Mathematics

Page No.
Date.

find no. of digits in number

1) non-Recursive approach :-

```
int getdigit (int n)
{   int count = 0;
    while (n > 0)          → O(n)
    {  n /= 10;
       count++;
    }
    return count;
}
```

2) ~~Non~~ recursive approach :-

```
int getdigit (long long n)  → O(n)
{
    if (n <= 0) return 0;

    return 1 + getdigit (n/=10);
}
```

3) most efficient method :-
Logarithmic method :- → O(1)

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

Arithmetic and geometric progression.

AP: $t_n = a + (n-1)d$ → t_n → nth term

Sum = $S_n = \frac{n}{2}[2a + (n-1)d]$

GP: $t_n = ar^{n-1}$

$S_n = a\left[\frac{1-r^n}{1-r}\right]$

Quadratic Equation:

$ax^2 + bx + c = 0$

If α and β are roots of this q. equation.

$\alpha + \beta = -b/a \quad \alpha \cdot \beta = c/a$

α or $\beta = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$

$D = \sqrt{b^2 - 4ac}$

$D < 0$ → Imaginary roots

$D = 0$ → Real and equal roots

$D > 0$ → Real and unequal roots

mean is usually average

median → the middle number of sorted array. If the number of terms are even then take input as mean of middle two digits.

prime numbers

2, 3, 5, 7, ...

⇒ Every prime numbers can be represented in form of $6n+1$ and $6n-1$ except 2 and 3.

Factors:- All the numbers which divides a number completely then these numbers are factors of number.

~~not~~ comp. least common multiple

GCD - Greatest ~~com~~ common divisor.

Factorial of $n = n \times (n-1) \times (n-2) \times \cdots \times 1$.

$0! = 1$

Permutation :- Arrangement of r things that can be done out total n things

$${}^{n}P_{r} = \frac{n!}{(n-r)!}$$

Combination :- Selection of r things out of total n things

$${}^{n}C_{r} = \frac{n!}{(n-r)!\,r!}$$

Page No.
Date.

modulo arithmetic:

representation → a o/o b

gives remainder

→ to prevent integer overflows.

→ The remainder obtained after the division ops ~~of~~ of two operands are called modulo operations.

→ operator for doing modulo is %

long long can handle integer from 2^{-32} to $2^{32}-1$.

Scanned with CamScanner

① # count digits in number

I/P: x = 9235 $x > 0$
O/P: 4
I/P: x = 38
O/P: 2

I) Iterative approach

```
int count (int x)
{ int res = 0;
  while(x>0)
  { x /= 10;
    res++;
  }
  return res;
```

TC → O(n).

II) Recursive approach

```
int count (int x)
{
  if (x <= 0) return 0;
  return 1 + count(x/=10);
```

TC → O(n)

III) return floor(log10(n)) + 1;

TC → O(1)

Palindrome Number

I/P: n = 78987
O/P: yes
I/P: n = 8668
O/P: yes
I/P: n = 8
O/P: yes

I/P: n = 21
O/P: no.
I/P: 8678
O/P: No.

n ≥ 0.

Palindrome → actual number = reverse number → 3443

code:

```
bool MPal(int n)
{ int rev = 0;
  int temp = n;
  while( temp <= 0)
  { rev = temp % 10;
    temp /= 10;
    rev *= 10;
  }
  rev /= 10;
  if (rev == n) return true;
  else return false;
}
```

③ # Factorial of numbers

Hm-

① Recursive [approach] =

```
int rec ( int n)
{ int res = 1;
  for( i = n; i >= 1; i--)
  { res *= i;
  }
  return res;
}
```

I/P: 2 O/P: 2

I/P: 3 O/P: 6

② Recursive approach

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

TC → Tn = T(n-1) + O(1) → O(n)

SP → O(n).

Iterative is considered better

④ count trailing zeroes ~~and~~ of factorial n.

I/P: n=5
O/P: 1
I/P: n=10
O/P: 2

I/P: n=100
O/P: 24
~~I/P:~~

Idea → count no. of 5 in factors

~~Code~~:

Naive approach:
get factorial and count zeroes

efficient approach:
count no. of 5's in factors of n.

$1 \times 2 \times 3 \times 4 \times \dots \times n$

$\lfloor \rfloor$ → floor function

Trailing zeroes = $\left\lfloor \frac{n}{5} \right\rfloor + \left\lfloor \frac{n}{5^2} \right\rfloor + \left\lfloor \frac{n}{5^3} \right\rfloor + \dots$ ①

Code:

```
int getTrailingZeroes(int n)
{
    int count=0, num=5;
    while (num <= n)
    {
        count = count + floor(n/num);
        num *= 5;
    }
    return count;
}
```

Scanned with CamScanner

or

```
int gettrail (ll n)
{
    int res=0
    for( int i=5 ; i<=n ; i*=5)
    {
        res += n/i ;
    }
    return res;
}
```

* Time complexity →

5^k iterations

$$5^k \leq n$$
$$k \leq \log n / \log 5$$
$$k \leq \log_5 n$$

$\therefore O(\log n)$.

