, 2, 3, 4, 6, 8 12, 24. Each of the number is called a factor of 24 and 24 is called a multiple of each of these numbers.

LCM: LCM stand for least common multiple. The lowest number that is exactly divisble by each of the given number is called the least common multiple of those numbers. for example

HCF: The term HCF stands for the highest common factor. The largest number that divide two or more number is the highest common factor for those number. HCF is also known as Greatest common divisor.

factorials: In mathematics, the factorial of a number say N is denoted by N!. The factorial of a number is calculated by multiplying all the integers between 1 & and N.

for example: $4! = 4 \times 3 \times 2 \times 1 = 24$

i.e $N! = N*(N-1)*(N-2)\ldots*2*1$

Note As per convention $0! = 1$

L-3

Palindrome Number

A number is said to be palindrome if the reverse of the number is same to the number.

For example: 78987 is a palindrome number because reverse of '78987' is '78987' which is same.

Ques Write a function which take input n and return true if the number is palindrome.

Code:

```
bool check_palindrome(int n)
{
    int x = n;
    int a;
    int b = 0;
    while(n != 0)
    {
        a = n % 10;
        b = b * 10 + a;
        n = n / 10;
    }
    return (x == b);
}
```

Time complexity = $\Theta(n)$

#Note A single digit number is always a palindrome number.

L-4 Factorial

Ques Write a function which take n as a input and return the factorial of n.

Code:

Iterative method

```
int factorial_by_iteration(long n){
    long x = 1;
    if (n == 0)
    {
      return 0;
    }
    else
     {
     for( int i = 1 ; i <= n ; i++ )
       {
          x *= i;
       }
     return x;
     }
}
```

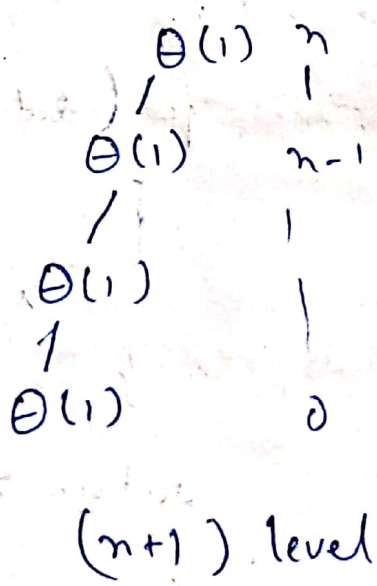
Iterative method has time complexity $\Theta(n)$, and auxiliary space $\Theta(1)$.

Recursive method

```
int factorial_by_recursion(long n)
{
    if(n==0)
    {
        return 1;
    }
    return n * factorial_by_recursion(n-1);
}
```

Time complexity of recursive function

$$T(n) = T(n-1) + \Theta(1)$$



Time complexity = $\Theta(n)$

Auxiliary space = $\Theta(n)$

Trailing Zero in factorial

Ques Find the number of trailing zero in the factorial of N.

I/p: n = 5 O/p: 1

I/p: n = 10 O/p: 2

I/p: n = 100 O/p: 24

Code:

```
int trailingZeroes (int N)
{
    int res = 0;
    while (N>=5) {
      N = N/5;
      res += N;
    }
    return res;
}
```

Explanation: To find trailing zero in a factorial we have to calculate number of '5' in a factorial. We have did same in the above code. We are finding number of '5' in factorial because trailing zero will come only when 5 is multiple

by 2 or some other even number. So as in a factorial we always find number of '2' more than number of '5' so we have to only calculate how many five are there.

Time complexity of the code is

$$5^k \leq n$$

$$k \leq \log_5 n$$

Time complexity $\theta(\log n)$.

Greatest common Divisor : L-06

GCD also called as HCF. is HCF of two numbers means the highest number which divide both of the digit is called GCD.

Ques Write a program which find the GCD of the two number?

I/p : a = 4 , b = 6
O/p : 2

I/p : a = 100 , b = 200
O/p : 100

Naive Solution

```
int gcd (int a, int b)
{
    int res = min(a,b);
    while (res > 0)
    {
      if ( a % res == 0 && b % res == 0 )
      {
        break
      }
      res--;
    }
    return res;
}
```

Time complexity : ~~O(n)~~ ~~O(min~~ $O(\min(a,b))$

This is the easiest method to ~~do~~ find GCD but it is not optmised. There is an another solution which can be done by using Euclidean Algorithm.

