Přednáška 3

MergeSort

Rozděl a panuj – analýza

Složitost rekurzivních algoritmů

Závěr z minula

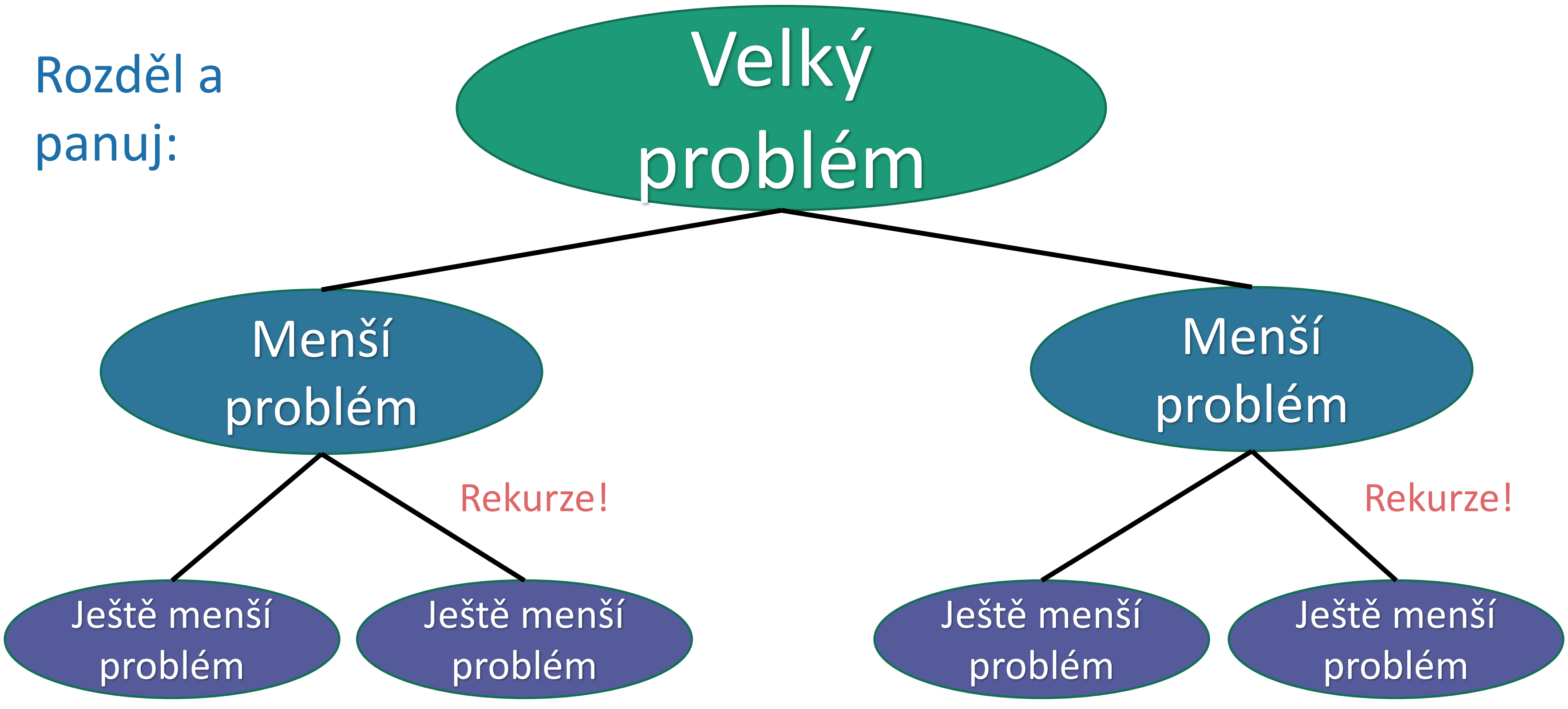
Insert sort (a Selection sort a Bubble sort) je algoritmus, který správně seřadí libovolné n-rozměrné pole v čase $O(n^2)$.

Můžeme to udělat lépe?

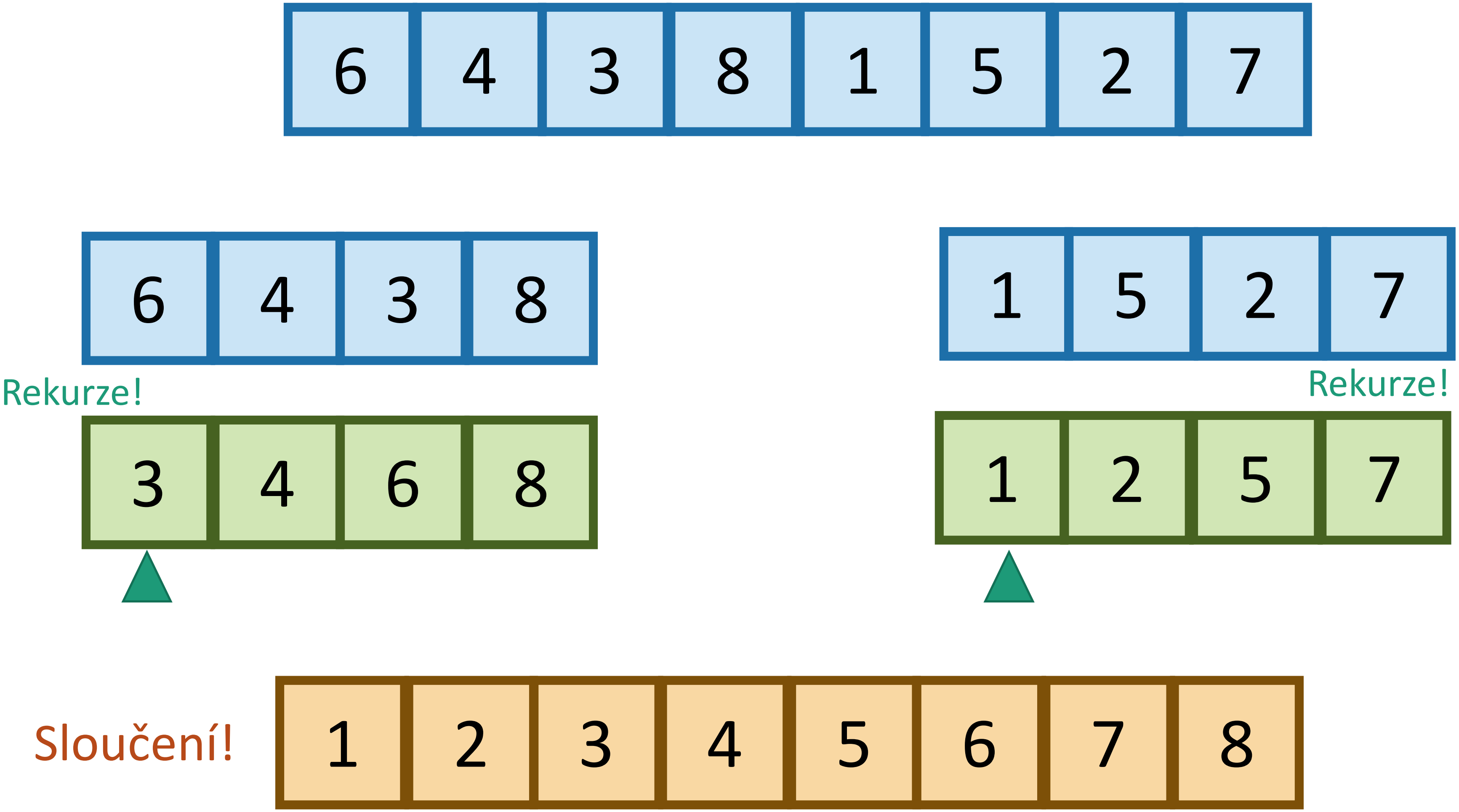
Můžeme to (seřazení) udělat lépe?