(5) GCD (Euclidean algorithm)

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

⑥ Lcm of two numbers

Idea => $a \times b = \text{Lcm}(a,b) \times \text{gcd}(a,b)$

Find gcd using euclidean algo.
and then
find Lcm.

⑦ Check if no. is prime or not.

* Naive approach:

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

→ worst case complexity → O(n)

* Optimised approach:

Idea: divisor always appear in pairs

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

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

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

and we can find factors of n till $\sqrt{n}$ only.

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

TC → $O(\sqrt{n})$

* more efficient approach:-
Idea:- By checking n%2==0 or n%3==0 and n%5==0, try this extra checks to save many operations.

```
bool isPrime(LL n)
{  if(n==1) return false; if(n==2 or n==3) retu
   if(n%2==0 or n%3==0 or n%5==
      return false;

   for(int i=5; i*i<=n; i=i+6)
      if(n%i==0 or n%(i+2)==0)
         return false;

   return true;
}
```

Imp →

Scanned with CamScanner

Date.

⑧ print prime factors

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

I/P: n=315
O/P: 3 3 5 7.

I/P: n=13
O/P: 13

* Naive approach:

Code:-

```
bool isPrime(ll n);

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

Important →

for isprime. inner while

TC → O(n) * O(n) * O(log n) → O(n² log n)

* efficient approach:

Idea:

① Divisors always appear in pairs

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

Let (x,y) be a pair

x * y = n

if (x ≤ y) x*x ≤ n

x ≤ √n.

Date.

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 printPrimefactor (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;
}
```

Important →

* More efficient approach:

```
void printPrimefactor (ll 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+=6)
{
    while(n%i==0)
    { print(i); n=n/i; }
    while(n%(i+2)==0)
    { print(i+2); n/=(i+2); }
}
if(n>3)
{ print(n);
}
```

TC → $O(\sqrt{n})$.

(9) Print ALL divisors of a number

* Naive approach:

Code:

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

TC → O(n) $\Omega(n)$

$\Omega(1)$ auxillary space

* efficient approach:

Idea:

1) Divisors of numbers always appear in pair

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

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

For a pair (x,y)

$x * y = n$

Let x be the smaller i.e. $x \leq y$.

$x * x \leq n$

$x \leq \sqrt{n}$

Code:

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

A solution that print all divisors but not in order

n = 25

i=1; → print(1)

→ print(25)

i=2: ~~print(2)~~

i=3:

i=4:

i=5; print(5);

A solution that prints the divisors in order

Code:

```
void divisors (ll n)
{   int i;
    for (int i=1; i*i<=n; i++)
    {  if (n%i==0)
          cout<<i<<" ";
    }
    i--; // Important or else the
         // last i will repeat
         // itself in further code.

for (int i=
For (  ; i>=1; i--)
{
    if (n%i==0)
    cout<<(n/i)<<" ";
}
```

Page No.
Date.

code →

Sieve of ~~Eratosthenes~~ : Eratosthenes :

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

to find all the numbers (prime) smaller than or equal to.

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

Naive approach :

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

isprime

TC → O(n) * O(√n) ⇒ O(n√n)

Sieves of eratosthenes algorithm :

Create of boolean array from 2 to n of size n+1

	T	T	T	T	T	T	T	T	T	T	T	...	T
0	1	2	3	4	5	6	7	8	9	10	11		n

Initial values of isPrime[n+1]

Now,

→ Mark all the multiples of 2, 3, 5 as false exclusive 2,3,5

		T	T	F	T	F	T	F	F	F	T	F	...	
0	1	2	3	4	5	6	7	8	9	10	11	12		n

code →

and later on

* Simple implementation of sieve eratosthenes

doubt *

```
void sieve (ll 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+=i)
        isPrime[j] = false;
    }
  }
  for(int i=2; i<=n; i++)
  {
    if(isPrime[i])
      cout<<i<<" ";
  }
}
```

[multiples of i] ←

→ Replace

→ for(int i=2; i*i<=n; i++)

Note:- {for(int j=2*i; j<=n; j=j+i)

}

this is for multiples of i.

Suppose n = 16

i=2 → j = 4, 6, ~~8~~ 8, 10, 12, 14, 16

i=3 → j = 6, 9, 12, 15, ~~18~~, ...

i=4 → j = ~~8~~, 12, 16.

