Algoritmy a datové struktury

Úvod

Cíle předmětu

- Přemýšlet analyticky o algoritmech.
- Získat „algoritmickou sadu nástrojů“ (tj. naučit se některé často používané algoritmy z různých oblastí).

Obsah dnešní přednášky

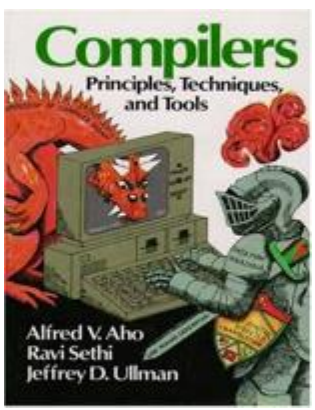
- Proč se učit o algoritmech?
- Co je to algoritmus
- Vlastnosti algoritmů
- Ukázka – násobení celých čísel
  - A můžeme to udělat rychleji?

Algoritmy jsou základ

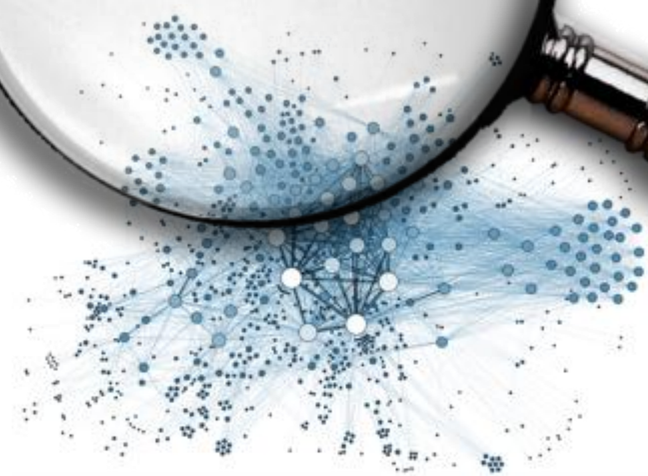
Operační systémy

Všude jsou algoritmy

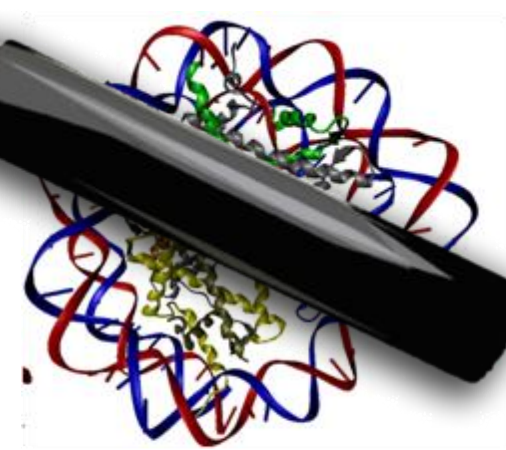
Kryptografie



Kompilátory)

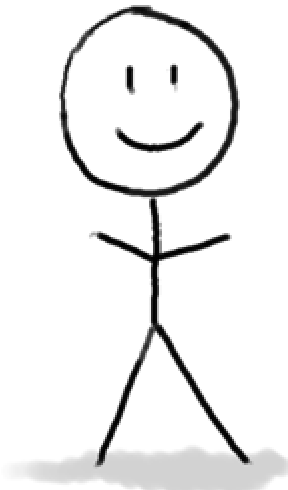


Sítě



Výpočetní biologie

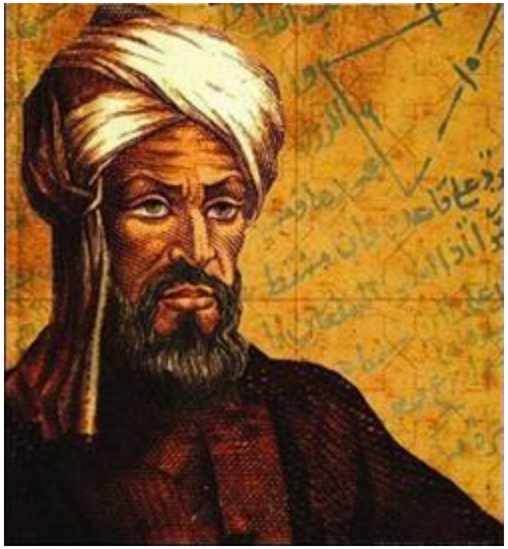
Naše hlavní otázky při návrhu algoritmů:

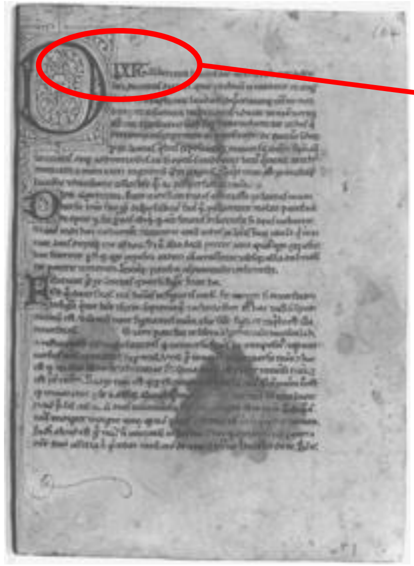


Funguje to?
Je to rychlé?
Mohu to udělat lépe?

Etymologie „algoritmu“

- Al-Khwarizmi byl učenec z 9. století, narozený v dnešním Uzbekistánu, který studoval a pracoval v Bagdádu během Abbassidského chalífátu.
- Mezi mnoha dalšími příspěvky v matematice, astronomii a geografii napsal knihu o tom, jak počítat příklady za použití arabských číslic.
- Jeho myšlenky se do Evropy dostaly ve 12. století.





Dixit algorizmi
(to říká Al-Khwarizmi)

- Původně „Algorisme“ [stará francouzština] odkazovala pouze na arabský číselný systém, ale nakonec se z toho vyvinul pojem „Algoritmus“, jak ho známe dnes.

Pojem algoritmus - intuitivně

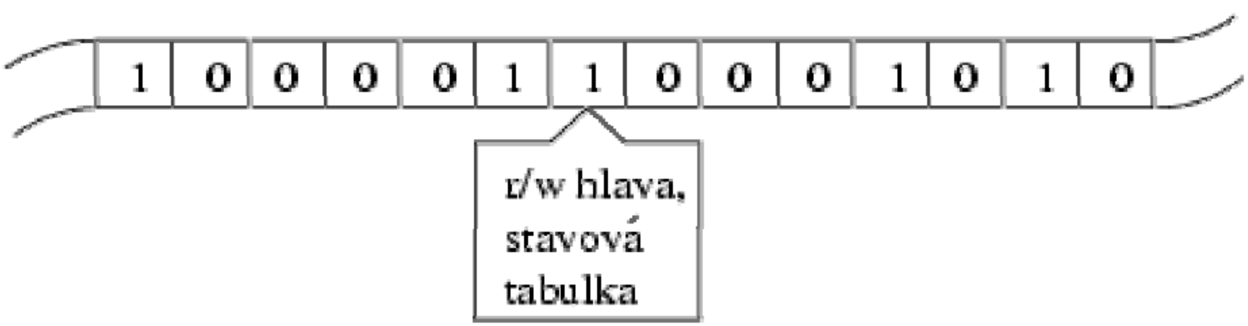
- **Algoritmus**
  - Všeobecná pravidla určující transformaci vstupních dat na výstupní.
  - Návod, který určuje postup vedoucí k řešení dané úlohy.
  - Posloupnost kroku vedoucích k řešení dané úlohy.
  - Např. recept – transformuje mouku, vodu, vajíčka na palačinky.
- **Program**
  - **Realizace algoritmu !**

Definice algoritmu - přesnější

- Algoritmus lze definovat jako jednoznačně určenou posloupnost konečného počtu elementárních kroků vedoucí k řešení daného problému (úlohy),přičemž, musí být splněny základní vlastnosti každého algoritmu.
- Toto je sice na první pohled pravdivá, ale při bližším prozkoumání nepřesná definice. Například některé matematické postupy by této definici vyhovovaly, ale nejsou algoritmy.
- Přesné znění definice algoritmu zní: „Algoritmus je procedura proveditelná Turingovým strojem.“
- Turingův stroj (1937) je teoretický model počítače popsaný matematikem Alanem Turingem. Skládá se z procesorové jednotky, tvořené konečným automatem a programu ve tvaru pravidel přechodové funkce a potenciálně nekonečné pásky pro zápis mezivýsledků a vstupů dat. Využívá se pro modelování algoritmů, v teorii vyčíslitelnosti.

