What are big numbers? Some examples...

- $\aleph_0, \aleph_1, \aleph_2, \ldots$ are very big numbers. ?.
- The largest known prime, discoverd by Slowinski and Gage, 1994, is $2^{859433} - 1$.
- Indlekofer and Jarai computed 1995 the largest known twin primes $242206083 \cdot 2^{38880} \pm 1$.
- RSA-130 = 1807082088687404805951656164405905566278102516769401349170127021450056662540244048387341127590812303371781887966563182013214880557 was factored by Lenstra et al. on April 10, 1996.
- Fermigier found out on May 19, 1996, that the elliptic curve $y^2 + xy + y = x^3 - 940299517776391362903023121165864x + 10707363070719743033425295515449274534651125011362$/ Q has rank ≥ 22.
- Pott used the number 5 in her Ph.D. thesis in functional analysis.
- The largest value of a probability measure is 1.

How to avoid using big numbers?

- Floating point arithmetic: i.e. Let $N \approx \pm m B^e$, where B is fixed (e.g. $B = 2, 10, 2^{32}$), e small, and $m \in \{0, \ldots, k\}$. This leads to the well known problems of numerical stability, error growth, non associativity...
- Modular arithmetic: Let p be a fixed (prime) number and perform your calculations modulo p. E.g. In algebraic geometry many invarinats remain stable under reduction modulo suited primes.

  Macaulay 3.0 performed all integer calculations modulo $p = 31991$.
- The Chinese remainder approach: Let $n = d_1 \cdots d_r$ with $\gcd(d_i, d_j) = 1$ for $i \neq j$. Do the calculations parallel on r machines modulo d_i and put the result together, using the Chinese remainder theorem:

  $$\mathrm{Z}/n\mathrm{Z} \cong \mathrm{Z}/d_1\mathrm{Z} \times \cdots \times \mathrm{Z}/d_r\mathrm{Z}.$$

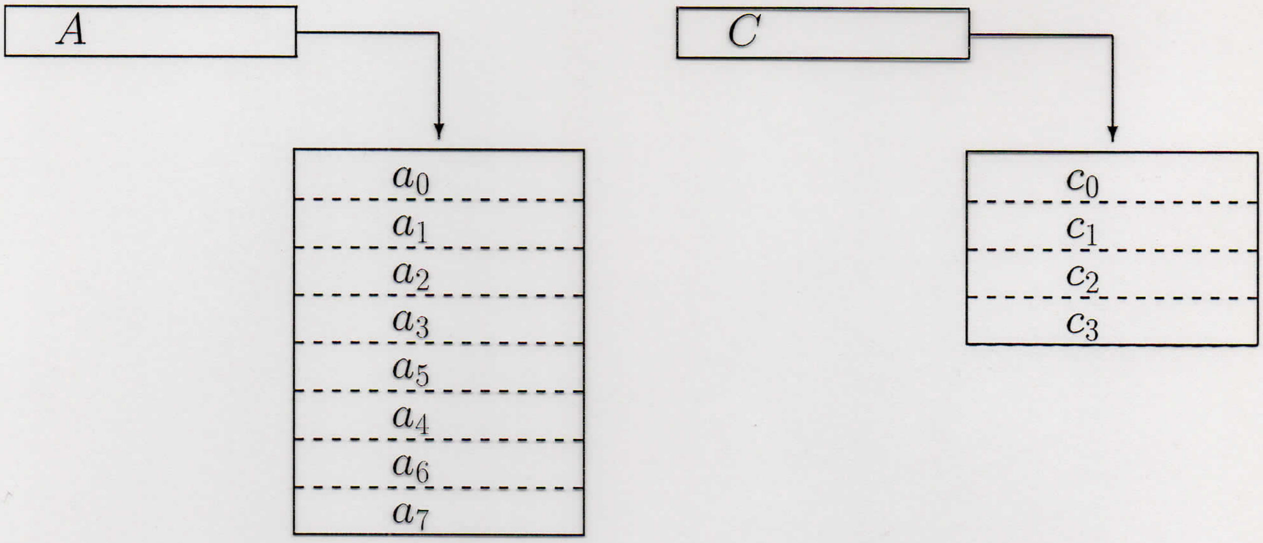
  If n is chosen big enough, the comuptation of $+$, $-$ and $\cdot$ is like the computation in Z. This method is sometimes used to evaluate polynomials over Z.

i.a.