Euclidean Algorithm

Basic Idea:

Let 'b' be an smaller than 'a'

$$gcd(a, b) = gcd(a-b, b)$$

Proof:

Let 'g' be GCD of 'a' and 'b'

$$a = gx \;\; , \;\; b = gy \text{ and } GCD(x, y) = 1$$

$$(a - b) = g(x - y)$$

Code Implementation by using Basic GCD Idea

```
int gcd (int a, int b)
{
  while (a != b)
  {
    if (a > b)
      a = a - b;
    else
      b = b - a
  }
  return a;
}
```

a = 12 , b = 15
a = 12 , b = 3
a = 9 , b = 3
a = 6 , b = 3
a = 3 , b = 3

Optmiso version of Euclid Algorithm

```
int gcd (int a, int b)
{
    if (b==0)
        return a;
    else
    return gcd(b, a%b);
}
```

Explanation:

gcd (12, 15)

↳ gcd (15, 12)

↳ gcd (12, 3)

↳ gcd (3, 0)

LCM of two numbers L-07

Ques Write a program to find LCM of two numbers

I/p : a = 4 , b = 6

O/p : 12

I/p : a = 12 , b = 15

O/p : 60

I/p : a = 3 , b = 7

O/p : 21

I/p : a = 2 , b = 8

O/p : 8

Naive Solution

```
int lcm(int a, int b)
{
    int res = max(a,b);
while (true)
{   if (res % a == 0 & res % b == 0)
        return res;
    res++;
}
return res;
}
```

Time complexity : $O(a * b - max(a,b))$

Efficient Solution

```
int gcd ( int a, int b)
{  if (b == 0)
       return a;
   return gcd ( b, a % b);
}
int lcm ( int a , int b)
{
    return (a * b) / gcd (a, b);
}
```

Time complexity : $O(\log(\min(a,b)))$

Check for Prime Number

Write a program to find whether the number is prime or not. If the number is prime if return true else if returns false.

Naive solution:

```
bool isPrime(int n)
{
    if (n == 1) return false;
    for (int i = 2 , i <= n ; i++)
        if (n % i == 0) {
            return false; }
    return true;
}
```

Time complexity: O(n).

Efficent method

Idea: Divisors always appear in pairs.

If (x, y) is a pair

$$x * y = n$$

and if $x \leq y$

$$x * x \leq n$$

$$x \leq \sqrt{n}$$

Explanation: If we write all divisors of a number we can notice that these divisors always appear in pairs. for example

30: (1,30), (2,15), (3,10), (5,6)

65: (1,65), (5,13)

25: (1,25), (5,5)

So, the idea is not check the divisor untill n, instead of that we will check untill $\sqrt{n}$. It is because, let us say if we have a divisor y which is greater than $\sqrt{n}$, then the divisor y will always have a

pair x ,which is always less than $\sqrt{n}$.

Code :-

```
bool isPrime(int n) {

  if(n==1)
   return false;

 for(int i=2; i*i<=n; i++)
    if(n%i==0)
       return false;

  return true;
  }
```

Explanation: Let us run this code for n=65

n = 65

i = 2

i = 3

i = 4

i = 5 Here at i=5 the if condition

become true and it returns false. Now let's check for $n = 37$

i.e $i = 2$, $i = 3$, $i = 4$, $i = 5$, $i = 6$

as $\sqrt{37} \approx 6$ so the loop will untill $i = 6$ and we don't get any number which divide so it will return true.

One more efficent idea

We can decrease the number of iteration by using some more extra checks or corner cases. As we know 2 is the only even number which is also a prime number. Now, the idea is we add a check for $n \% 2 == 0$ and $n \% 3 == 0$ so we can save many iterations. for large value of n.

Code :-

```
bool isPrime(int n)
{
    if (n == 1) return false;
    if (n == 2 || n == 3) return true;
    if (n % 2 == 0 || n % 3 == 0)
        return false;
    for (int i = 5; i * i <= n; i = i + 6)
        if (n % i == 0 || n % (i + 2) == 0)
            return false;
    return true;
}
```

Explanation : This is the most optimized code we have. Due to the corner case $n \% 2 == 0$ and $n \% 3 == 0$, many of the iteration is removed. Now ~~for int i~~ our loop will start from $i = 5$ and it will check for 5 and 7 as all the other number before 7 is checked by

by the corner case of 2 and 3. Now we will check for i=11 and i=13 in next iteration. This will reduce the number of iteration.

Prime factors L-09

Write a program which take an input n and return all the prime factor of n.