Výpočet Turingova stroje

- Výpočet začíná tak, že jsou na pásce uložena počáteční data (pokud jsou nějaká) a vlastní kód programu. Hlava je pak uvedena do stavu, který odpovídá načtení kódu programu a stroj tak započne výpočet, přechází mezi stavy a po skončení většinou zapíše výsledek.



Základní vlastnosti algoritmu

- **Hromadnost**
  - Algoritmus je použitelný na libovolné vstupní údaje splňující požadované podmínky
  - Algoritmus neřeší jeden konkrétní problém, ale obecnou třídu obdobných problému
  - Např. nikoliv výpočet 2x8, ale součin dvou celých čísel
- **Determinismus**
  - Každý krok algoritmu musí být jednoznačně a přesně definován; v každé situaci musí být naprosto zřejmé, co a jak se má provést, jak má provádění algoritmu pokračovat.
- **Rezultativnost (konečnost)**
  - Algoritmus při zadání vstupních dat vždy vrátí nějaký výsledek (může se jednat i jen o chybové hlášení). V konečném poctu kroku musí algoritmus vrátit výsledek

Základní vlastnosti algoritmu

- **Efektivní (efektivnost)**
  - Obecně požadujeme, aby algoritmus byl efektivní, v tom smyslu, že požadujeme, aby každá operace požadovaná algoritmem, byla dostatečně jednoduchá na to, aby mohla být alespoň v principu provedena v konečném čase pouze s použitím tužky a papíru. (tj. byla elementární).
  - **Efektivita**
  - časová - spotřeba času, důležitější - měl by být rychlejší?
  - paměťová - spotřeba paměti
  - přehlednost a srozumitelnost
- **Správný (správnost)**
  - Algoritmus je správný tehdy, když pro všechny údaje splňující vstupní podmínku se proces zastaví a výstupní údaje splňují výstupní podmínku.
- Algoritmus nesmí být závislý na prostředí, ve kterém je realizován.

Etapy řešení algoritmické úlohy

1) Zadání - Formulace problému, stanovení cílů.

2) Rozbor, analýza - volba strategie, navržení postupu.
   - *specifikace*
   - *vstupy - identifikátor, popis, typ proměnné, vstupní podmínky*
   - *výstupy - identifikátor, popis, typ proměnné*
   - *(další proměnné)*

3) Algoritmus – *návrh – můžeme použít různý zápis*
   - *Pseudokód*
   - *vývojový diagram*
   - *Zápis v programovacím jazyce*
   - *Strukturovaný zápis v přirozeném jazyce*

4) Testování, ověření správnosti.

5) Přepis do progr. jazyka (kódování).
   - Ladění
   - Optimalizace

6) Dokumentace.

Ověření logické správnosti algoritmu

- K ověření správnosti algoritmu nestačí vyzkoušet reakci algoritmu na konečný počet vstupních dat, i když se to tak v praxi často dělá a takové ověření o správnosti algoritmu leccos vypoví.
- Pro ověřování správnosti algoritmu neexistuje univerzální metoda, algoritmus by měl být matematicky dokázán sledem předem známých kroků (operací), které nevyvratitelně vedou pro všechna přípustná data ke správnému výsledku úlohy.
- Algoritmus můžeme považovat za korektní, pokud není opomenuta žádná z možností zpracování dat při průchodu algoritmem.
- Algoritmus je částečně (parciálně) správný, právě když platí, že pokud skončí, vydá správný výsledek.
- Nutné je také ověřit konečnost algoritmu (pro všechna přípustná data algoritmus po konečném počtu kroků skončí).

Vytváření algoritmu

- Návrh algoritmu je tvořivý proces (nelze automatizovat).
- Neexistuje všeobecný návod.
- Existují známé strategie a paradigmata pro návrh algoritmu.

Všeobecné strategie

- **Dekompozice** – rozdělení problému na podproblémy, které jsou z určitého hlediska jednodušší.
- **Abstrakce** – zanedbání a/nebo ukrytí detailu.

Klasifikace algoritmu podle implementace

Rekurzivní/iterační - rekurzivní algoritmus volá sám sebe dokud není dosažena ukončovací podmínka.

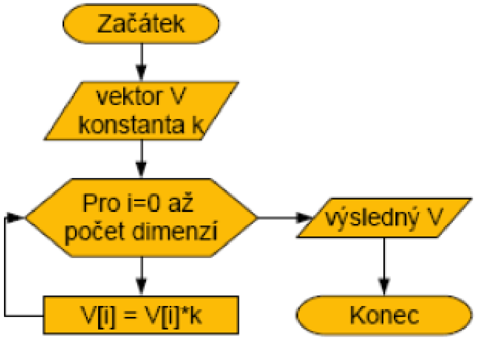
- Iterativní algoritmus používá opakující se konstrukce (cykly).
- Některé problémy lze jednoduše řešit rekurzí, jiné iteračně.

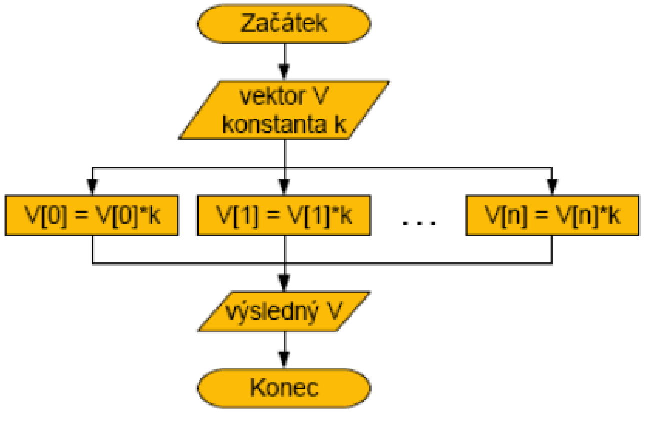
```
Stromecek (délka , úhel )
{
Kresli (délka , úhel );
Stromecek ( délka /2, úhel +45);
Stromecek ( délka /2, úhel -45);
}
```

Klasifikace algoritmu podle implementace

Sériové/paralelní

- Využívají výhody paralelního zpracování, dnes např. použití GPU.
- Ne vždy lze převádět sériový algoritmus na paralelní.
- Ne vždy je výhodné převádět sériový algoritmus na paralelní.





Klasifikace algoritmu podle implementace

Deterministické/náhodné

- Deterministické algoritmy vracejí pro stejný vstup vždy stejný výsledek.
- Náhodné algoritmy prohledávají náhodně prostor až dosáhnou přijatelné řešení.
- Např. nalezení znaku ’a’ v poli s 50% znaku ’a’ a 50% znaku ’b’.

Klasifikace algoritmu podle implementace

Přesné/přibližné

- Hledají přesné nebo přibližné řešení problému.
- U numerických výpočtů nelze určit přesné řešení kvůli konečné aritmetice.
- U grafových problému nelze určit přesné řešení kvůli složitosti problému.

  Např. hledání kořene rovnice.

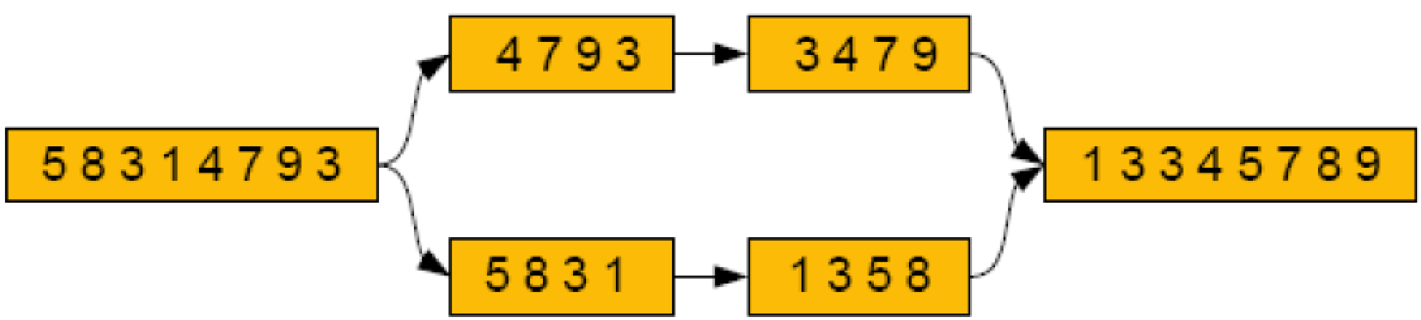
  Pokud existuje analytické řešení – přesný výsledek (kořen kvadratické rovnice).