- Merge sort: rozděl-a-panuj přístup
- Připomeňme:

Rozděl a panuj:



Merge sort



Jak to uděláme na místě (v jednom poli?

Merge sort pseudokód (část)

MERGESORT(A):

- n = length(A)
- **if** n $\leq$ 1:
  - **return** A

Pokud A má délku 1, už je seřazeno!

- L = MERGESORT(A[0 : n/2])

Seřaď levou polovinu

- R = MERGESORT(A[n/2 : n])

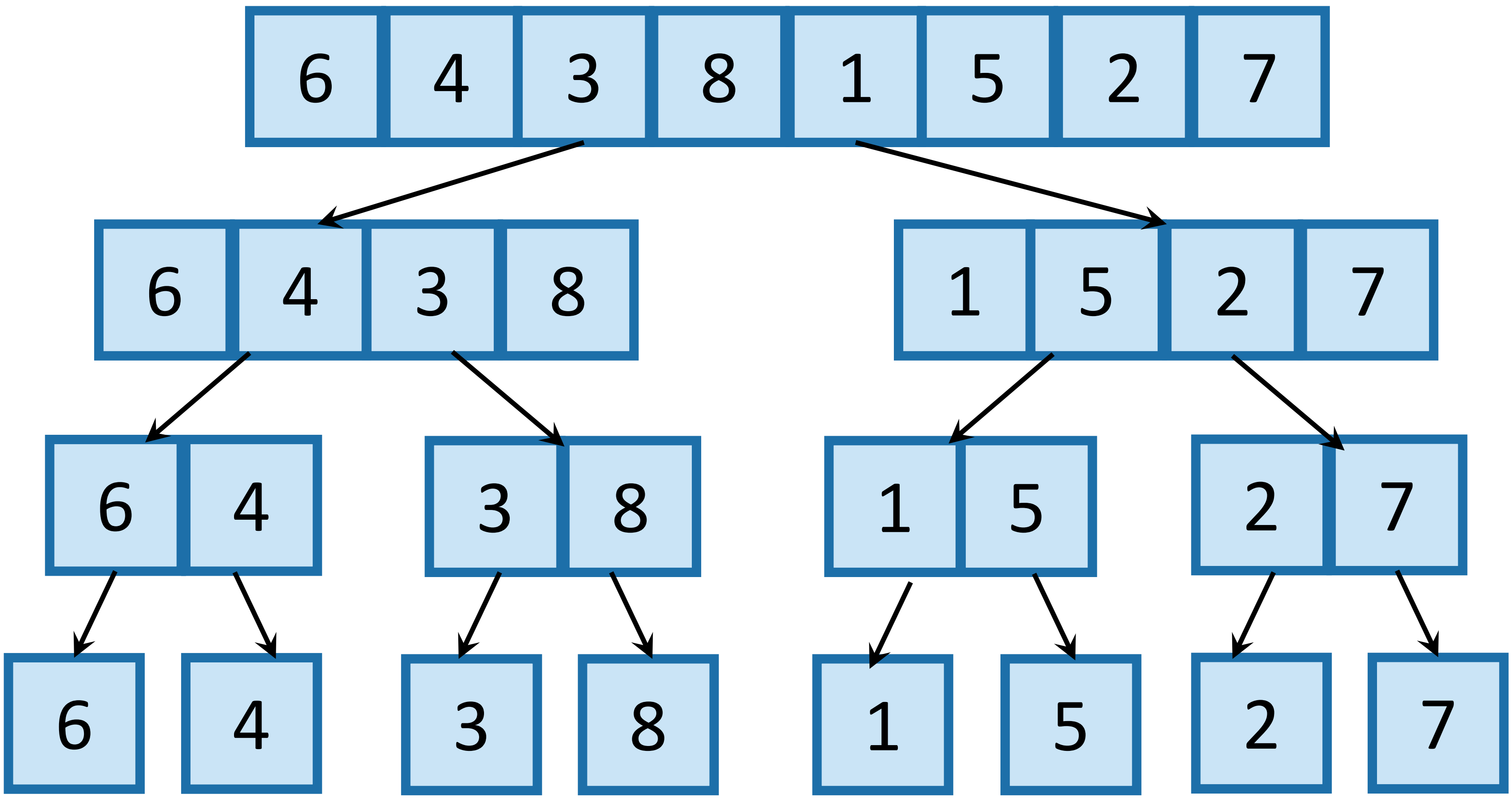
Seřaď pravou polovinu

- **return** MERGE(L,R)