I/p : n = 12
O/p : 2 2 3

I/p : n = 13
O/p : 13

I/p : n = 315
O/p : 3 3 5 7

Naive sol'n

```
void primefactor(int n)
{
  for(int i=2; i<n; i++)
  {
    if(isPrime(i))
    {
      int x = i;
      while(n%x==0)
      { print(i); x = x*i; }}}
```

Explanation: Here we will check whether the i is prime or not then we will go in while loop and print (i) if $(n \% x == 0)$. Here we have declare x because if the n is divisble by x two times then it will print i two times.

Time complexity - $O(n^2 \log n)$

Idea for efficent solution

① Divisors always appear in pairs.

30: (1,30), (2,15), (3,10), (5,6)

so, let (x,y) be a pair

$$x * y = n$$

If $x \leq y$

$$x * x \leq n$$

$$x \leq \sqrt{n}$$

From this we can conclude that if we go from 2 to $\sqrt{n}$ then there will be a prime

divisor.

(2) A number n can be written as multiplication of power of prime factors.

$12 = 2^2 * 3$

$450 = 2^1 * 3^2 * 5^2$

Code :-

```
void primefactor(int n)
{
    if (n<=1) return ;
  for(int i=2 ; i*i <=n ; i++)
   {
      while (n% i == 0)
        {
          print(i);
           n = n/i;
        }
   }
  if (n>1)
    print(n);
}
```

& More efficent solution

```
void printprimefactor (int n)
{
    if (n <=1) return ;
    while (n %, 2 == 0)
    { print(2); n = n /2 ; }
    while (n %, 3 == 0)
    { print(3); n = n /3 ; }
for (int i=5; i*i <= n ; i = i+6 ;)
    {
        while (n %, i == 0)
        { print(i); n = n /i ; }
        while (n %, (i+2) == 0 )
        { print(i+2); n = n /(i+2); }
    }
    if (n>3)
    print(n);
}
```

All divisors of a Number L-10

~~I/p: n=15~~

Write a program which find all the divisor of n and print it in sorted order.

I/p: n = 15

O/p: 1 3 5 15

I/p: n = 100

O/p: 2, 4, 5, 10, 20, 25 50, 100

Naive solution :-

```
Void printDivisor(int n)
{
    for(int i=1; i<=n; i++)
        if(n % i == 0)
            print(i);
}
```

Time complexity: $\Theta(n)$

Auxiliary space: $\Theta(1)$

Efficent solution idea

1) Divisors always appear in pairs. 30: (1,30), (2,15) (3,10), (5,6)

2) One of the divisor in every pair is smaller than or equal to $\sqrt{n}$.

for a pair (x, y)

$x \times y = n$

let x be the smaller, i.e $x \leq y$

$x \times x \leq n$

$\Rightarrow \boxed{x \leq \sqrt{n}}$

Code:-

```
void printDivisor (int n)
{
  for (int i=1; i*i<=n; i++)
    if (n % i == 0)
    {
      print (i);
      if (i != n/i)
        print(n/i);
    }
}
```

This code will print all the divisor but not in sorted order.

for

Code in sorted order:

```
void printDivisors (int n)
{
    int i;
    for (i=1 ; i * i < n ; i++)
        if (n % i == 0)
            print(i);
    for ( ; i >= 1 ; i--)
        if (n % i == 0)
            print (n/i)
```

Time complexity: $\theta(\sqrt{n})$

Auxiliary space: $\theta(1)$.

Sieve of Eratosthenes

Ques Write a program which take an input n and returns all the prime number less than n and return n also if it is a prime number.

I/p : n = 10
O/p : 2 3 5 7

I/p : n = 23
O/p : 2 3 5 7 11 13 17 19 23

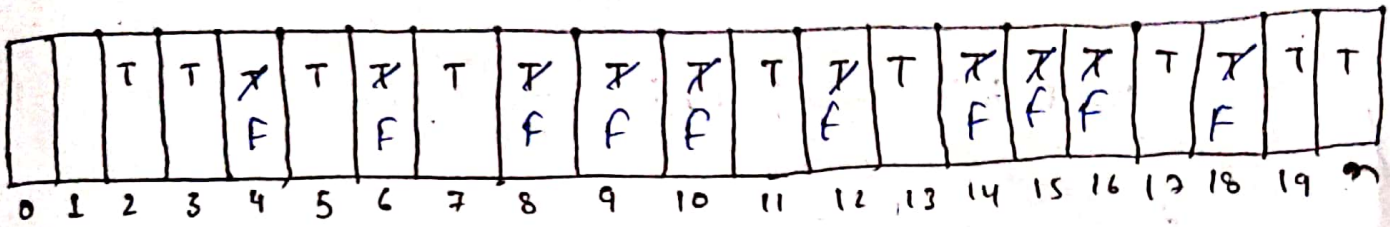
Naive Solution :

```
void printprime (int n)
{
  for (int i = 2; i <= n; i++)
  {
    if (isPrime (i))
      print(i);
  }
```

Time complexity : $O(n * \sqrt{n})$

Efficent Solution

We will use sieve of eratosthenes algorithm and we will create a boolean array of size [n+1].