  Jinak nutno použít numerickou metodu (např. metodu půlení intervalu).

Klasifikace algoritmu podle použitého paradigmata

Rozděl a panuj (Divide and conquer)

- Opakovaně redukovat problém na jeden či více menších problému dokud nelze jednoduše vyřešit.
- Dělení většinou probíhá rekurzivně.
- Např. mergesort – ve více lidech abecedně setřídit knihy.



Klasifikace algoritmu podle použitého paradigmatu

Sniž a panuj (Decrease and conquer)

- Jednoduší verze rozděl a panuj.
- Lze použít v případě, že vyřešení malé části je současně i řešením celku .
- Např. hledání půlením intervalu – hledání v telefonním seznamu.

Klasifikace algoritmu podle použitého paradigmatu

Žravé metody (Greedy methods)

- V každém kroku se vybírá možnost, která se jeví nejlepší.
- Hledá lokální optimum.
- Nevrací se zpět.
- Když funguje, bývá nejrychlejší.
- Např. Kruskaluv algoritmus pro minimální kostru grafu.

Klasifikace algoritmu podle použitého paradigmata

Redukce

- Převádí problém na jiný problém, jehož řešení je známé a není příliš náročné.
- Např. hledání mediánu v nesetříděném poli lze transformovat na problém řazení pole a výběru prostředního prvku v setříděném poli.

Klasifikace algoritmu podle použitého paradigmata

Dynamické programování

- Při řešení problému se postupuje přes řešením série podproblému, atd.
- Řešení podproblému se ukládají do tabulky, aby se předešlo opakovaným výpočtům.
- Podproblémy se překrývají
- Např. výpočet hodnot Pascalova trojúhelníku.

1

1 1

1 2 1

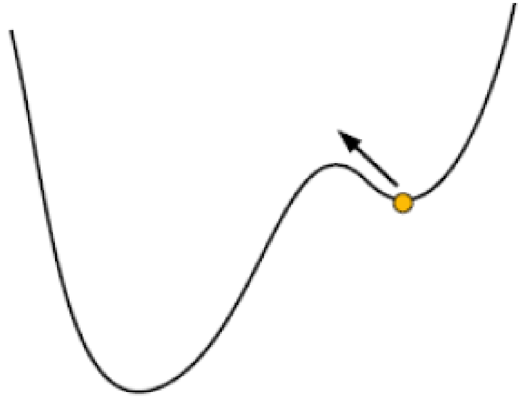
1 3 3 1

1 4 6 4 1

Klasifikace algoritmu podle použitého paradigmata

Heuristické algoritmy

- Na základe heuristiky prohledávají prostor řešení.
- Většinou nevracejí přesné řešení, ale pouze aproximaci, protože přesné řešení není možné najít v rozumném čase.
- Např. hledání globálního extrému funkce bez omezení intervalu metodou simulovaného žíhání.
- Prozkoumává blízké okolí dosavadního řešení a s pravděpodobností T (teplota)
- Přijímá se i horší řešení.
- Může se dostat i z lokálních extrému.



Klasifikace algoritmu podle použitého paradigmata

Algoritmy inspirované biologií

- Genetické algoritmy.
- Pracují na základě napodobování biologických evolučních procesů. Pro vědu jsou přirozené procesy probíhající v přírodě velkým vzorem, protože mohou být chápány a aplikovány i jako možná řešení daného problému.

Klasifikace algoritmu podle složitosti

- **K některým problémům existuje hned několik řešení.**
- Některá z nich se pro rostoucí velikost vstupní množiny stávají nepoužitelná, jiná pracují rychle.
- Pro udávání rychlosti algoritmu nemá cenu uvádět časový údaj – pro každý počítač je jiný.
- Složitost algoritmu ukazuje trend růstu času/paměti s rostoucí vstupní množinou.

Ukázka přístupu - přemýšlet analyticky o algoritmech.

Násobení

- Něco co dobře znáte ze základní školy.
- Dá se na něm něco vylepšit?

To byl trochu velký problém

XLIV × XCVII = ?

$$
\begin{array}{r}
44 \\
\times\ 97 \\
\hline
\end{array}
$$

Násobení celého čísla

$$\begin{array}{r} 44 \\ \times\ 97 \\ \hline \end{array}$$

Násobení celého velkého čísla

$$
\begin{array}{r}
12345678959314 13 \\
\times \quad 45638235203955 33 \\
\hline
\end{array}
$$

Násobení celého čísla

$$
\begin{array}{lr}
 & \overbrace{1233925720752752384623764283568364918374523856298}^{n} \\
\text{x} & 4562323582342395285623467235019130750135350013753 \\
\hline
\end{array}
$$

Jak rychlý je algoritmus násobení základní školy?

???

(Kolik jednociferných operací?)

Asi n^2 jednomístných operacích

Každé číslo dole musíme vynásobit všemi čísly nahoře, tedy maximálně n x n (n^2) násobení a pak musíme sečíst n různých n-místných čísel ...

Big-Oh notace

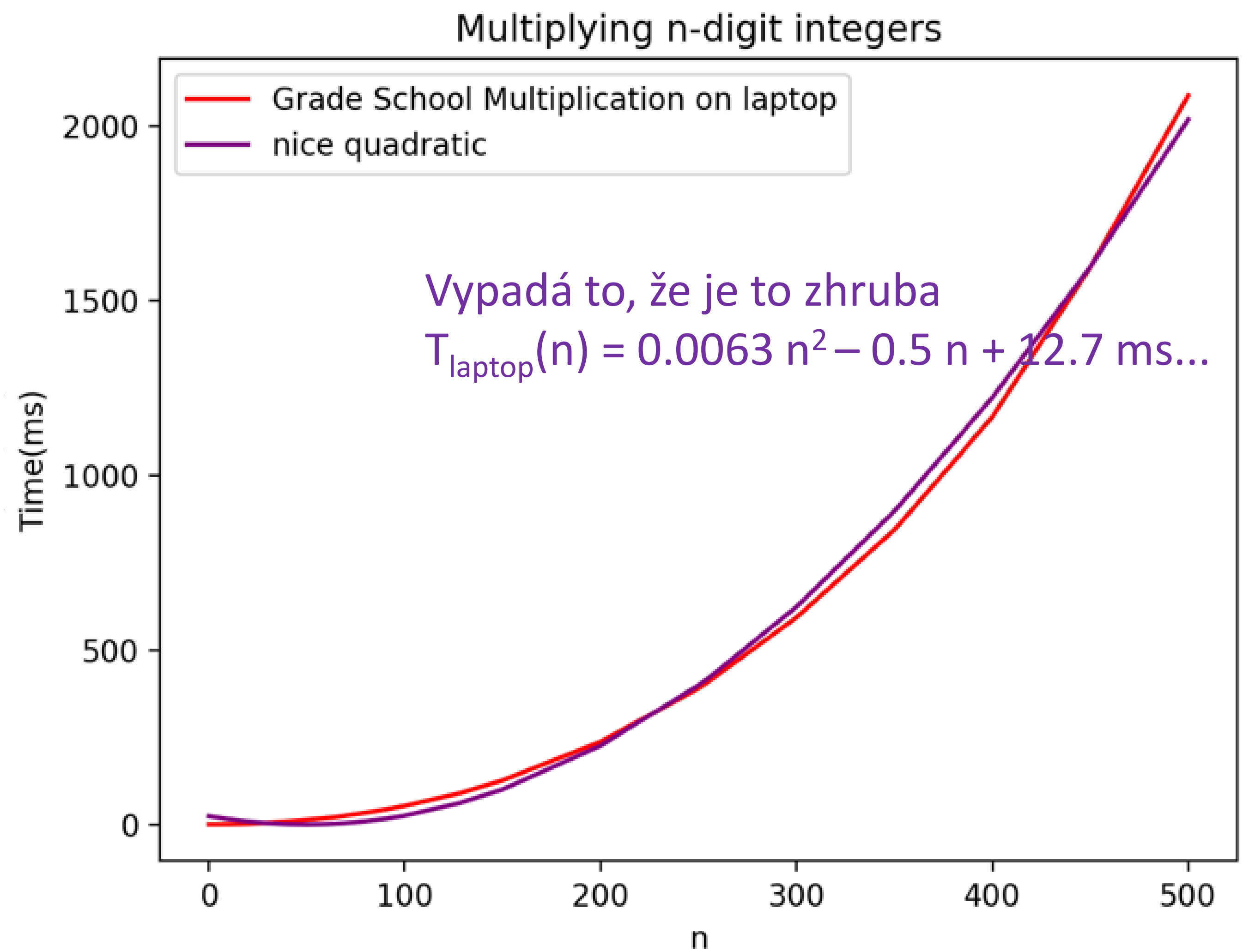
- Říkáme, že násobení na základní škole

  „běží v čase $O(n^2)$“

- Formální definice přijde příště!
- Neformálně nám notace big-Oh říká, jak se doba běhu mění s velikostí vstupu.

