L5- Basics Maths for DSA

Digit Concept

N = 7789

extraction of digits → means you need all the digits individually.

7789 % 10 = 9

/10 → 778 % 10 = 8

/10 → 77 % 10 = 7

/10 → 7 % 10 = 7

$\frac{7789}{10} = 778.9$

Pseudo Code

7789 ✓ dry run

```
while (N > 0)
{
    lastdigit = N % 10;
    N = N/10;
}
```

7789 % 10 → 9

7789 / 10 → 778

Q1 Count digits

N = 7789

O/P = 4

```
cnt = 0
while (N > 0)
{
    last digit = N % 10;
    cnt = cnt + 1;
    N = N / 10;
}
cout << cnt << ;
```

T.C ⇒ $O(\log_{10}(N))$

Q2 Reverse a number

N = 1234 N = 10300

O/P = 4321 O/P = 301

7789 ⟶ 9877

```
revN = 0
while (N > 0)
{
    lastdigit = N % 10;
    N = N/10;
    revN = (revN x 10) + lastdigit
}
```

Dry Run

N = 7789 % 10 = 9

/10 → 778 % 10 = 8

/10 → 77 % 10 = 7

/10 → 7 % 10 = 7

revN = 0

revN = (revN x 10) + lastdigit

1st iteration (7789)

revN = (0 x 10) + 9 = 9

2nd iteration (778) = last digit = 8

778 > 0

778 % 10 = 8 revN = (9 x 10) + 8

778/10 = 77 revN = 98

3rd iteration $(98 \times 10) + 7 = 987$

4th iteration $(987 \times 10) + 7 = 9877$

Code

```
int main()
{
    int n;
    cin >> n;
    int revN = 0
    while (N > 0) {
    int lastdigit = n % 10;
    ~~n/10~~
    revN = (revN x 10) + lastdigit
      n = n/10
    }
    Cout << revNumber
}
```

Palindrome

n = 1 3 3 1

o/p = true

n = 1 2 3

o/p = false

→ It's a reverse of a number

Steps

- Reverse a number
- while doing N/10, at the end it will be '0' so it's important to store duplicate while extracting
- So now we will compare.

```
int main()
{
    int n;
    cin >> n;
    revN = 0
    duplicate = n
    while (n > 0) {
        int ld = n % 10;
        revN = (revN * 10) + ld
    }   n = n/10;
```

```
if (dup == revN)
    true
else false.
```

Armstrong Number

N = 371

$\Rightarrow 3^3 + 7^3 + 1^3 = 371$

true

N = 35

$\Rightarrow 3^2 + 5^2$

$= 34$ false

N = 1634

$\Rightarrow 1^4 + 6^4 + 3^4 + 4^4 = 1634$

true

→ Extract the number

→ sum = sum + pow (lastDigit, cntDigits)

↳ no of digit)

Approach

n = 153

sum = 0

no of digit = 3 (so we need to cube every digit)

1st iteration, extract digit '3', and cube it ⇒ 27, add it to the sum ⇒ 0 + 27 = 27

2nd iteration, extract digit '5' and cube it ⇒ 125, add it to sum ⇒ 27 + 125 = 152

3rd iteration, E.D = 1, cube = 1, S = 152 + 1 = 153

now compare original number == Sum of digit

```
bool armstrongNumber (int n)
{
    int original_no = n
    int count = 0
    int temp = n

    // to count no of digit
    while (temp != 0)
    { count++;
      temp = temp/10;
    }
    int sumofpower = 0
    while (n != 0)  // calculate sum of nth power of each
                       digit
    {
      int digit = n % 10;
      sumofpower += pow(digit, count)
      n /= 10;
    }
    return (sumofpower == original no);
}
```

Print all Divisors of a number

→ [1 to N]

N = 36

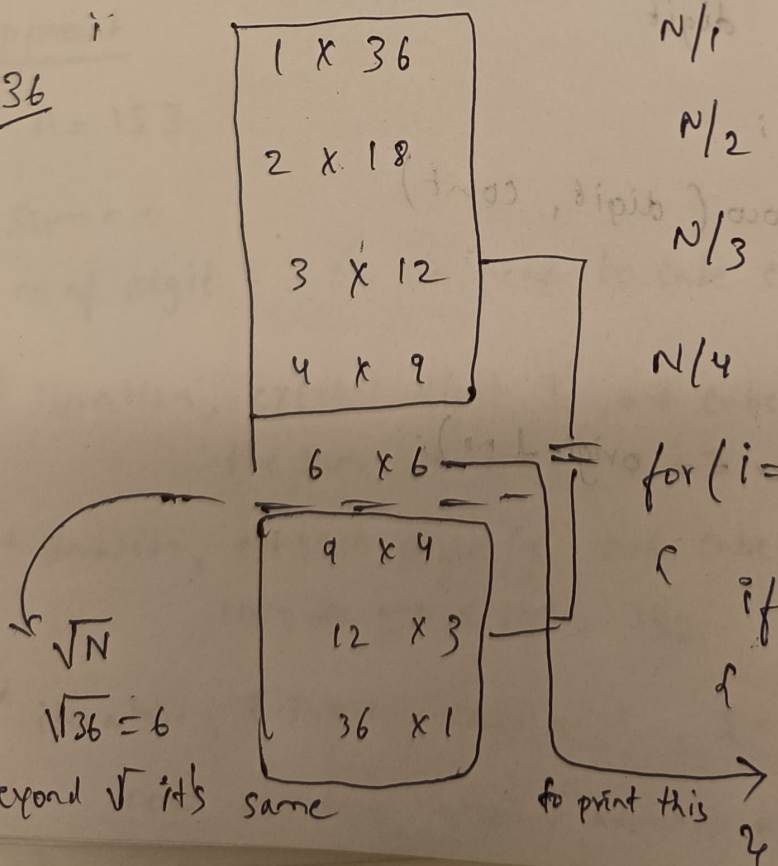
o/p = 1, 2, 3, 4, 6, 9, 12, 18, 36.