Proveď sloučení obou polovin

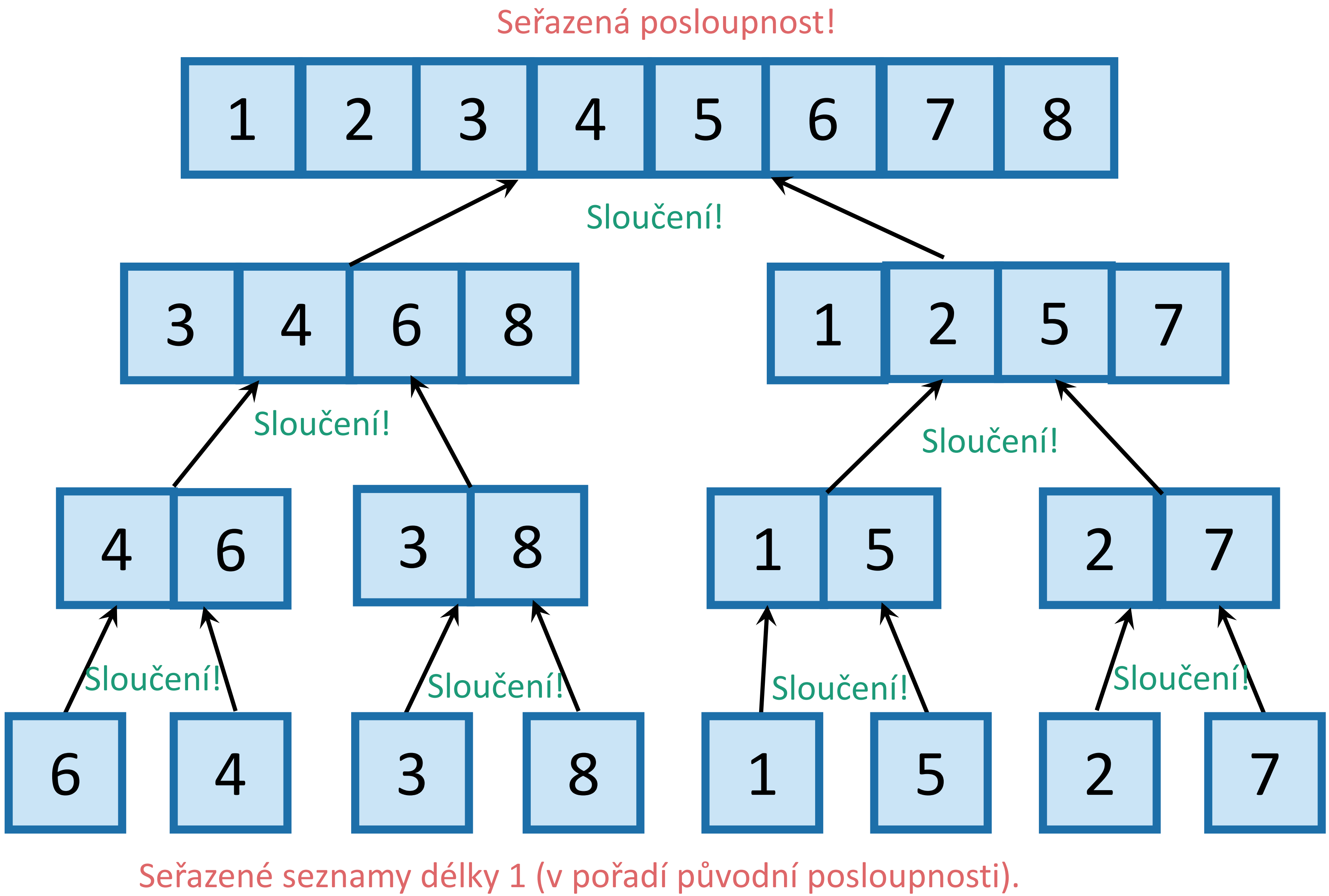
Co se přitom skutečně děje?

Nejprve, rekurzivně rozdělíme pole cestou dolů vždy na poloviny



To pole délky 1 je seřazeno

Potom, proveď sloučení zpátky směrem nahoru!



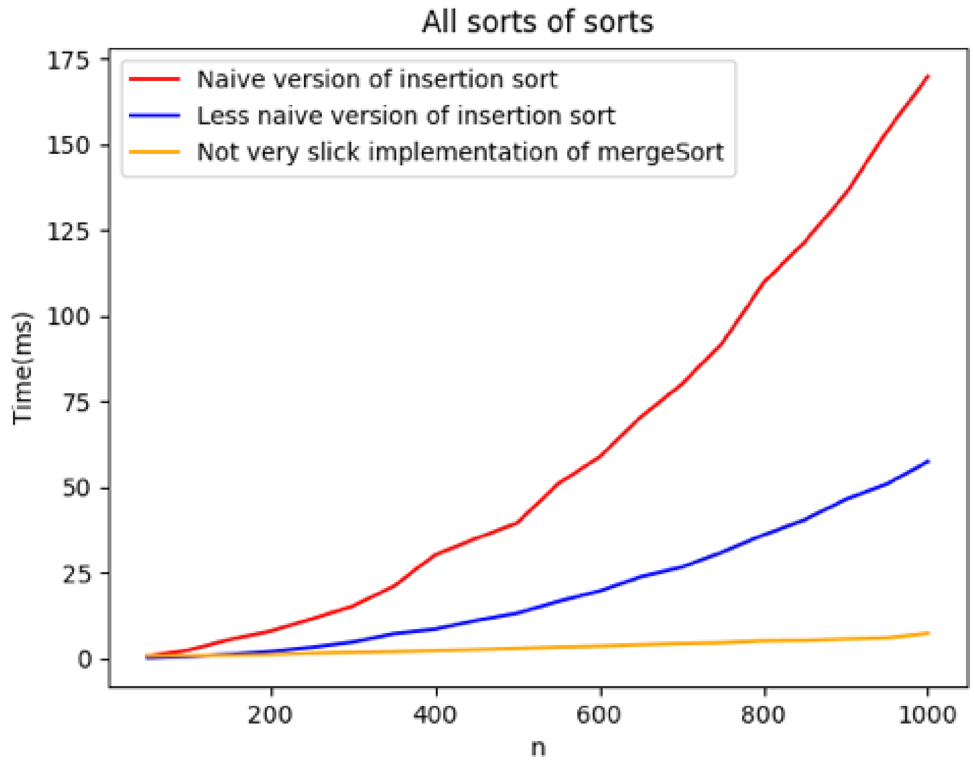
Dvě otázky

1. Funguje algoritmus správně?
2. Je algoritmus rychlý?

Empiricky:

1. Zdá se, že funguje správně.
2. Zdá se být rychlý.

Když bychom naprogramovali algoritmus a sledovali čas pro různá *n* …



Funguje algoritmus správně …

- Použijeme …

Důkaz indukcí!

Pracuje to?

Předpokládejme pro zjednodušení, že n je sudé.

- **Induktivní hypotéza:**

"V každém rekurzivním volání na poli délky nanejvýš i vrátí MERGESORT seřazené pole."

- **První krok (i=1):** 1-rozměrné pole je vždy seřazeno.
- **Induktivní krok: Potřeba ukázat**: pokud induktivní hypotéza platí pro k = i-1, potom platí také pro k=i.
- Tedy, potřebujeme ukázat, že pokud L a R jsou seřazena, potom MERGE(L,R) jsou také seřazena.
- **Závěr:** Ve vrcholu rekurzivního volání vrací MERGESORT seřazené pole.