Doba běhu na počítači a graficky znázorněná pro různé *n*

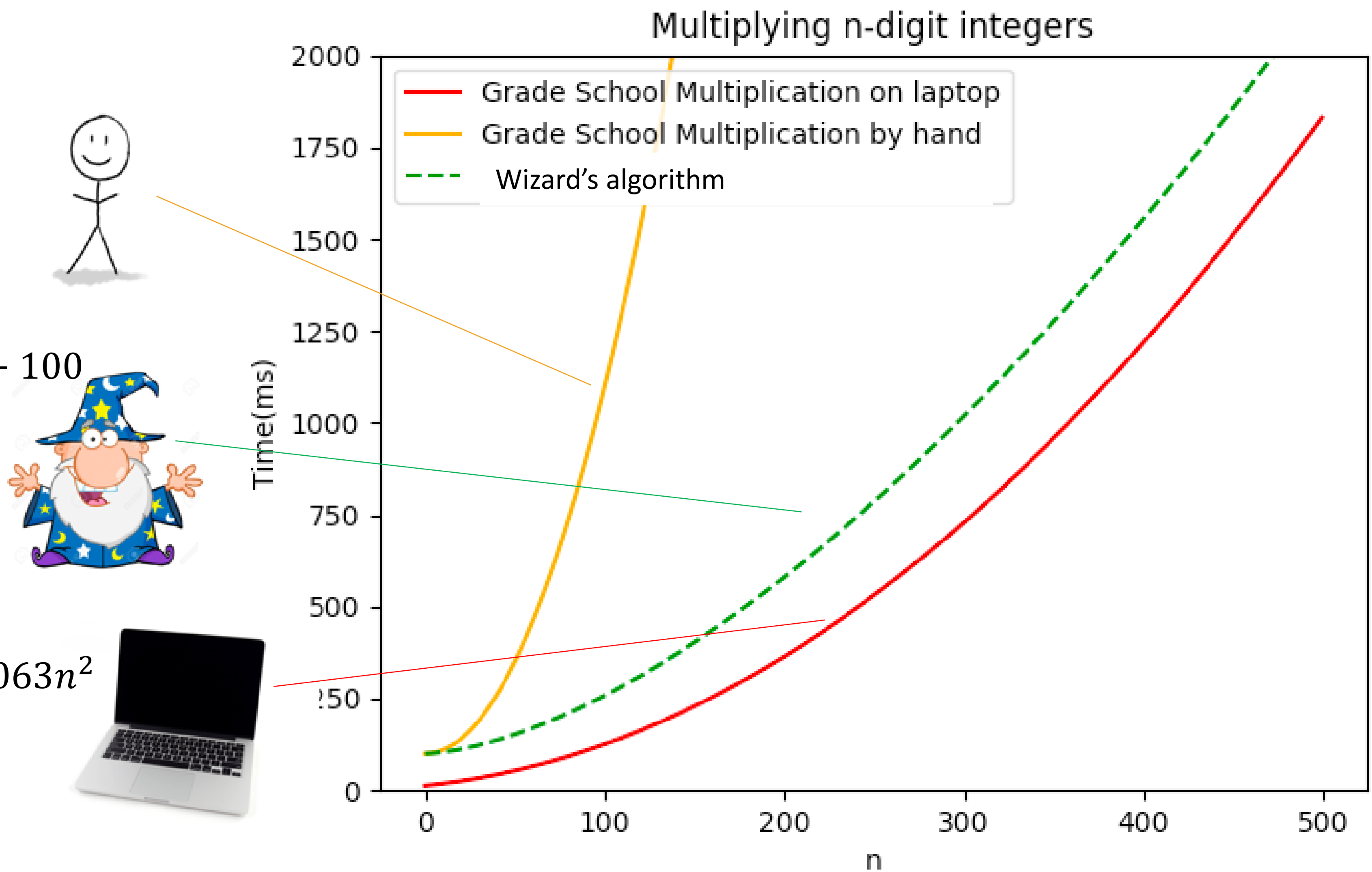
Doba běhu „se podobá“ n^2



Vypadá to, že je to zhruba

$T_{laptop}(n) = 0.0063\ n^2 - 0.5\ n + 12.7$ ms...

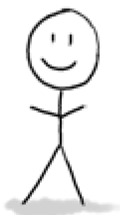
Proč má notace big-oh smysl?