Initial values in isPrime[n+1]

Multiples of 2, 3 and 5

2: 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22....

3: 6, 9, 12, 15, 18, 21, 24....

5: 10, 15, 20, 25, 30....

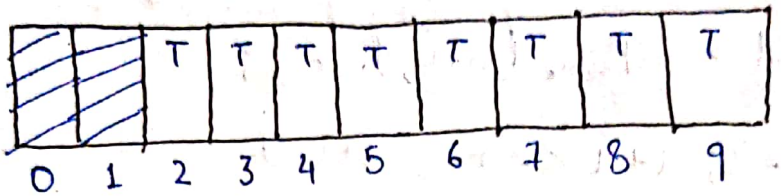
Now we will ~~make~~ mark multiple of 2, 3 and 5 as false in boolean array.

Code :-

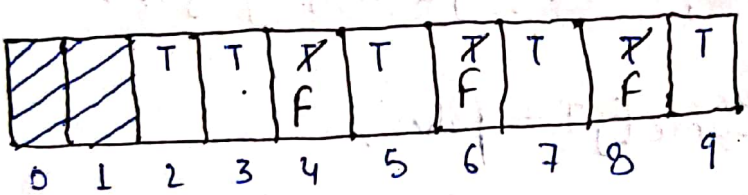
```
void sieve (int n)
{
    vector<bool> isPrime (n+1, true);
    for (int i = 2; i*i <= n; i++)
    {
        if (isPrime[i])
        {
            for (int j = 2*i; j<= n; j = j+1)
            {
                isPrime[j] = false;
            }
        }
    }
    for (int i = 2; i <= n; i++)
    {
        if (isPrime[i])
        cout << i << " ";
    }
```

Explanation :

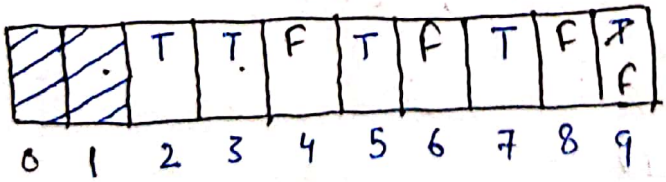
Firstly it will make a boolean array of n+1 length.



Now, for i = 2 we print 2 and mark 4, 6 and 8 as false.



Now, for i = 3 we print 3 and mark 6 and 9 as false



Now, we can see that leaving the prime number all the other number become false in the array and now we will print the number whose value is true.

Optimised Solution

Code:-

```
void sieve (int n)
{
  vector<bool> isPrime (n+1, true);
 for (int i = 2; i*i <= n ; i++)
 {
    if (isPrime[i])
    {
      for (int j = i*i ; j <= n ; j = j+i)
        { isPrime[j] = false ;
        }
    }
for (int i = 2 ; i <= n , i++)
  {
    if (isPrime [i])
      cout << i << " ";
    }
  }
```

Computing Power

L-12

Write a function which takes an input x and n. And writes it will return the pow(x, n).

I/p : x = 2, n = 3

O/p : 8

I/p : x = 5, n = 0

O/p : 1

Naive solution :

```
int power (int x, int n)
{
    int res = 1;
    for (int i = 0; i < n; i++)
        res = res * x;
    return res;
}
```

Time complexity : $\theta(n)$.