- MERGESORT(A):
  - n = length(A)
  - **if** $n \leq 1$:
    - **return** A
  - L = MERGESORT(A[1 : n/2])
  - R = MERGESORT(A[n/2+1 : n])
  - **return** MERGE(L,R)

Zde jen ukázka, jak by se to dalo udělat, případně si to rozmyslete. Nebudeme se tím dál zabývat. Budeme se dále zabývat rychlostí algoritmu.

Zkuste se zamyslet a navrhněte indukční krok!

TIP: Budete muset dokázat, že algoritmus MERGE je správný, pro který možná budete potřebovat ... další důkaz indukcí!

Předpokládejme, že n je mocnina 2 pro zjednodušení.

Jak je rychlý?

Předpoklad:

Merge sort – doba běhu je $O(n\log(n))$

- Důkaz provedeme analýzou stromu rekurze.
- Ale nejprve, jak je to ve srovnání s Insert sortem?
  - Připomeňme, Insert sort běží v čase $O(n^2)$.

Připomenutí:

Všechny logaritmy, které budeme uvádět, mají základ 2

Rychlé připomenutí logaritmů

- Z definice: log(n) je číslo takové, že platí $2^{\log(n)} = n$.
- Intuitivně: log(n) je kolikrát musíme dělit n číslem 2, abychom se dostali k 1.

$32, \underbrace{16, 8, 4, 2, 1}_{\text{5-krát na polovinu}} \Rightarrow$ log(32) = 5

$64, \underbrace{32, 16, 8, 4, 2, 1}_{\text{6-krát na polovinu}} \Rightarrow$ log(64) = 6

log(128) = 7
log(256) = 8
log(512) = 9
....
log(**# částic ve vesmíru**) < 280

- log(n) roste velmi pomalu!

$O(n \log\ n)$ vs. $O(n^2)$?

- $\log(n)$ roste mnohem pomaleji než n
- $n \log(n)$ roste mnohem pomaleji než n^2

Závěr. Doba běhu O(n log n) je mnohem lepší než O(n^2)!

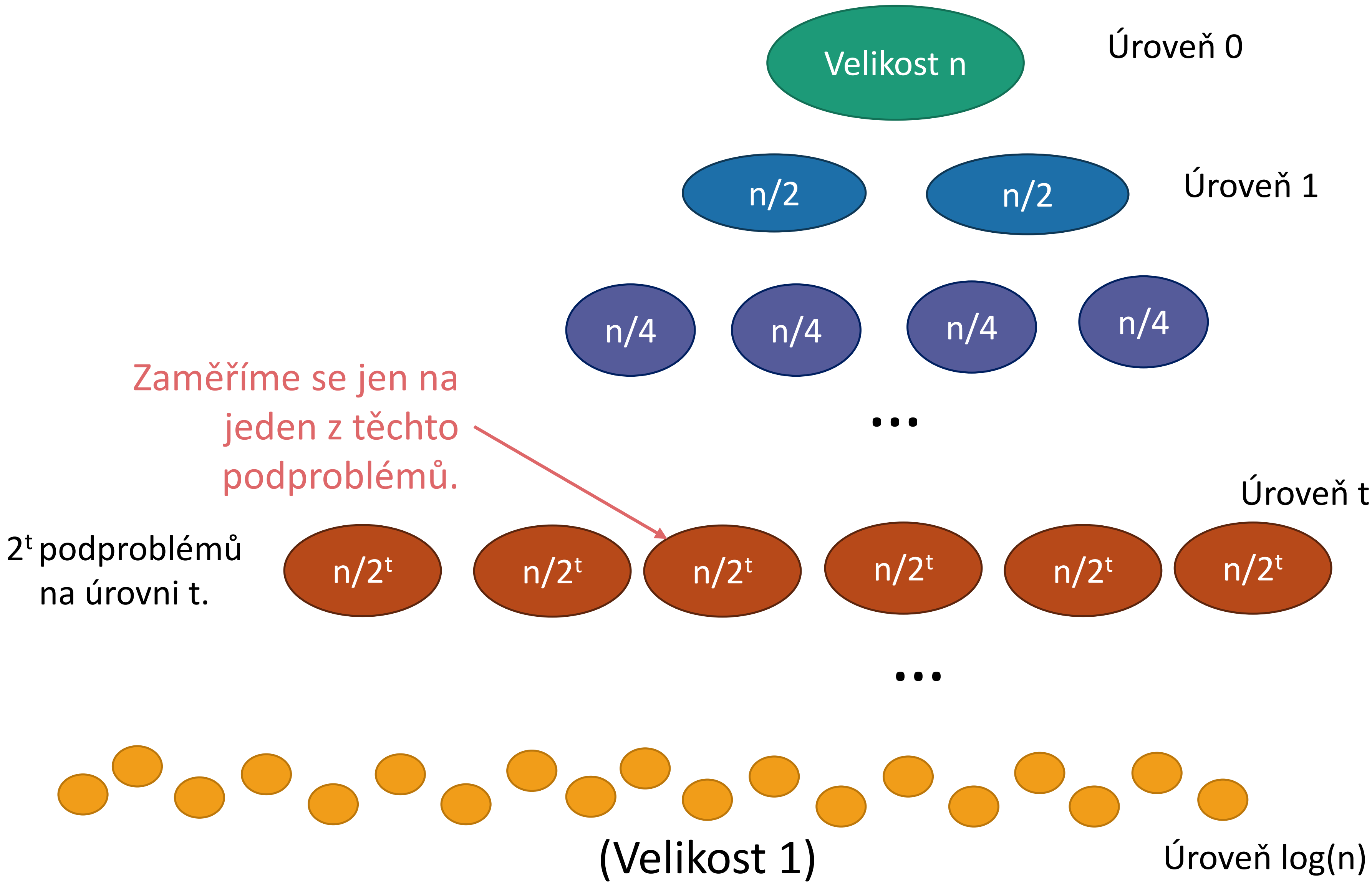
Předpokládámě pro zjednodušení, že n je sudé.