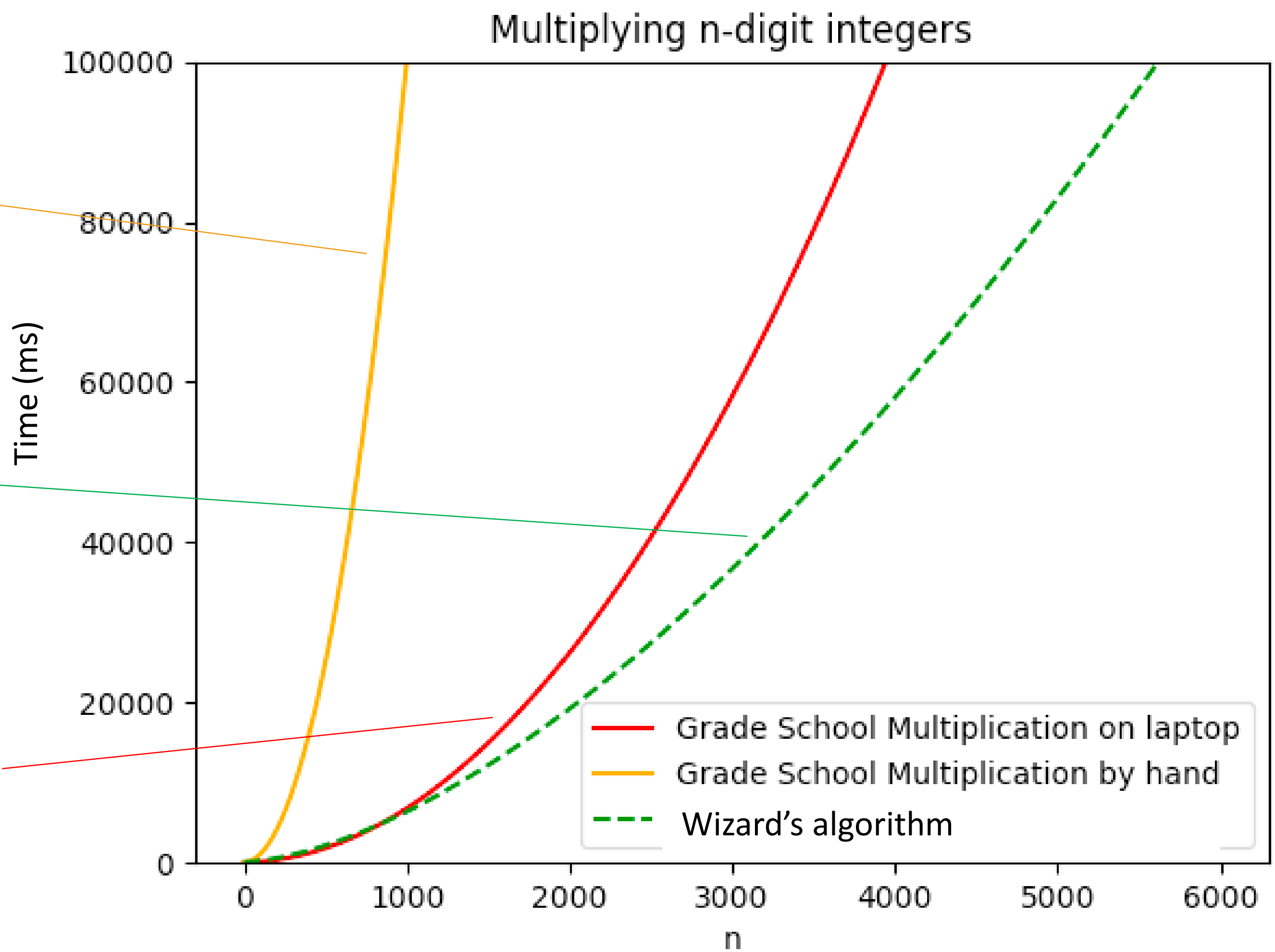
$$\approx \frac{n^{1.6}}{10} + 100$$

$$\approx .0063n^2$$

Vezmeme větší *n* …



$$\approx \frac{n^{1.6}}{10} + 100$$



$$\approx .0063n^2$$

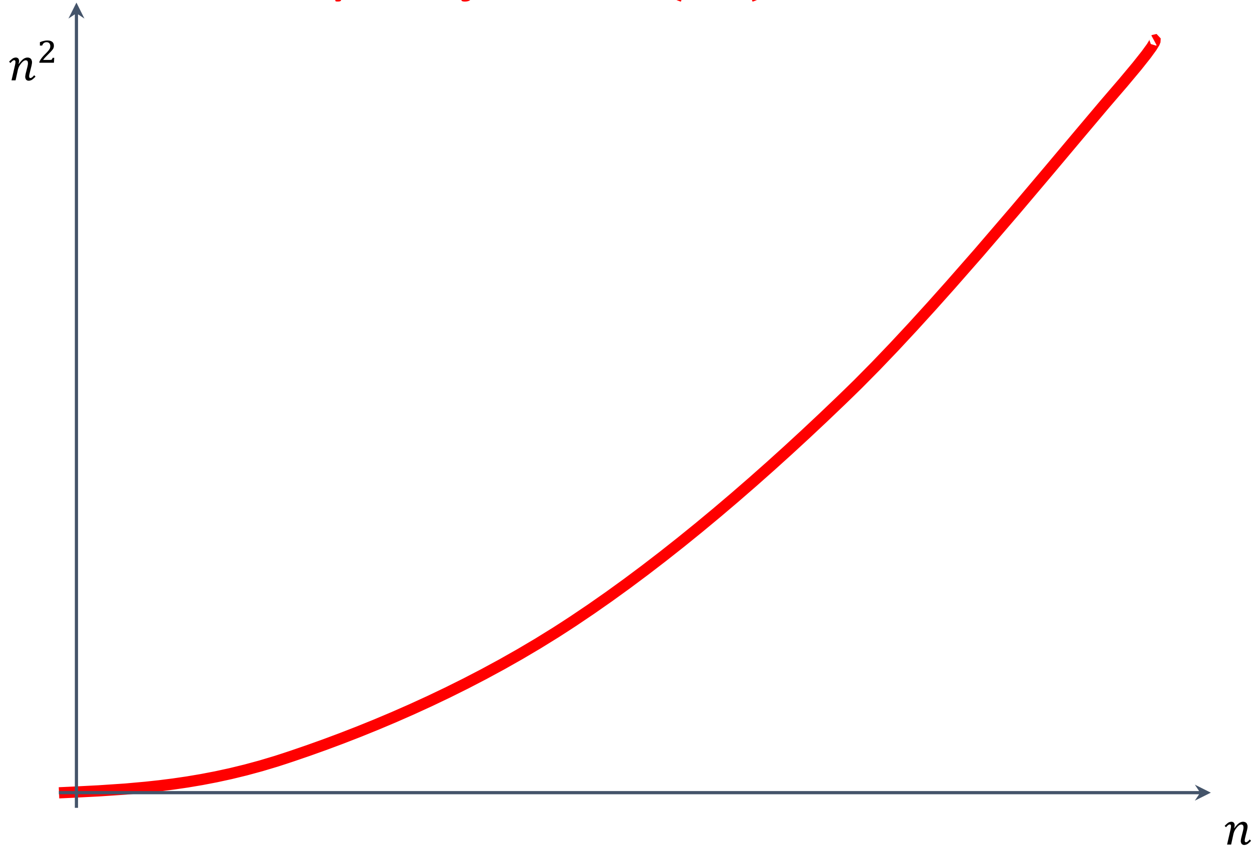
Závěr pro nás

- Algoritmus, který běží v čase $O(n^{1.6})$, je „lepší“ než algoritmus, který běží v čase $O(n^2)$.

- Otázka tedy zní ...

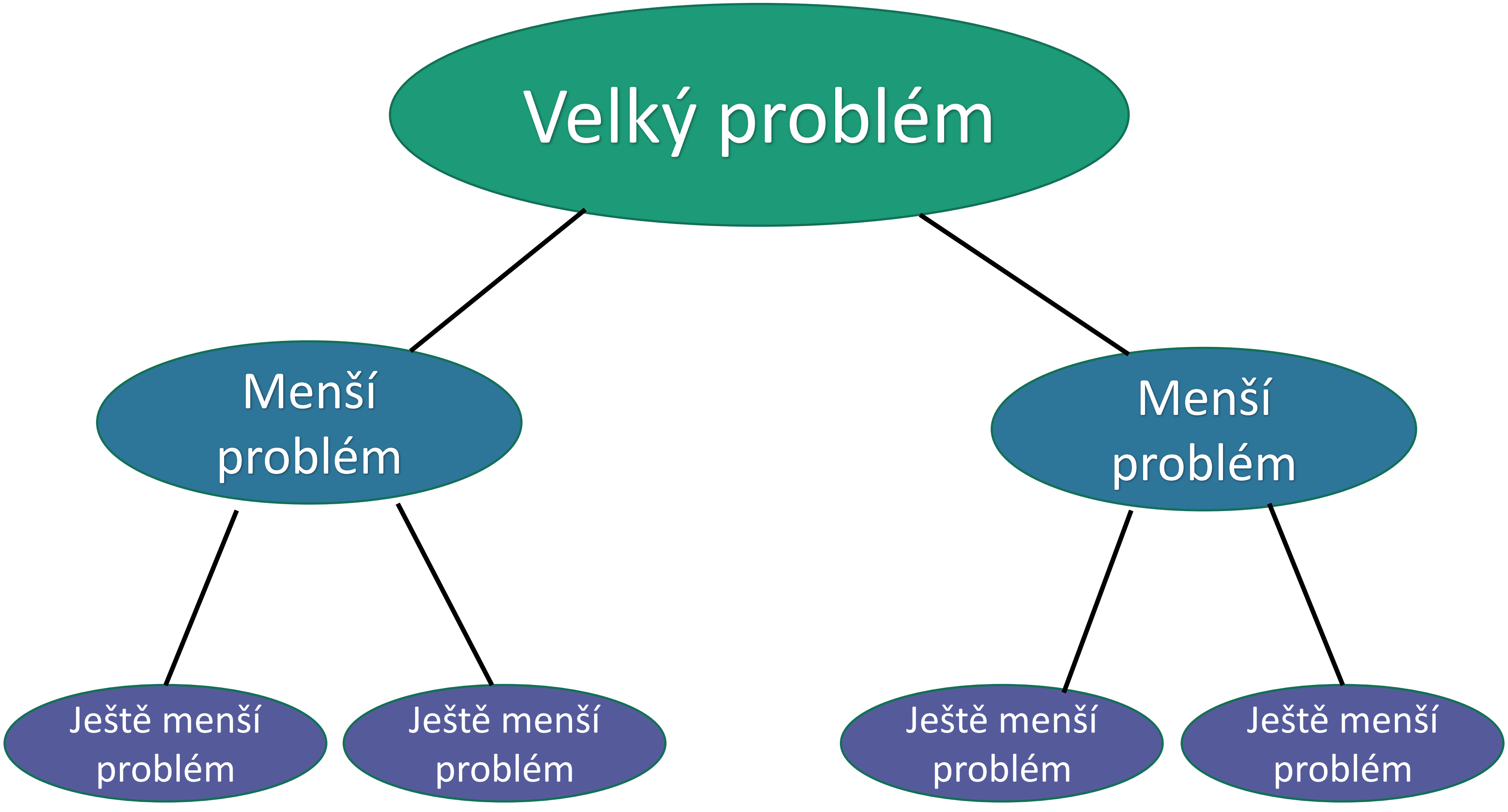
Můžeme násobení udělat lépe?

Můžeme znásobit celá čísla n číslic rychleji než $O(n^2)$?



