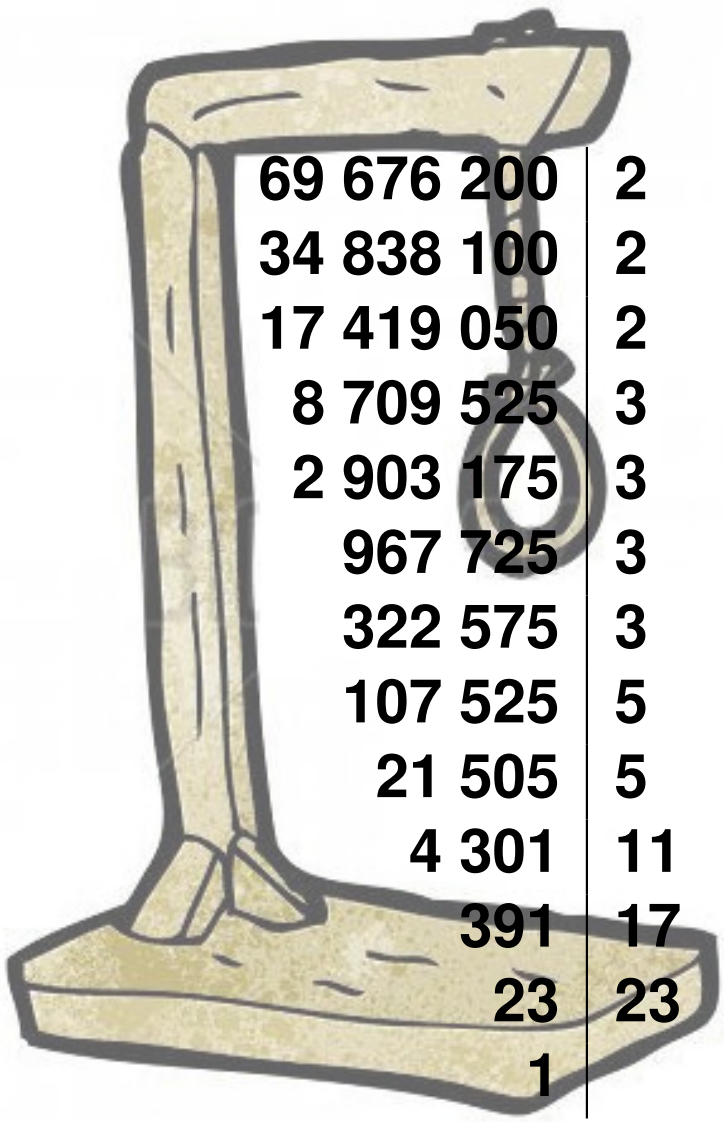
Egy (talán nem is olyan unalmas) rejtvény

Prímszám-e az alábbi (232 jegyű) N?

Ha nem, mi az N prímtényezős felbontása?

$N = 19276274018175879149728258618075604723624$
$76625567679170783212620286838088030022428259 4$
$03044415025081917195850003043729070625163485 4$
$17235287365918325957070141465180723970176735 4$
$99259074075047284383488122852375163022847202 9$
20617035001

Akasztófa módszer



$$N = a \cdot b, \quad a \leq b$$
$$\Downarrow$$
$$a \leq \sqrt{N}$$

Ha $N \approx 1,93 \times 10^{231}$, akkor

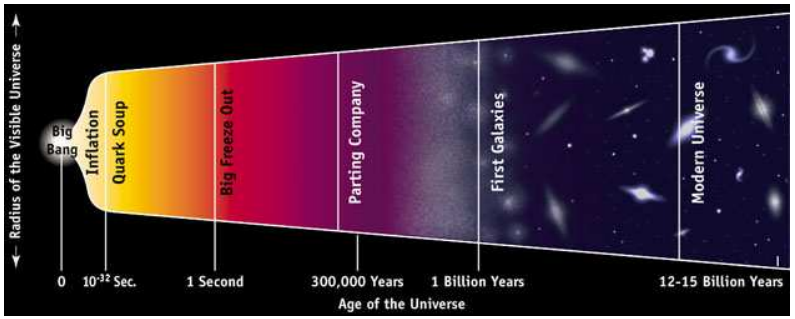
$$\sqrt{N} \approx 4,39 \times 10^{115}$$

osztásra is szükség lehet
(ha N prím vagy két nagyon közeli prím szorzata).

Katasztrófa módszer

- $N \approx 1,93 \times 10^{231} \Rightarrow \sqrt{N} \approx 4,39 \times 10^{115}$ osztás
- A leggyorsabb sci-fi szuperszámítógép:
  - az információ fénysebességgel terjed ($3 \times 10^8 m/s$)
  - egy processzor mérete: egy hidrogén atom átmérője ($5 \times 10^{-11} m$)
  - processzorok száma: a protonok száma az Univerzumban ($1,575 \times 10^{79}$)
  - $\Rightarrow 9,45 \times 10^{97}$ művelet ($\approx$ osztás) másodpercenként
  - A $4,39 \times 10^{115}$ osztás időigénye: $\approx 4,65 \times 10^{17}$ másodperc

- Az Univerzum életkora: 4.33×10^{17} másodperc



Mit nevezzünk hatékony algoritmusnak?

A definíció...

- ...nem függhet egy konkrét számítógéptől;

  futásidő $\longrightarrow$ **lépésszám**

- ...nem adhat meg univerzális időkorlátot;
- ...meg kell engedje, hogy a futásidő függjön az input méretétől.

Mit nevezzünk hatékony algoritmusnak?