Nyní dokážeme náš předpoklad

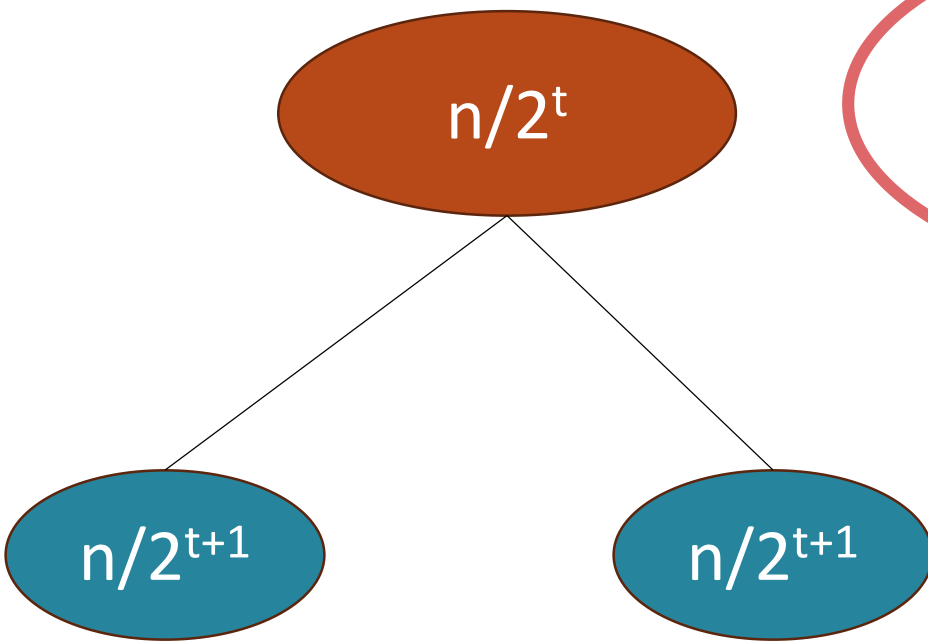
Předpoklad:

MergeSort běží v čase $O(n\log(n))$

Dokážeme předpoklad



Kolik práce (počet operací) je v tomto dílčím problému?



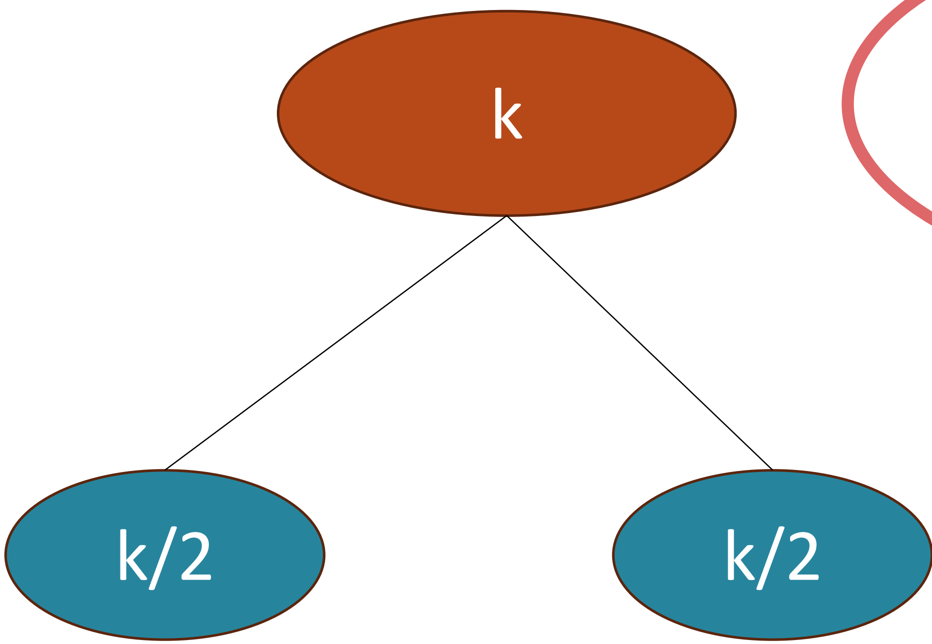
Čas strávený sloučením (MERGE) dvou dílčích problémů

+

Čas strávený v rámci dvou dílčích problémů

Kolik práce (počet operací) je v tomto dílčím problému?

Nechť $k=n/2^t$...

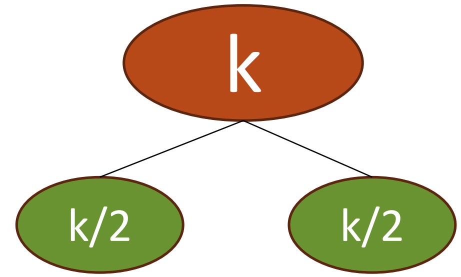


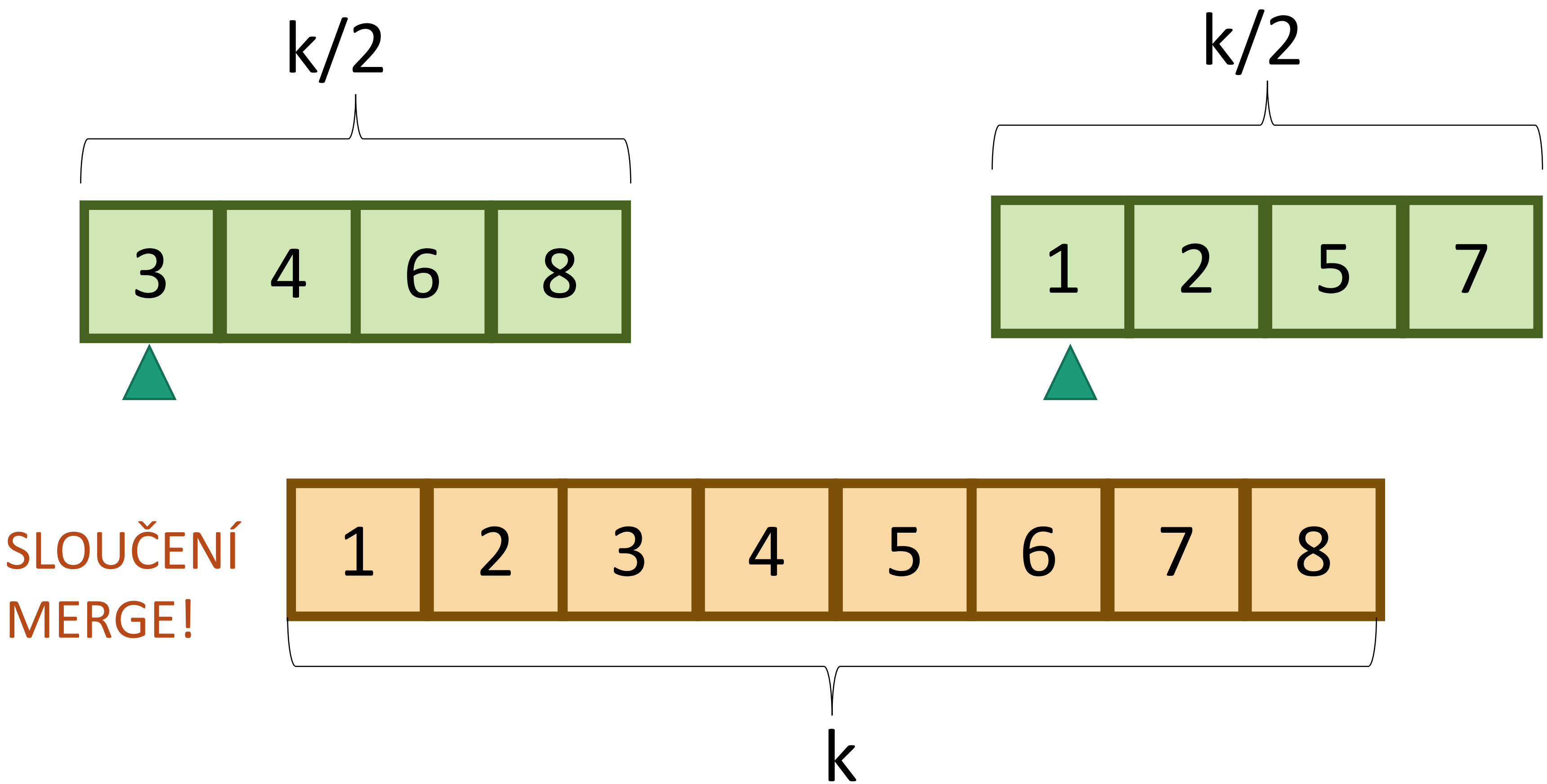
Čas strávený sloučením (MERGE) dvou dílčích problémů