Technika „Rozděl a panuj“

Rozdělíme problém na menší (snadnější) dílčí problémy



Rozděl a panuj pro násobení

Rozdělíme celé číslo:

$$1234 = 12 \times 100 + 34$$

$$1234 \times 5678$$

$$= (\, 12 \times 100 + 34 \,)\,(\, 56 \times 100 + 78 \,)$$

$$= \underbrace{(\, 12 \times 56 \,)}_{1}10000 + (\, \underbrace{34 \times 56}_{2} + \underbrace{12 \times 78}_{3} \,)100 + \underbrace{(\, 34 \times 78 \,)}_{4}$$

Jedno čtyřciferné násobení → Čtyři dvouciferné násobení

Obecněji

Pro zjednodušení předpokládáme, že n je sudé

Rozdělíme n-místné celé číslo:

$$[x_1 x_2 \cdots x_n] = [x_1 x_2 \cdots x_{n/2}] \times 10^{n/2} + [x_{n/2+1} x_{n/2+2} \cdots x_n]$$

$$\begin{aligned} x \times y &= (a \times 10^{n/2} + b)(c \times 10^{n/2} + d) \\ &= \underbrace{(a \times c)}_{1} 10^n + (\underbrace{a \times d}_{2} + \underbrace{c \times b}_{3}) 10^{n/2} + \underbrace{(b \times d)}_{4} \end{aligned}$$

Jedno n-ciferné násobení $\Rightarrow$ Čtyři (n/2)-ciferné násobení

Přístup rozděl a panuj

(Předpokládáme, že n je sudé …)

x,y mají n-číslic

Násobení(x, y):

Pokud n=1:

Vrať xy

Základ: znáte nazpaměť násobení jednociferných čísel …

Zapiš $x = a\ 10^{\frac{n}{2}} + b$

Zapiš $y = c\ 10^{\frac{n}{2}} + d$

a, b, c, d jsou čísla, které mají n/2-číslic

Rekurzivně spočti ac, ad, bc, bd:

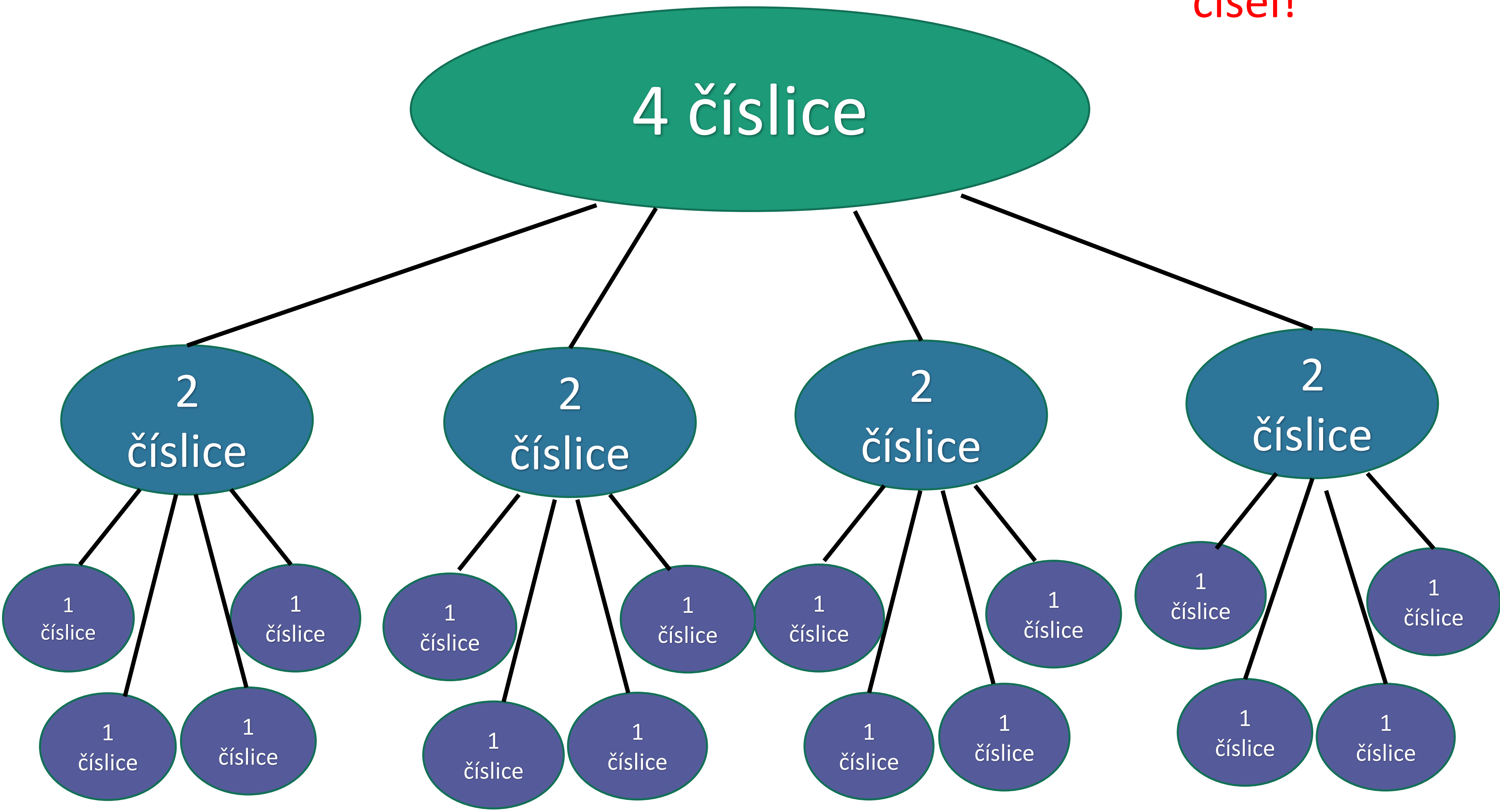
ac = **Vynásob**(a, c), etc..

Sečteme ac, ad,bc,bd, abychom dostali xy:

$xy = ac\ 10^{n} + (ad + bc)\ 10^{n/2} + bd$

Strom rekurze

Násobí se 16 jednociferných čísel!



Jaká je doba běhu?

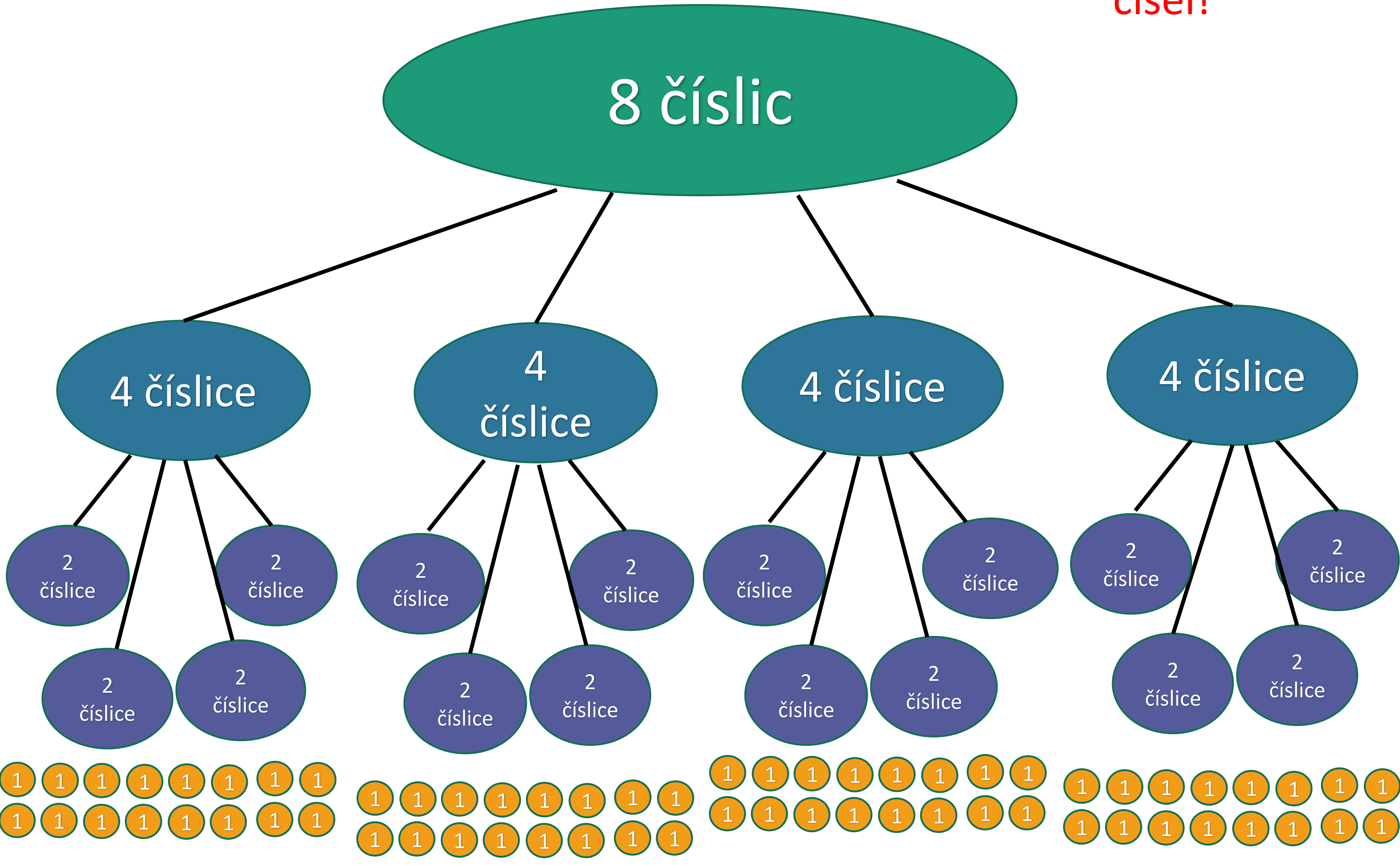
- Lepší nebo horší než algoritmus základní školy?

