§12. Computational Complexity.

General question: How long will it take a computer to perform some computations?

§12.1 Elementary bit operations.

Computers store numbers in a binary (base 2) form:

$$n = (b_{k-1} b_{k-2} \dots b_1 b_0)_2 \text{ where each } b_i \in \{0,1\}$$

they are called <u>bits</u>.

In other words,

$$n = b_{k-1} \cdot 2^{k-1} + b_{k-2} \cdot 2^{k-2} + \dots + b_1 \cdot 2^1 + b_0 \cdot 2^0 .$$

Examples: $26 = (11010)_2$, or $26 = 2^4 + 2^3 + 2$

$11 = (1011)_2$ or $11 = 2^3 + 2^1 + 2^0$.

~~To store $n \in \mathbb{Z}^+$ we need k bits where k is~~

The numbers consisting of k bits are those between

$$(10\dots0)_2 = 2^{k-1} \quad \text{and}$$

$$(11\dots1)_2 = 2^{k-1} + 2^{k-2} + \dots + 2 + 1 = 2^k - 1 .$$

Number of bits required for $n \in \mathbb{Z}^+$ is the unique $k \in \mathbb{Z}^+$ such that

$2^{k-1} \le n < 2^k$

Or in other words, $k = \lfloor \log_2 n \rfloor + 1$.

the biggest integer $\le \log_2 n$.

k grows much slower than n.

Example: $2^{10} = 1024$, so 1000 has 10 bits
10^6 has 20 bits
10^{12} has 40 bits.

Assume m has k bits, n has l bits, $k \ge l$.
Then $m+n$ has either k or $k+1$ bits

$$2^{k-1} < 2^{k-1} + 2^{l-1} \le m+n < 2^k + 2^l \le 2^{k+1}$$

$m \cdot n$ has either $k+l-1$ or $k+l$ bits

$$2^{k+l-2} = 2^{k-1} \cdot 2^{l-1} \le m \cdot n < 2^k \cdot 2^l = 2^{k+l}.$$

Q: How many operations do we need to compute $m+n$, $m \cdot n$?

Compute $26+11$ in binary form.

$$\begin{array}{r} 11010 \\ +\ 1011 \\ \hline 100101 \end{array} \quad \left(\begin{array}{l} 0+0=0 \\ 1+0=0+1=1 \\ 1+1=0 \text{ and } 1 \text{ to carry} \end{array} \right)$$

In general we have k elementary bit operations (addition of bits, with or without a carry, and taking 1 to carry).

Compute 26·11 in binary.

$$\begin{array}{r} 11010 \\ \times\ 1011 \\ \hline 11010 \\ +\ 11010 \\ \hline 1001110 \\ +\ 11010 \\ \hline 100011110 \end{array}$$

Such a method is called "long multiplication".
It requires $\leq l$ copies of the first number and then we add them up.
That needs $\leq l-1$ additions, each involving k bit operations
In total we need $\leq k(l-1)$ bit operations.
If both numbers have k digits, that requires $< k^2$ bit operations.
If we multiply two $2k$-digit numbers then the number of bit operations grows 4 times.

Now multiply 2 numbers by an alternative method (by Karatsuba, 1960):
Let $m = (a_{2k-1}, a_{2k-2} \dots a_1 a_0)_2$
$n = (b_{2k-1}, b_{2k-2} \dots b_1 b_0)_2$

We firstly divide m, n in half:

$$m = 2^k m_1 + m_0$$
$$n = 2^k n_1 + n_0 \quad \text{where}$$

$$m_1 = (a_{2k-1} a_{2k-2} \cdots a_k)_2, \quad m_0 = (a_{k-1} a_{k-2} \cdots a_0)_2$$
$$n_1 = (b_{2k-1} \cdots b_k)_2, \quad n_0 = (b_{k-1} \cdots b_0)_2$$

Then
$$\begin{aligned} m \cdot n &= (2^k m_1 + m_0)(2^k n_1 + n_0) \\ &= 2^{2k} m_1 n_1 + 2^k m_1 n_0 + 2^k m_0 n_1 + m_0 n_0 \\ &= 2^{2k} m_1 n_1 + 2^k m_1 n_1 + 2^k m_0 n_0 + m_0 n_0 \\ &\quad + 2^k (m_1 - m_0)(n_0 - n_1) \end{aligned}$$

Assume that multiplication of two k-digit numbers takes $M(k)$ bit operations. Then computing $m \cdot n$ requires

$$3 \cdot M(k) + 2k + 4 \cdot 2k$$

for $m_1 n_1$, $m_0 n_0$ and $(m_1 - m_0) \cdot (n_0 - n_1)$

to compute $m_1 - m_0$ and $n_0 - n_1$

for four final additions.

$$= 3 \cdot M(k) + 10k.$$