Basic maths for DSA

Date
Page

(1) Digit Counts

N = 7789

7 7 8 9

extraction of digits

		(int)
7789 % 10 =	9	7789 / 10 = 778
778 % 10 =	8	778 / 10 = 77
77 % 10 =	7	77 / 10 = 7
7 % 10 =	7	7 / 10 = 0

number n : 4567 → It can be anything

count the number of digits in n which evenly divides n.

Logic: extract digit → store it into temp
check that n is divisable by that digit or not
(but first check that digit != 0)
take count variable and increment it
return that variable

question

output

- 4567 → 0
- 123 → 2
- 84 → 1
- 77770 → 4
- 12040 → 3

```
int evenlyDivides(int N)
{   int count=0, number;
    number = N
    while(number != 0)
    {   int digit = number % 10;
        number = number / 10;
        if(digit != 0 && N % digit == 0)
        {count++;}
    }
return count;
}
```

Extra learning | No. of Digits count

* Brute force approach

(ex.) 7789 / 10 = 9
778 / 10 = 8
77 / 10 = 7
7 / 10 = 7

int count = 0;
count++; 1
count++; 2
count++; 3
count++; 4
return count → (4)

number of digits

---

* Optimal approach

logarithmic property:

if you take the log of any number with base 10 and round it up, you will get the number of digits in that number
so, number of digits

$\log_{10} 1 = 0$
$\log_{10} 10 = 1$
$\log_{10} 100 = 2$
... x

$\log_{10} N + 1$

```
int function (int n)
{
    int count = (int) (log10(n) + 1);
    return count;
}
```

typecasting

(2) Reverse a number

7789 → 9877

n

revnum = 0

while (n > 0)

{ int digit = n % 10;

revnum = (revnum * 10) + digit

n = n / 10;

}

cout << revnum;

① 7789 → 9 → 9 (0×10+9)

② 778 → 8 → 98 (9×10+8)

③ 77 → 7 → 987 (98×10+7)

④ 7 → 7 → 9877 (987×10+7)

Question

* Given a signed 32 bit integer x, return x with digit reversed.

if reverse of x causes value outside signed 32 bit range (integer) $(-2^{31}, 2^{31}-1)$

return 0;

MIN = -2147483648

MAX = 2147483647

rev = 0;

while (x != 0)

{ int digit = x % 10

if (

(rev > MAX/10 || (rev == MAX/10 && digit > 7))

(rev < MAX/10 || (rev == MIN/10 && digit < -8))

)

{ return 0; }

rev = (rev * 10) + digit;

```
    x = x/10
}
return rev;
```

(3) Number is ~~palin~~ palindrome or not.

→ check (num == revnum)
if yes true
is no false

---

(4) Number is armstrong or not.

→ armstrong means

ex. $371 = 3^3 + 7^3 + 1^3 = 27 + 343 + 1 = (371)$
sum of digit's cube (digit^3) = num itself
but
$1634 = 1^4 + 6^4 + 3^4 + 4^4 = (1634)$
that means

'armstrong number' => sum of (digit)^digitcount == number itself

optimal approach

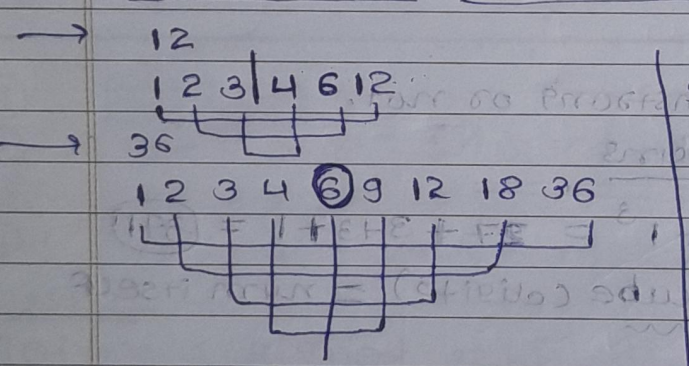
```
temp = n
sum = 0
digitcount = log10(n) + 1
while (temp != 0)
{  digit = temp % 10;
   sum = sum + pow(digit, digitcount);
   temp = temp/10;
}
return (sum == n) ; true / false
```