- Jak na tuto otázku odpovíme?

  1. Pokusem – na počítači – implementujeme příslušný algoritmus a sledujeme dobu běhu se vzrůstajícím počtem číslic v číslech, které násobíme
  2. Ukážeme, jak totu otázku řešit analyticky.

Strom rekurze

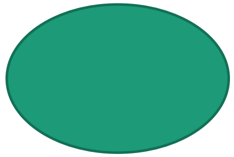
Násobíme 64 jednociferných čísel!



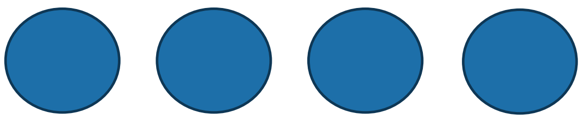
Požadavek byl:

Doba běhu tohoto algoritmu je
NEJMÉNĚ n^2 operací

Máme zde n^2 1-ciferných problémů

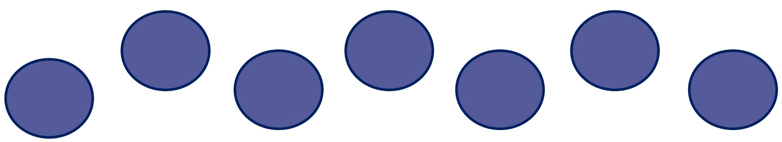


1 problém
velikosti n



4 problémy
Velikosti n/2

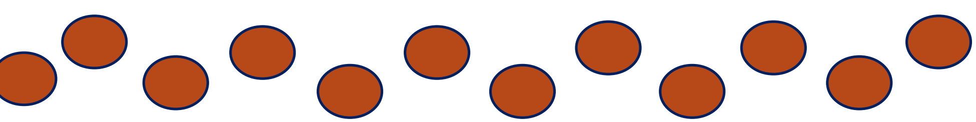
…



4^t problémy
velikosti $n/2^t$

Poznámka: toto je jen schéma – nebudeme kreslit všechny 4^t kruhy!

…



$\underline{\;n^2\;}$ problémů
Velikosti 1

- Pokud rozdělíme n na polovinu $\log_2(n)$ krát, dostanete se dolů až k 1.
- Tedy na úrovni
  $t = \log_2(n)$
  dostaneme …

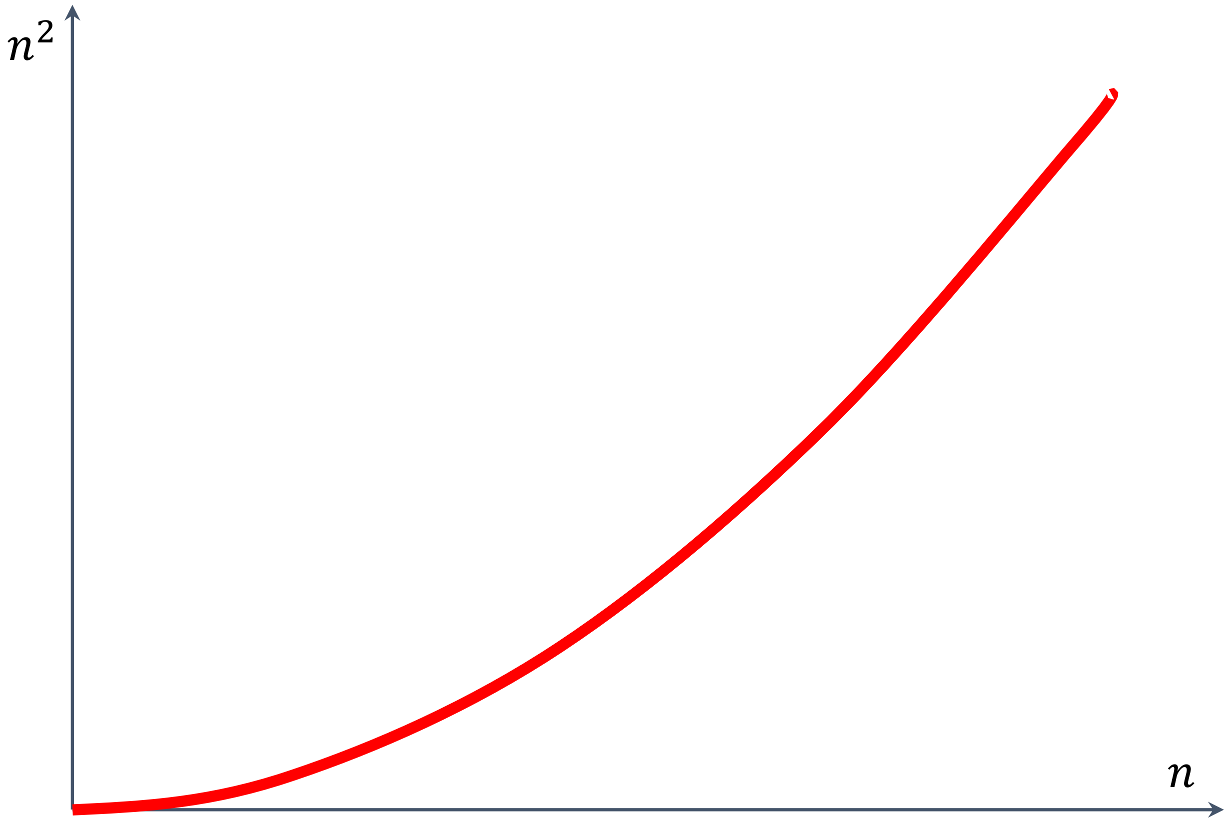
$$4^{\log_2 n} = n^{\log_2 4} = n^2$$

problémů velikosti 1.

Připomínám vzorec z logaritmů:
$a^{\log_a x} = x$

To je trochu zklamáním

Všechno to dobře funguje, ale stále je (nejméně) $O(n^2)$...



Technikou „Rozděl a panuj“ můžeme ale dosáhnout pokroku

- Karatsuba přišel na to, jak to udělat lépe!

$$
\begin{aligned}
xy &= (a \cdot 10^{n/2} + b)(c \cdot 10^{n/2} + d) \\
&= ac \cdot 10^{n} + (ad + bc)10^{n/2} + bd
\end{aligned}
$$

Potřebujeme tyto tři členy

- Kdybychom mohli počítat třeba jen tři věci místo původních čtyř ...

Násobení celých čísel dle Karatsuby

- Rekurzivně spočítejme tyto TŘI věci :
  - ac
  - bd
  - (a+b)(c+d)

Odečteme je

získáme toto

(a+b)(c+d) = ac + bd + bc + ad

- Sestavíme součin:

$$
\begin{aligned}
xy &= (a \cdot 10^{n/2} + b)(c \cdot 10^{n/2} + d) \\
&= ac \cdot 10^{n} + (ad + bc)10^{n/2} + bd
\end{aligned}
$$

✓ ✓ ✓

Jak Karatsubovo násobení funguje?