For values of vector which are already marked then we will not even pass "it".

Doubt

Code →

Page No.
Date.

* Improved version of sieve of erathosthenes
→ Replace i in second loop with i*i

Code :-

```
void sieve (ll 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)
      }
   }
   for( int i=2; i<=n; i++)
   {
      if (isPrime[i])
            cout <<i <<" ";
   }
}
```

Replacing →

→ optimised →
Implementation
~~i~~ i^2, i^2+i, i^2+2i, ...

Earlier → already marked because of 2, 3
5 : 10, 15, 20, 25, 30, ...
Now →
5 : 25, 30

composite numbers smaller than i
i*(i-1)
i*(i-2)

Page No.
Date.

code →

* shortes Implementation of sieve eratosthen algorithm:

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

→ O(nloglogn) → TC

#define ll long long.

Page No.
Date.

Code →

Computation of power (x, n) → x^n.

$n \geq 0$

I/P: x=2, n=3
O/P: 8

I/P: x=3, n=4
O/P: 81

I/P: x=5 n=0
O/P: 1

I/P: x=5, n=1
O/P: 5

Inbuild fn: pow(x,n)

function: 1) Naive approach :-

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

TC → O(n).

2) Efficient approach :-

~~code~~ (x,n)

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

code:-
```
int power (ll x, ll n)
{
    if (n==0)
        return 1;
```

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

→ pow(3,5)

↳ temp = pow(3,2)

↳ temp = pow(3,1)

↳ temp = pow(3,0) = 1

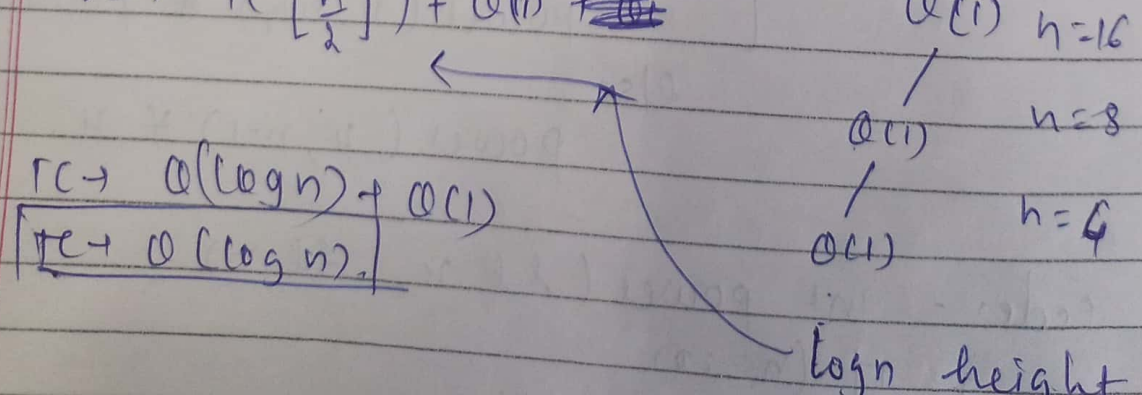
↳ return 3

↳ return 3*3;

→ return (3*3)*(3*3)*3

temp * x → on n%2==0

TC → T([n/2]) + O(1)



TC → O(logn) + O(1)

TC → O(logn)

Need to Revise

code →

iterative power : (Binary exponention).

$3^{10} = 3^8 \times 3^2$	10 : 1010
$3^{19} = 3^{16} \times 3^2 \times 3^1$	19 : 10011

* Every number can be written as sum of power of 2 (set bits in binary representation)
* we can teaverse through the bits of a number (from LSB to MSB) in O(log n) time

```
while(n>0)
{ if(n%2!=0)
        // Bit is 1

  else  // Bit is 0

  n=n/2;
}
```

10 : 1010
$3^8 3^1 3^2 3^1$

code :-

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

    x=x*x;
    n/=2;
  }
return res;
}
```

$n >> 1 = n/2$
$n >> x = n/2^x$

we handle large numbers using modulo.

Page No.
Date.

$x = 4, \quad n = 5$

Initially : res = 1

1st iteration : res = 4
$x = 16$
$n = 2$

IInd iteration : res = 4
$x = 16 * 16 = 256$
$n = 1$

IIIrd iteration : res = $256 \times 4 = 1024$
$x = 256 \times 256 = 65536$
$n = 0$.

Time complexity : O(log n)
Auxillary space : O(1).

5 : 0 1 0 1
$4^5 4^3 4^2 4^1$

res = $4^4 \times 4^1$

we can further optimise it using bitwise operator.