$B = 2^{32}$

Representation by arrays:

Let $A = a_0 + a_1 B + \cdots + a_7 B^7$
and $C = c_0 + c_1 B + c_2 B^2 + c_3 B^3$.



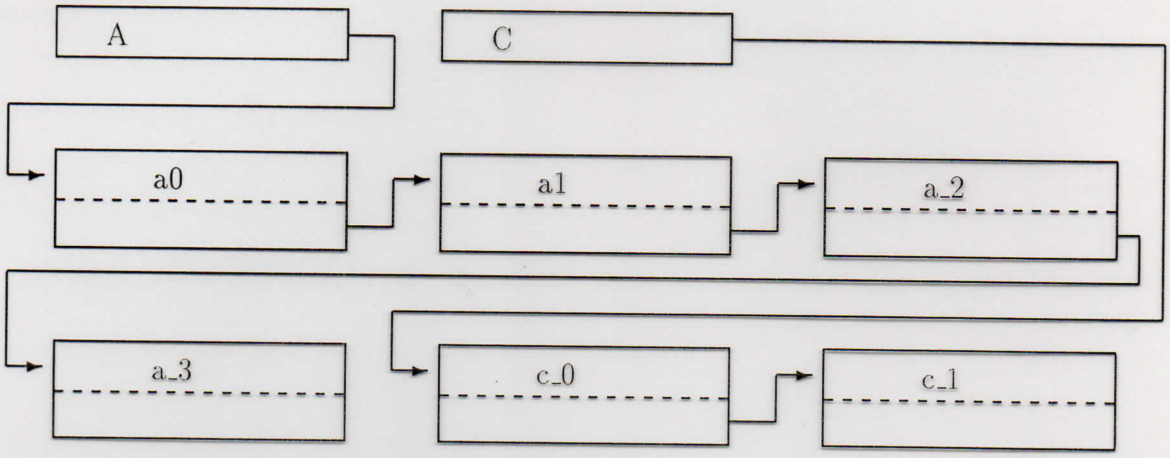
This leads to the problem of fragmentation, because the memory of your computer develops like this:

$C \leftarrow A \cdot C$ leads to:

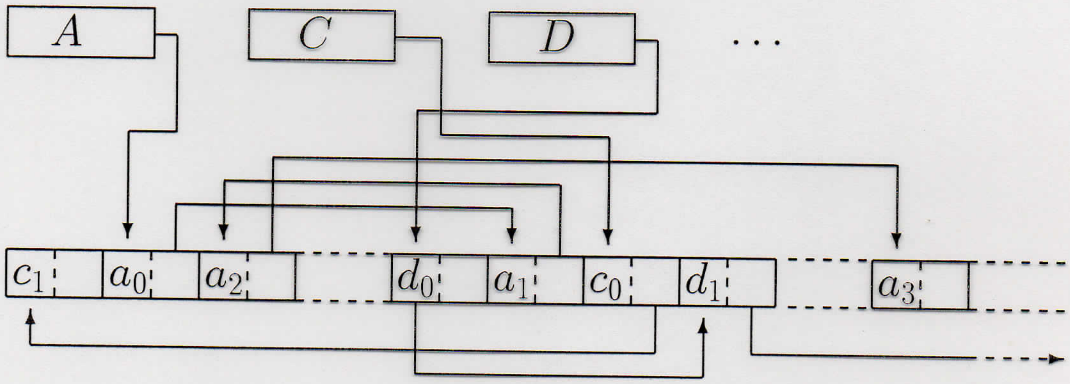
some computations later ...

Representation by lists:

Let $A = a_0 + a_1 B + a_2 B^2 + a_3 B^3$
and $C = c_0 + c_1 B$.



This avoids the fragmentation problem completely, but in the memory of your computer, there can happen someting like this:



It can happen, that you have to access many different parts of the memory, in order to walk through a number.

Lists vs. Arrays

Arrays: Fast in performing basic integer operations $(+, -, \cdot, /)$, esp. when the integers are large. E.g. factoring algorithms, primality testing...

Lists: Fast, if the integers occur as parts of more complicated objects, esp. when the integers themself are small $(< B^5)$. E.g. computations with matrices over rationals, elements of algebraic fields, points of elliptic curves...

SIMATH uses three different packages, to perform integer calculations:

- Arrays of fixed length (by R. Staszewski, Essen 1991)
- Arrays of arbitrary length (by R. Dentzer, Heidelberg 1993)
- Lists (by the SIMATH group itself, Saarbrücken, since 1986)

For more infos about SIMATH see the URL http://emmy.math.uni-sb.de/