Násobeni(x, y):

Stále předpokládáme, že n je sudé

x,y jsou n-ciferná čísla

Pokud n=1:

Vrať xy

a, b, c, d jsou n/2-ciferná čísla

Zapiš $x = a\ 10^{\frac{n}{2}} + b$ a $y = c\ 10^{\frac{n}{2}} + d$

ac = **Vynásob**(a, c)

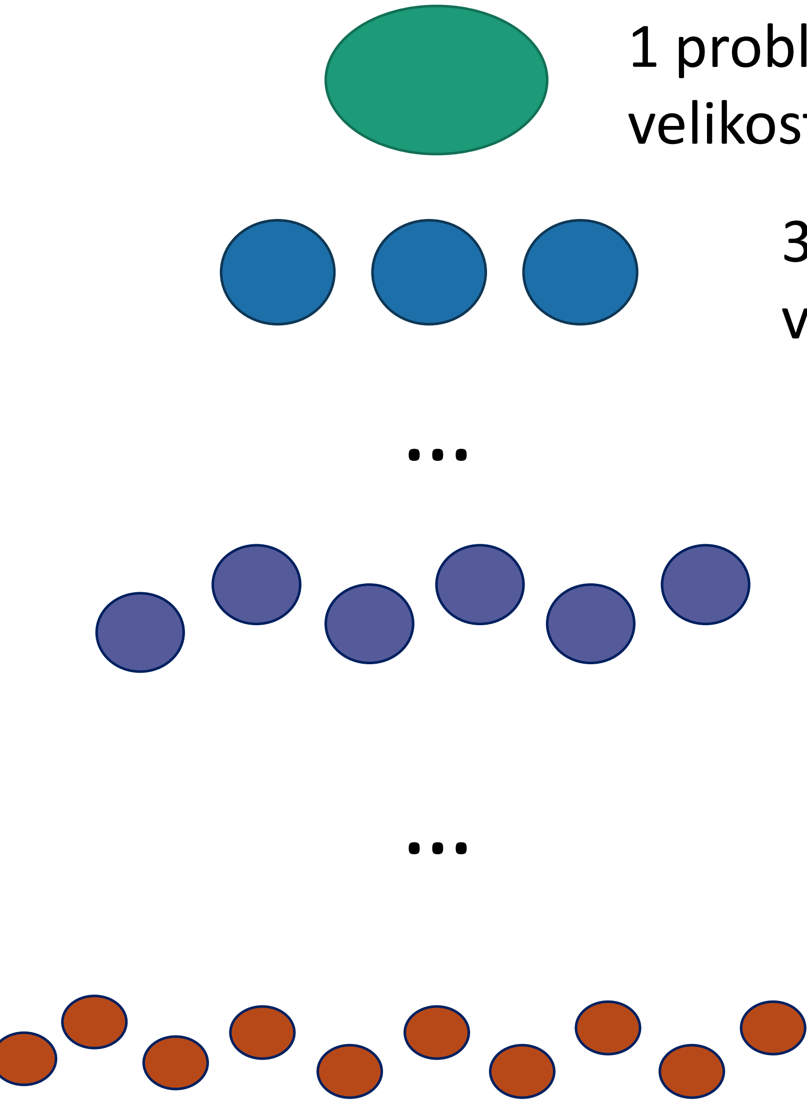
bd = **Vynásob**(b, d)

z = **Vynásob**(a+b, c+d)

xy = ac $\mathbf{10^{n}}$ **+ (z – ac - bd)** $\mathbf{10^{n/2}}$ **+ bd**

Vrať xy

Jaký je čas běhu?



1 problém
velikosti n

3 problémy
velikosti n/2

3^t problémů
velikosti $n/2^t$

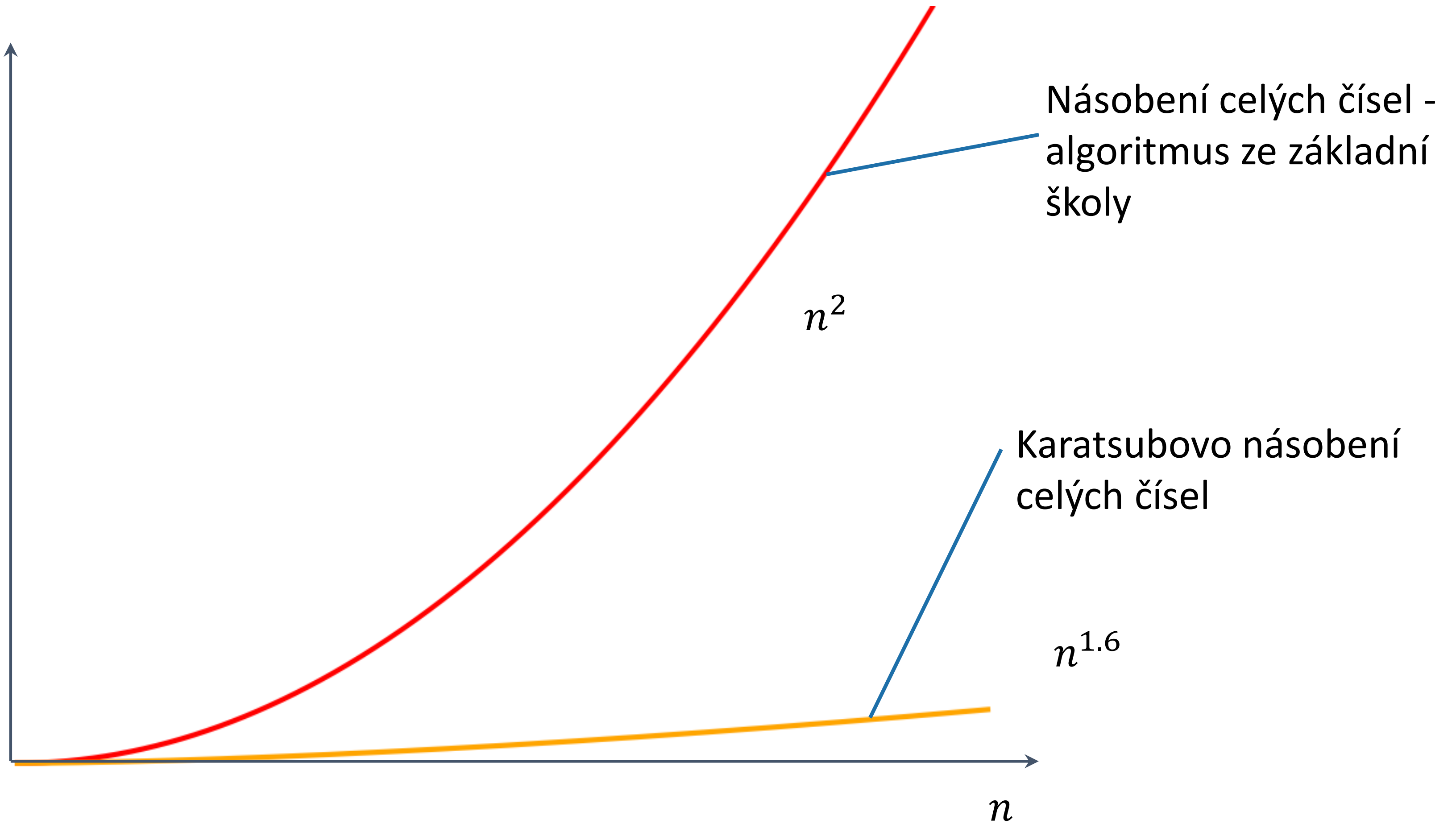
$\underline{n^{1.6}}$ problémů
Velikosti 1

- Pokud rozdělíme *n* $\log_2(n)$ krát, dostanete se dolů až k 1.
- Takže na úrovni
  $t = \log_2(n)$
  dostaneme …

$$3^{\log_2 n} = n^{\log_2 3} \approx n^{1.6}$$

problémů velikosti 1.

To je mnohem lepší!



Můžeme to udělat ještě lépe?

- **Toom-Cook** (1963): místo tří problémů velikosti n/2, navrhl rozdělit do pěti problémů velikosti n/3.
  - Doba běhu $O(n^{1.465})$
- **Schönhage–Strassen** (1971):
  - Doba běhu $O(n\log(n)\log\log(n))$
- **Furer** (2007)
  - Doba běhu $n\log(n)\cdot 2^{O(\log^*(n))}$
- **Harvey and van der Hoeven** (2019)
  - Doba běhu $O(n\log(n))$

Co jsem se naučili

- Algoritmus a jeho vlastnosti.
- Karatsubovo násobení celých čísel.
- Požívaná technika (přístup) řešení algoritmů:
  - Rozděl a panuj
- Nástroj pro analýzu algoritmů:
  - Úvod do asymptotické analýzy (asymptotické složitosti algoritmů)

Příště

- Řazení
- Asymptotická složitost a (formální) definice Big-Oh notace
- Přístup: Rozděl a panuj - další