+

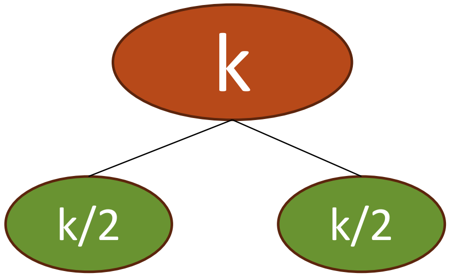
Čas strávený v rámci dvou dílčích problémů

Jak dlouho trvá operace SLOUČENÍ? (MERGE?)





Jak dlouho trvá operace SLOUČENÍ? (MERGE?)

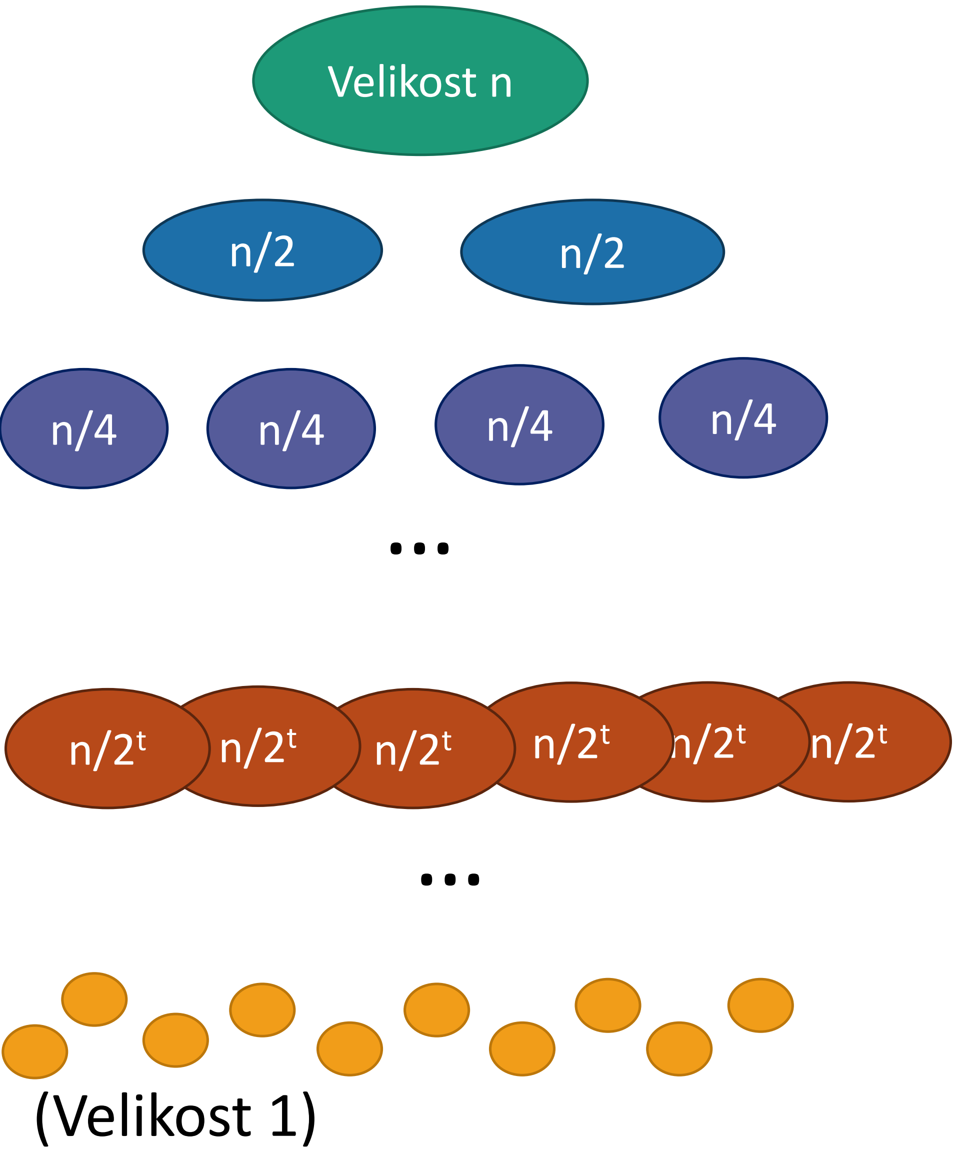


Jak dlouho trvá běh operace MERGE (SLOUČENÍ) na dvou seznamech (polích) o velikosti k/2?

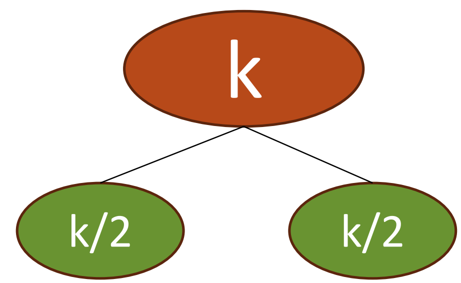
Zkuste odhadnout …

Odpověď: Trvá to čas O (k), protože procházíme seznamem jen jednou.