→ loop from 1 to N

```
for(i=1; i<=N; i++)
{
    if(N%i==0)
        print(i)
}
```

T.C ⇒ O(N)

2nd Approach



T.C = O(√N)

```
for(i=1; i<=sqrt(N); i++)
{
    if(n%i==0)
    {
        print(i)
        if((n/i)!=i)
            print(n/i)
    }
}
```

```
void printDivisor (int n){
    vector<int> ls;
    for(int i=1; i<=sqrt(n); i++)
    { if(n%i==0)
        { ls.push_back(i);
          if((n/i)!=i)
          { ls.push_back(n/i);
          }
        }
    }
    sort(ls.begin(), ls.end());
    //to print
    for(auto it:ls) cout<<it<<" ";
}
```

Prime Numbers

→ A number that has exactly '2' factors.

N = 11 ⇒ 1, 11 Prime

N = 13 ⇒ 1, 13 "

N = 4 ⇒ 1, 2, 4 Not Prime

Brute force Approach

```
cnt = 0
for (i = 1; i <= n; i++)
{
    if (N % i == 0)
        cnt++
}
if (cnt == 2) Prime
else (Not Prime)
```

T.C ⇒ O(N)

Optimized Approach

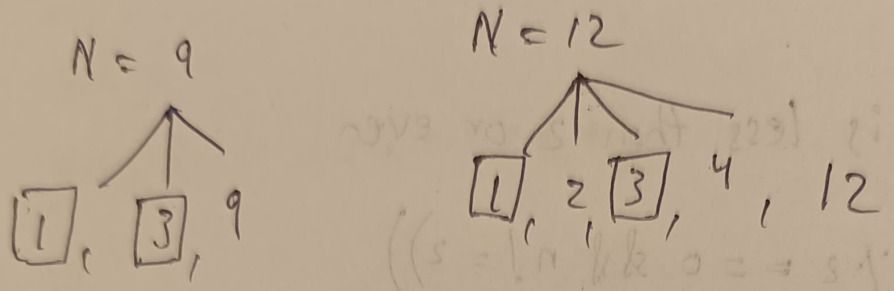
```
bool isPrime (int n) {
    for (int i = 2; i*i < n; i++)
    {   if (N % i == 0)
            return false
    }
    return true;
}
int main ()
{   bool ans = isPrime (n)
    if (n != 1 && ans == true)
        cout << "Prime No" else cout << "Non-Prime";
```

Better Code -

```
bool isPrime(int n)
{
// Check if the num is less than 2 or even
if (n < 2 || (n % 2 == 0 && n != 2))
    return false

// Check if the num is divisible by any odd nos up to its
square root
    int root = sqrt(n);
    for(int i = 3; i <= root; i += 2)
    {
        if (n % i == 0)
            return false;
    }
// If the nos is has not been found to be divisible
by any smaller odd nos, then its prime
return true;
}
int main()
{ int n; cin >> n
// Calling the function to check if the given number is prime or not
if (isPrime(n)) { cout << "true" << endl.
    else cout << "false" << endl.
```

GCD/HCF



out of this highest common factor (HCF) = 3

$$gcd(9, 12) = 3$$

for every two nos there is always one gcd which is '1'.

$$\Rightarrow gcd(20, 40) \rightarrow 20$$

Brute force

N1 = 9 N2 = 12

for → 1 to 12 i ✓ 3 ✓

gcd = 1

→ min(n1, n2) → e.g. (20, 40) (40, 20)

```
for (i = 1; i <= n1; i++)
    if (n1 % i == 0 && n2 % i == 0)
        gcd = i;
}
return gcd
```

T.C ⇒ O(min(n1, n2))

Euclidean Algorithm

N1, N2

$gcd(n1, n2) = gcd(n1 - n2, n2)$ where $n1 \geq n2$

in book

$gcd(a, b) = gcd(a - b, b)$ $\quad a \geq b$

e.g. 15, 20

keep on truncating

$gcd(20, 15) = gcd(5, 15)$

$\downarrow$

5

$gcd(15, 5) = gcd(10, 5)$

$gcd = (5, 5)$

$gcd(0, 5)$

The moment one number is '0' the other number is the gcd.

This is equalivent to dividing. & the final,

$$\boxed{gcd(a, b) = gcd(b, a \,\%\, b)} \qquad \boxed{a \geq b}$$

```
while (a>0 && b>0)
{
   if (a>b)      a = a % b;
   else          b = b % a;
}
if (a==0) print (b)
else      print (a)
```

for the swap for the big one nos to divide

// if one becomes zero the other nos is gcd.

T.C ⇒ whenever division is happening log will come

$$O(\log_{\phi}(\min(a,b)))$$