Recall Division Algorithm. Let a be an integer and d a positive integer. Then there are unique integers q and r, with $0 \le r < d$, such that $a = dq + r$.

<u>Definition</u> In the above, $q = a \text{ div } d$, $r = a \bmod d$.

Remark) In Python3, $q = a \,//\, d$, $r = a \,\%\, d$.

Note also that $q = \lfloor \frac{a}{d} \rfloor$, $r = a - d\lfloor \frac{a}{d} \rfloor$

Ex. $101 \bmod 11 = 2$ as $101 = 11 \cdot 9 + 2$

So, $101 \text{ div } 11 = 9$

Modular Arithmetic on $\mathbb{Z}/m\mathbb{Z}$ ($\mathbb{Z}_m$) for positive integer m.

We can classify integers into classes represented by their remainder mod m.

$$\mathbb{Z}_m = \{0, 1, \cdots, m-1\}$$

$$a +_m b = (a+b) \bmod m$$

$$a \cdot_m b = (a \cdot b) \bmod m$$

Ex. $7 +_{11} 9 = 5$, $7 \cdot_{11} 9 = 8$

Properties. 1. Closure: If $a, b \in \mathbb{Z}_m$, then $a +_m b \in \mathbb{Z}_m$ and $a \cdot_m b \in \mathbb{Z}_m$

2. Associativity: If $a, b, c \in \mathbb{Z}_m$, then $(a +_m b) +_m c = a +_m (b +_m c)$ and $(a \cdot_m b) \cdot_m c = a \cdot_m (b \cdot_m c)$

3. Commutativity: If $a, b \in \mathbb{Z}_m$, then $a +_m b = b +_m a$ and $a \cdot_m b = b \cdot_m a$.

4. Identity elements: The elements 0 and 1 are identity elements for addition and multiplication. That is,

$$a +_m 0 = 0 +_m a = a \quad \text{and} \quad a \cdot_m 1 = 1 \cdot_m a = a$$

for any $a \in \mathbb{Z}_m$

5. Additive inverses. If $a \in \mathbb{Z}_m - \{0\}$, then $m - a$ is an additive inverse of a in $\mathbb{Z}_m$. That is,

$$a +_m (m-a) = 0 \quad \text{and} \quad 0 +_m 0 = 0$$

6. Distributive Laws. If $a, b, c \in \mathbb{Z}_m$, then

$$a \cdot_m (b +_m c) = a \cdot_m b + a \cdot_m c$$

$$(a +_m b) \cdot_m c = a \cdot_m c + b \cdot_m c$$

Base b expansion of n.

Let b be an integer > 1. If n is a positive integer then $n = a_k b^k + a_{k-1} b^{k-1} + \cdots + a_1 b + a_0$ where k is a nonnegative integer, $a_0, \ldots, a_k$ are nonnegative integer less than b.

In Python3,

```
def base_repr(n, p):
    ls = []
    while n:
        ls.append(n % p)
        n //= p
    return ls    # returns base-b digits in an array reversed.
```

Decimal Expansions (most common)

$$375 = 3\cdot 10^2 + 7\cdot 10 + 5$$

Binary Expansions (used by computers)

$$\begin{aligned}
375 &= 2\cdot 187 + 1 \\
&= 2(2\cdot 93 + 1) + 1 \\
&= 2(2(2\cdot 46 + 1) + 1) + 1 \\
&= 2(2(2\cdot(2\cdot \underline{23} + 0) + 1) + 1) + 1 \\
&= 2(2(2(2\cdot(\underline{2\cdot 11 + 1}) + 0) + 1) + 1) + 1 \\
&= 2(2(2(2(2\cdot(2\cdot 5 + 1) + 1) + 0) + 1) + 1) + 1 \\
&= 2(2(2(2(2(2(2\cdot 2 + 1) + 1) + 1) + 0) + 1) + 1) + 1 \\
&= 2(2(2(2(2(2(2\cdot(\underline{2\cdot 1 + 0}) + 1) + 1) + 1) + 0 + 1) + 1) + 1 \\
&= (101110111)_2
\end{aligned}$$