Strom rekurze



Počet operací v tomto uzlu je O(k).



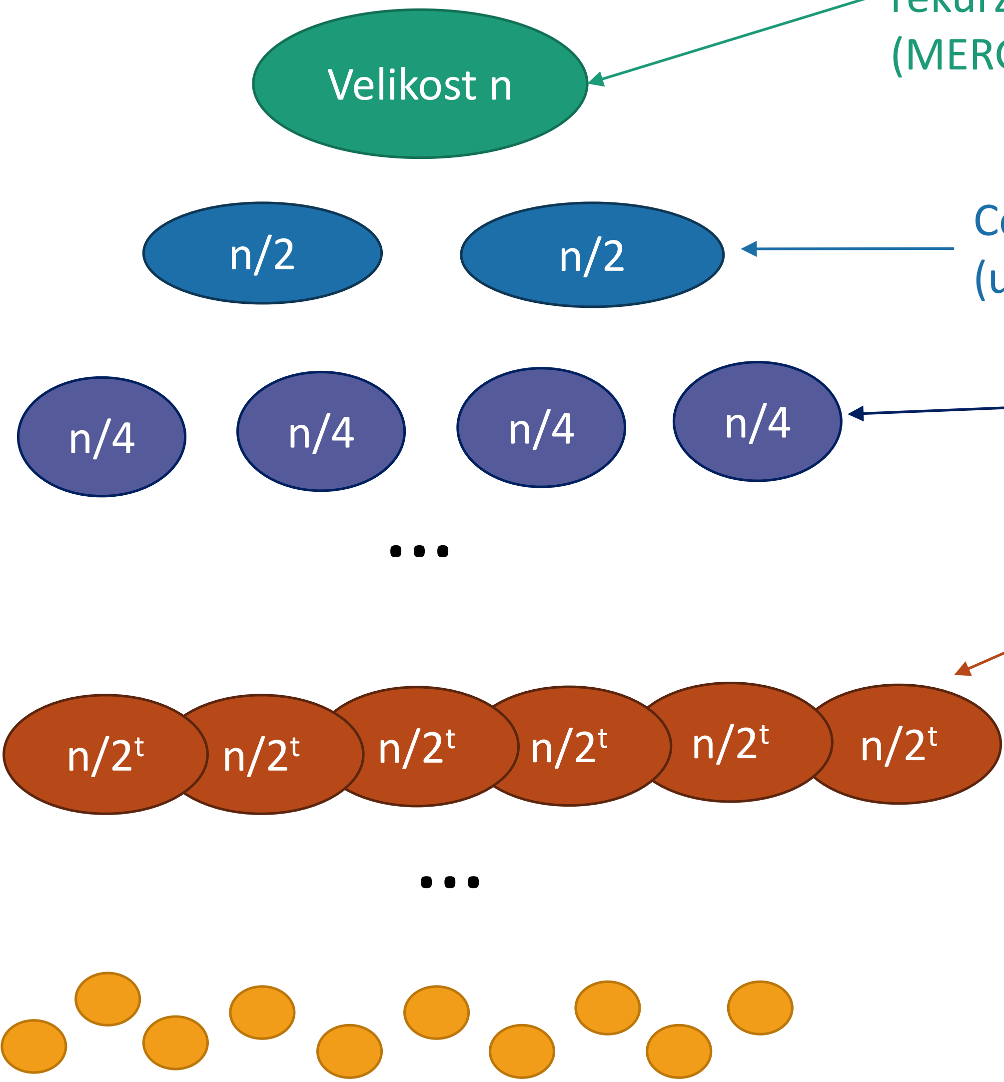
Strom rekurze

Kolik operací se provede na této úrovni rekurzivního stromu? (provede se operace sloučení (MERGE) podproblémů).

Co na této úrovni rekurzivního stromu? (u obou problémů velikosti n/2)

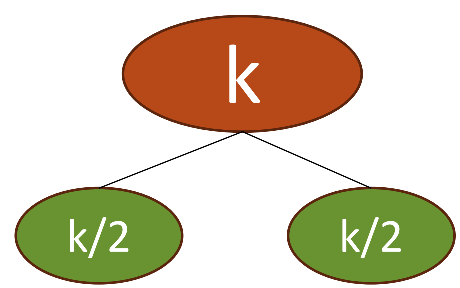
Na této úrovni?

Tato úroveň?



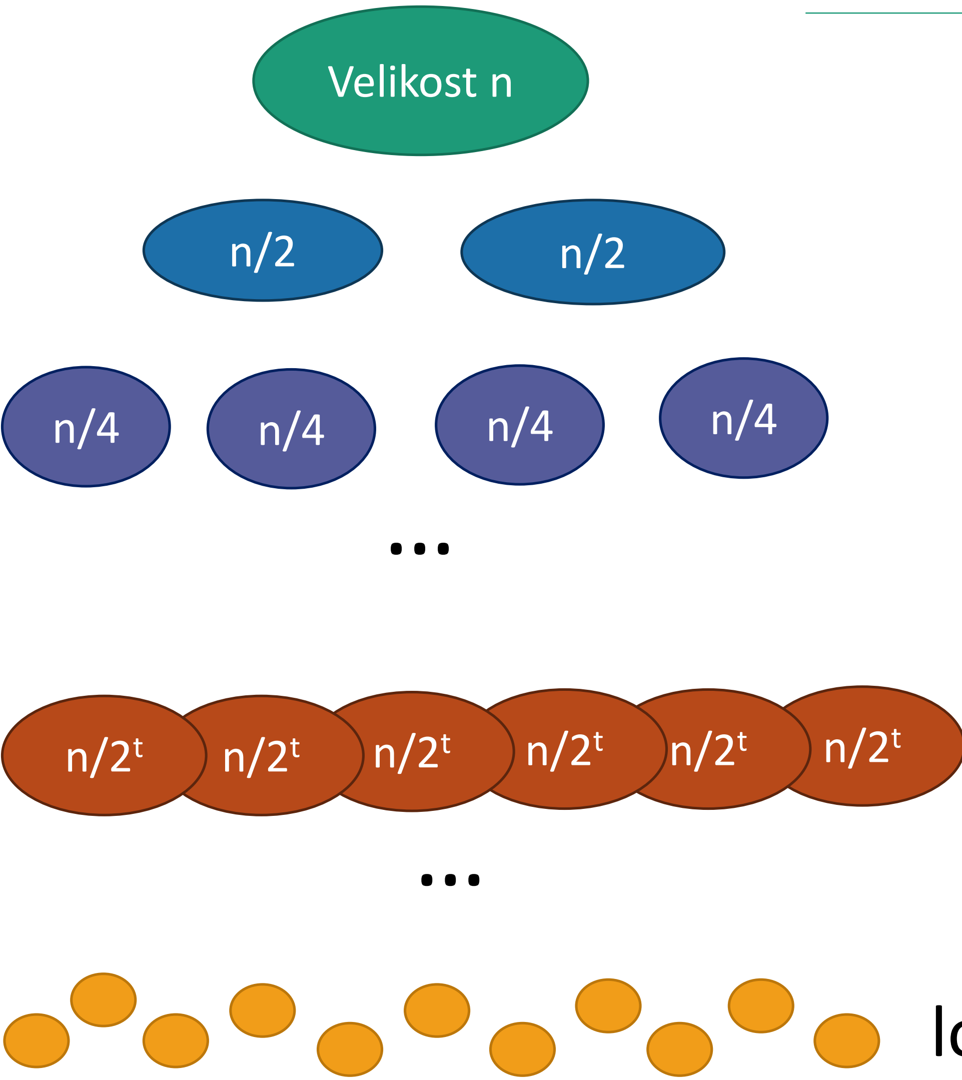
(Velikost 1)

