Recall: $f(k)$ is $O(g(k))$ if $\exists\, C, N > 0$ such that

$$f(k) \leq C g(k) \text{ for all } k \geq N.$$

Properties:

(1) If $f_1(k)$ and $f_2(k)$ are $O(g(k))$ then so is $f_1(k) + f_2(k)$

Proof: $f_1(k) \leq C_1 g(k)$ for $k \geq N_1$,

$f_2(k) \leq C_2 g(k)$ for $k \geq N_2$

$\Rightarrow f_1(k) + f_2(k) \leq (C_1 + C_2) g(k)$ for $k \geq \max\{N_1, N_2\}$ $\boxtimes$

(2) If $f_1(k)$ is $O(g_1(k))$ and $f_2(k)$ is $O(g_2(k))$ then $f_1(k) f_2(k)$ is $O(g_1(k) g_2(k))$

Proof: Ex.

(3) If $f(k)$ is $O(g(k))$ and $g(k)$ is $O(h(k))$ then $f(k)$ is $O(h(k))$ — Ex.

Relation with limits:

Proposition: (a) If $\frac{f(k)}{g(k)} \to L$ as $k \to \infty$ then $f(k)$ is $O(g(k))$

(b) If $\frac{f(k)}{g(k)} \to \infty$ as $k \to \infty$ then $f(k)$ is <u>not</u> $O(g(k))$

Remark: it is possible that $\frac{f(k)}{g(k)}$ does not tend to anything as $k \to \infty$.

Proof: 1(a) $\frac{f(k)}{g(k)} \to L$ as $k \to \infty$.

This means that for any $\varepsilon > 0$ there exists some $N = N(\varepsilon)$ such that

$$L - \varepsilon \leq \frac{f(k)}{g(k)} \leq L + \varepsilon \text{ for any } k \geq N$$

Take $\varepsilon = 1$, $C = L+1$ and $N = N(1)$. Then

$$f(k) \leq (L+1)\, g(k) \text{ for any } k \geq N(1). \boxtimes$$

(b) — Ex.

Remark: The "big O" notation does not give us any lower bounds for f in terms of g.

Example: Compute $n!$ where ~~k~~ $n \leq 2^k$.

Naive approach: start with 1 and multiply it by $2, 3, 4, \dots, n$.

It requires $n-1$ multiplication, which is $O(2^k)$

Multiplication ~~requires~~ involves:

$\leq k$ bits number

$\leq nk$ bits number

By long multiplication it requires

$O(nk^2)$ bit operations which is $O(2^k \cdot k^2)$.
In total the algorithm requires
$$O(2^k \cdot k^2 \cdot 2^k) = O(2^{2k} \cdot k^2) \text{ bit operations.}$$
Not polynomial time.

§13. Comptutational complexity of some known algorithms.

§13.1 Division with remainder:

Given a, b at most k bits long. Want to find q, r such that
$$a = q \cdot b + r, \quad 0 \le r < b.$$

Long division:
$$a = 26 = (11010)_2$$
$$b = 11 = (1011)_2$$

```
         1 0  ← quotient.
1011 | 1 1 0 1 0
       1 0 1 1
     ----------
       0 0 1 0 0  ← remainder
```

In general algorithm requires $\le k^2$ bit operations plus $O(k)$ comparisons. In total there are $O(k^2)$ bit operations. Algorithm is polynomial time.

§ 13.2. Computing GCD.

Given a, b, both at most k bits long ($a, b \leq 2^k$). Want to compute $\gcd(a, b)$.

Naive approach: try ~~small~~ values d from 1 to $\min\{a, b\}$ and test, whether it divides both a and b.

Number of checks is $\min\{a, b\}$ which can be as large as 2^k — not polynomial time.

Euclidean algoritm:

$$a = q_1 b + r_1$$

$$b = q_2 r_1 + r_2$$

$$r_1 = q_3 r_2 + r_3$$

$$\ldots$$

$$r_{n-3} = q_{n-1} r_{n-2} + r_{n-1} \neq 0 \leftarrow \gcd(a, b)$$

$$r_{n-2} = q_n r_{n-1} + r_n = 0$$

Each step of the algorithm is $O(k^2)$ bit operations. So in total we have $O(n k^2)$ bit operations.

Proposition: For each i, we have $r_{i+2} \leq \frac{1}{2} r_i$.

Proof. Case 1: $r_{i+1} \leq \frac{1}{2} r_i$. Then $r_{i+2} < r_{i+1} \leq \frac{1}{2} r_i$ ✓

Case 2: $r_{i+1} > \frac{1}{2} r_i$.

$$r_i = q_{i+2} r_{i+1} + r_{i+2} \Rightarrow r_{i+2} = r_i - q_{i+2} r_{i+1}$$

$$\leq r_i - r_{i+1} < r_i - \frac{1}{2} r_i = \frac{1}{2} r_i$$

$\boxtimes$