Instead, keeping the digit one by one will be simpler in the expression. For example,

$375 = 2\cdot 187 + 1$	_ _ _ _ _ _ _ _ 1
$187 = 2\cdot 93 + 1$	_ _ _ _ _ _ _ 1 1
$93 = 2\cdot 46 + 1$	_ _ _ _ _ _ 1 1 1
$46 = 2\cdot 23 + 0$	_ _ _ _ _ 0 1 1 1
$23 = 2\cdot 11 + 1$	_ _ _ _ 1 0 1 1 1
$\vdots$	$\vdots$

Finally 1 0 1 1 1 0 1 1 1

The expansions often get longer for binary. To shorten the expansion, hexadecimal expansion is also used.

Ex. $(11\ 1110\ 1011\ 1100)_2$ into hexadecimal

value in decimal 11 → B in hexadecimal

value in decimal is 12 → C in hexadecimal

value in decimal 14 → E in hexadecimal

value in decimal 3 → 3 in hexadecimal

Recall Hexadecimal system $0, 1, 2, 3, 4, 5, 6, 7, 8, 9 \to$ use as they are

$10, 11, 12, 13, 14, 15 \to A, B, C, D, E, F$

The number in hexadecimal expansion is $(3EBC)_{16}$

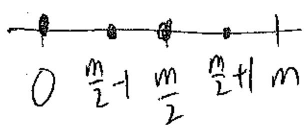
Modular Exponentation (Left-to-right binary method)

Problem) Find $y^n \bmod m$ for given y, n, and $m > 0$.

① $n=2$. This is modular squaring. Note that

$y^2 \bmod m$ can be calculated by

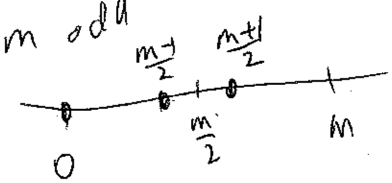
m even



$r^2 \bmod m$ where $-\frac{m}{2} \leq r < \frac{m}{2}$ and

$y \bmod m = r \bmod m$.

m odd



This reduces the size after squaring.

② $n \geq 3$. The method is useful for large n.

Step1: Write n in binary expansion, collect digits in an array $[b_0, b_1, \cdots, b_{\ell-1}]$ of length ℓ where ℓ is the length of the expansion.

Step2:

```
s = 1
for i = 0 to l-1
    if b_i :  r = sy % m     # if b_i = 1
    else :  r = s            # if b_i = 0
    s = r^2 % m
return r
```

Ex. $n = (1011)_2$

$i=0$: $b_0 = 1$
$$\begin{cases} s = 1 \\ r = sy \% m = y \% m \\ s = r^2 \% m = y^2 \% m \end{cases}$$

$i=1$: $b_1 = 0$
$$\begin{cases} s = y^2 \% m = y^{(10)_2} \% m \\ r = s \\ s = r^2 \% m = y^{(100)_2} \% m \end{cases}$$

$i=2$: $b_2 = 1$
$$\begin{cases} s = y^{(100)_2} \% m \\ r = sy \% m = y^{(101)_2} \% m \\ s = r^2 \% m = y^{(1010)_2} \% m \end{cases}$$

$i=3$: $b_3 = 1$
$$\begin{cases} s = y^{(1010)_2} \% m \\ r = sy \% m = y^{(1011)_2} \% m \quad \text{(last bit, so it returns } r\text{)} \\ s = r^2 \% m = y^{(10110)_2} \% m \quad \text{(unnecessary, can be skipped)} \end{cases}$$

③ Running time analysis

Each i gives at most 2 multiplications.

Thus, the running time is O(number of bits of n)

This is $O(\log_2 n)$, or $O(\log n)$

$\uparrow$ base 2 logarithm (for $\log_2$) $\qquad$ $\uparrow$ natural logarithm (for $\log$)

(recall $\log_2 n = \dfrac{\log n}{\log 2}$)