Az input mérete: $\approx$ a tárolásához szükséges memória mérete (pl. bájtok száma).

Például:

$N = 192762740181758791497282586180 7\ldots$ (232 számjegy)

Az input mérete: $n = 232$

Mit nevezzünk hatékony algoritmusnak?

Az input mérete: n	„A” Algoritmus futásideje: 1.2^n	„B” Algoritmus futásideje: $100 \cdot n^5$
20	38,34	$3,2 \times 10^8$
50	$9,1 \times 10^3$	$3,13 \times 10^{10}$
100	$8,28 \times 10^7$	1×10^{12}
200	$6,86 \times 10^{15}$	$3,2 \times 10^{13}$
1000	$1,52 \times 10^{79}$	1×10^{17}
2000	$2,3 \times 10^{158}$	$3,2 \times 10^{18}$

Definíció

Egy algoritmus akkor **polinomiális futásidejű**, ha léteznek olyan $c > 0$ és $k > 0$ konstansok, hogy az algoritmus megáll legföljebb $c \cdot n^k$ lépés után minden n méretű input esetén.

Általában a polinomiális algoritmusok tekinthetők a gyakorlatban hatékonynak.

Mit nevezzünk hatékony algoritmusnak?

Az input mérete: n	„A” Algoritmus futásideje: 1.2^n	„B” Algoritmus futásideje: $100 \cdot n^5$
20	38,34	$3,2 \times 10^8$
50	$9,1 \times 10^3$	$3,13 \times 10^{10}$
100	$8,28 \times 10^7$	1×10^{12}
200	$6,86 \times 10^{15}$	$3,2 \times 10^{13}$
1000	$1,52 \times 10^{79}$	1×10^{17}
2000	$2,3 \times 10^{158}$	$3,2 \times 10^{18}$

Hasznos tény Analízisből:
ha $a > 1$, akkor az a^n exponenciális függvény „idővel túlnő” minden polinomfüggvényt.

(Precízen: minden rögzített $a > 1$ és $c, k \geq 0$ esetén van olyan n_0 küszöb, hogy $a^n > c \cdot n^k$, ha $n \geq n_0$.)

Ezért (például) az „A” algoritmus **NEM** polinomiális futásidejű!

Számelméleti algoritmusok

Számelméleti algoritmus: az input és az output is néhány egész számból áll.

Példa: írásbeli összeadás (Input: $a, b \in \mathbb{N}$, Output: $a + b$)

$$\begin{array}{r} 376067500620631546788072394447804559247439685 \\ + \quad 178938803682342419767517030643710875679678 \\ \hline \ldots 70123119363 \end{array}$$

a: k jegyű, b: ℓ jegyű (és $k \geq \ell$) $\Longrightarrow$ input mérete: $n = k + \ell$

Az írásbeli összeadás jegyenként fix számú, mondjuk legföljebb c darab elemi bitműveletet végez.

Ezért a lépésszáma $\leq c \cdot k \leq c \cdot n$

Így az írásbeli összeadás polinomiális futásidejű algoritmus.

(Sőt: ***lineáris futásidejű***: a lépésszáma $\leq c \cdot n^1$.)

Számelméleti algoritmusok

Példa: akasztófa módszer (Input: $N \in \mathbb{N}$)

N: n jegyű $\implies$ $10^{n-1} \leq N < 10^n$

$\sqrt{N}$ osztásra is szükség lehet:

$$\sqrt{N} \geq \sqrt{10^{n-1}} = \left(\sqrt{10}\right)^{n-1} > 2^n \quad \text{(ha } n \geq 3\text{)}$$

Mivel a 2^n exponenciális függvény „idővel túlnő” minden polinomfüggvényt, ezért
az akasztófa módszer **NEM** polinomiális futásidejű
(és így a gyakorlatban „nagy számokra” használhatatlan).

Ha egy számelméleti algoritmus lépésszáma magukkal az input szám(ok)kal (vagy pl. azok négyzetgyökével) arányos, akkor az **NEM** polinomiális futásidejű.

Ugyanis: az input méretétől, vagyis a számjegyek számától a szám értéke exponenciálisan függ.

Számelméleti algoritmusok

Mennyi a számjegyek száma (vagyis az input mérete)?

a: n jegyű $\implies$ $10^{n-1} \leq a < 10^n$

$n - 1 \leq \lg a < n$

Az a számjegyeinek száma 10-es számrendszerben: $\approx \lg a$.

Ha a számrendszer alapja t (például $t = 2$ vagy $t = 16$, stb): az a számjegyeinek száma $\approx \log_t a$.

Például 10-esről 2-es számrendszerre áttérve:

$$\log_2 a = \frac{\log_{10} a}{\log_{10} 2} \approx 3,32 \cdot \log_{10} a.$$

Ezért a polinomiális futásidő szempontjából **teljesen mindegy**, hogy az input milyen számrendszerben van megadva: az input méretét a számrendszerváltás csak egy rögzített konstansszorosára változtatja.

Számelméleti algoritmusok

Összefoglalva:

A számelméleti algoritmusok esetében

- az input mérete az input számok összes számjegyeinek száma,
- ami lényegében azonos az input számok logaritmusainak összegével (ahol a logaritmus alapja most mindegy);
- az algoritmus akkor polinomiális futásidejű, ha a lépésszáma $\leq c \cdot n^k$, ahol n az input számok összes számjegyeinek száma (és c és k rögzített);
- ha az algoritmus lépésszáma például az egyik input számmal arányos, akkor az **NEM** polinomiális futásidejű, mert a szám értéke a számjegyei számának exponenciális függvénye, ami „idővel túlnő” minden polinomfüggvényt.