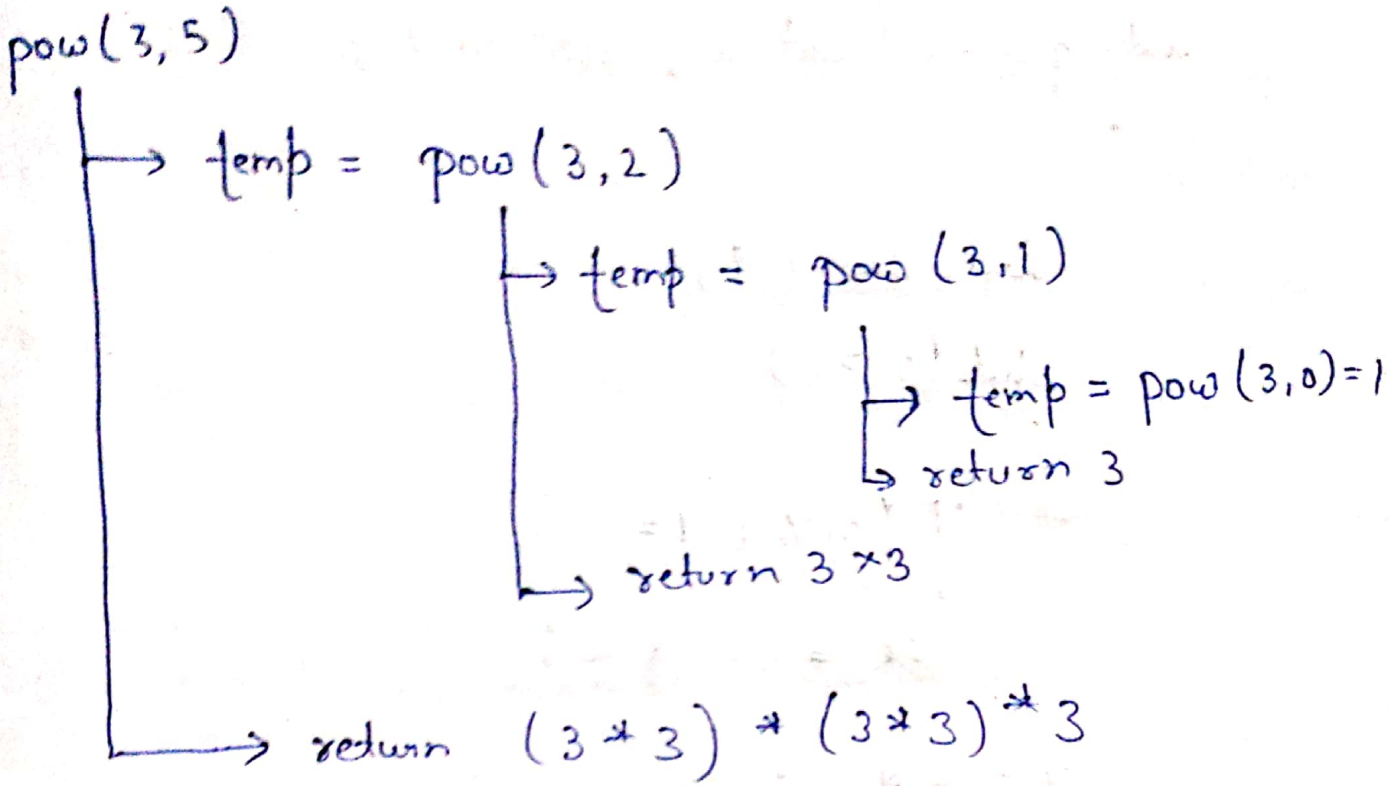
Efficent Solution

$$power(x,n) = \begin{cases} \text{if } n\%2 == 0 \\ power(x, n/2) * power(x, n/2) \\ \text{else} \\ power(x, n-1) * x \end{cases}$$

Code :-

```
int power (int x , int n)
{
    if (n == 0)
        return 1;
int temp = power ( x, n/2);
 temp = temp * temp
if ( n%2 == 0 )
   return temp ;
else
   return temp * x ;
```

Explanation:



Iterative Soln for power of number

L-13

* Every number can be written as sum of powers of 2 (sets bits in ~~power~~ binary representation).
* We can tranverse through all bits of a number. from (LSB to MSB) in O(log n) time.

Code:-

```
int power (int x, int n)
{
    int res = 1;
    while (n>0)
    {
        if (n%2 != 0)
            res = res * x;
        x = x * x;
        n = n/2;
    }
    return res;
}
```

Explanation:

$x = 4$, $n = 5$

Initially : $res = 1$

Ist iteration : $res = 4$

$x = 16$

$n = 2$

IInd iteration : res = 4

x = 16 * 16 = 256

n = 1

IIIrd iteration : res = 4 * 256 = 1024

x = 256 * 256 = 65536

n = 0

Time complexity : O(log n)

Aux space : O(1)

03 — Bit Magic

Bitwise Operator in C++ L-01

The bitwise algorithms are used to perform operations at bit-level or to manipulate bits in different ways. The bitwise operations are found to be much faster and are some times used to improve the efficency of a program.