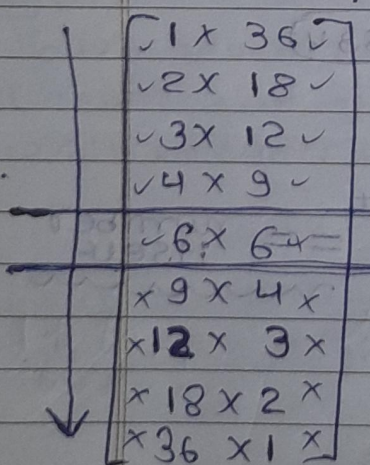
(5) Print all divisors

Brutforce

```
take num
loop from 1 to num
    check num%i == 0
        print i
end loop.
```

mathematical observation





sorted, unique

```
set<int> factors;
int num;
cin >> num;
for(int i=1; i<=sqrt(num); i++)
{ if (num%i==0)
  { int temp;
    temp = num/i;
    factors.insert(i);
    factors.insert(temp);
  }
}
display
↓
foreach loop
```

numbers are ≤ square root of n.

Mistake

with sorting ↑

You should have use vector instead of sets.
Both approach have the same time complexity $O(\sqrt{n} \cdot \log n)$

However vector with sorting is faster then using sets.

because set take logarithmic time for each insertion.

* but if you don't want to use sorting function then you can take 2 vectors which will decrease time complexity to $O(\sqrt{n})$

Optimal Solution

$n = 36$

```
vector<int> small, large;
int num;
cin >> num;
for(int i=1; i<=sqrt(num); i++)
{ if (num % i == 0)
  { small.push_back(i);
    if (i != num/i);
    { large.push_back(num/i);}
  }
}
for (int i: small)
  { display i}
for (int i = large.size()-1; i>=0; i--)
  { display i }
```

max 6 — $O(\sqrt{n})$

max 6 — $O(\sqrt{n})$

max 6 — $O(\sqrt{n})$

$O(\sqrt{n})$

Geeks for Geeks
sum of all divisors (બધાનો sum)

(6) Check Prime | num that is divisible by 1 & itself ✗

definition: numbers that has 2 factors (1 & Number itself) ✓

Brute force approach : O(n)

Optimal approach : O(√n)

```
if (num == 1)
    not prime return;
for (i = 2 to sqrt(num))
    if (num % i == 0)
        count++;
check count == 0
    → It is prime
else → It is not prime.
```

O(√n)

---

(7) GCD & LCM (HCF)

GCD

(9,12)

factors: 1, 3, 9 ; 1, 2, 3, 4, 6, 12

1, 3

GCD → 3

Brute force

```
int gcd = 1
int length = a>b ? b : a
for (1 to length)
    if (a%i == 0 && b%i == 0)
        gcd = i;
return gcd;
```

Bruteforce — $O(\min(a,b))$

Better — $O(\min(a,b))$ → faster

same TC but faster

```
int min = a>b ? b : a;
for (int i = min; i >= 1; i--)
    if (a%i==0 && b%i==0) {
        return i;
    }
return 1;
```

Optimal — Euclidean (2 तरीके से हल)

Algo.

> if a > b
> then, gcd(a,b) = gcd(a−b, b)

gcd(52,10) ⇒ gcd(42,10)
⇒ gcd(32,10)
⇒ gcd(22,10)
⇒ gcd(12,10)
⇒ gcd(2,10) ⇒ gcd(10,2)
⇒ gcd(8,2)
⇒ gcd(6,2)
⇒ gcd(4,2)
⇒ gcd(2,2)
⇒ gcd(0,2)

ans (2)

gcd(a%b, b) ≡

logic

```
while (a>0 && b>0)
    a>b ? a=a%b : b=b%a;
return (a==0) ? b : a;
```

gcd(a,b);

$O(\log(\min(a,b)))$