Provede se O(k) operací u tohoto uzlu.



Strom rekurze

Vyzkoušejte sami



(Velikost 1)

Úroveň	# problemů	Velikost každého problému	Množství „práce“ na této úrovni
0	1	n	$O(n)$
1	2	$n/2$	$O(n)$
2	4	$n/4$	$O(n)$
…			
t	2^t	$n/2^t$	$O(n)$
…			
$\log(n)$	n	1	$O(n)$

Celková doba běhu…

- O(n) kroků na každé úrovni
- Počet úrovní log(n) + 1 levels
- Celkově tedy: O(n log(n))

- Merge Sort seřadí posloupnost n celých čísel v čase O(n log(n)).
- Algoritmus Merge Sort je (asymptoticky) lepší než Insertion Sort

Závěry:

- Insertion Sort (a Selection Sort, Bubble Sort) běží v čase $O(n^2)$ – jeho časová složitost je $O(n^2)$
- MergeSort je algoritmus typu rozděl-a-panuj, který běží v čase $O(n \log(n))$ - jeho časová složitost je $O(n \log(n)$ (asymptotická časová složitost)

Závěry:

- Jak můžeme dokázat, že algoritmus je korektní?
  - Jednou z možností je důkaz indukcí
- Jak měříme dobu běhu algoritmu?
  - Analýza nejhoršího případu
  - Asymptotická analýza doby běhu
- Jak můžeme analyzovat dobu běhu rekurzivního algoritmu?
  - Jeden ze způsobů je nakreslit strom rekurze.

Složitost rekurzivních algoritmů

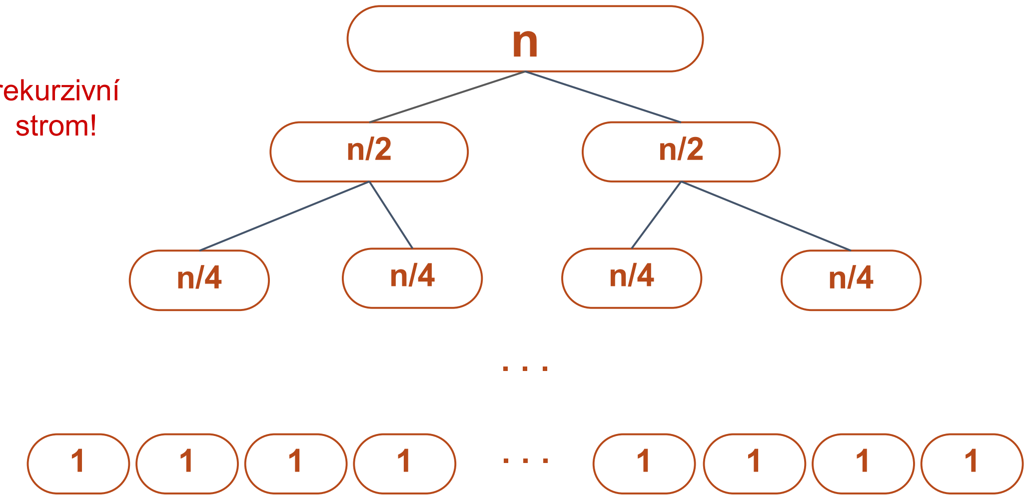
Rekurentní vztah

Nástroj pro popis doby běhu rekurzivních algoritmů

Rekurentní vztahy

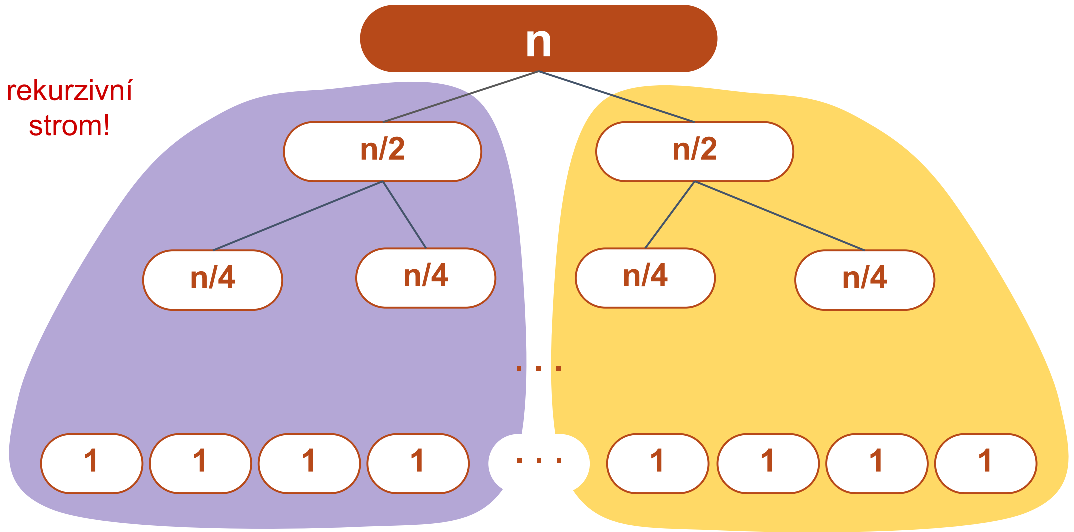
Abychom vytvořili rekurentní tvar pro Merge Sort, můžeme o jeho době běhu uvažovat následovně :

rekurzivní strom!



Rekurentní vztahy

Abychom vytvořili rekurentní tvar pro Merge Sort, můžeme o jeho době běhu uvažovat následovně :



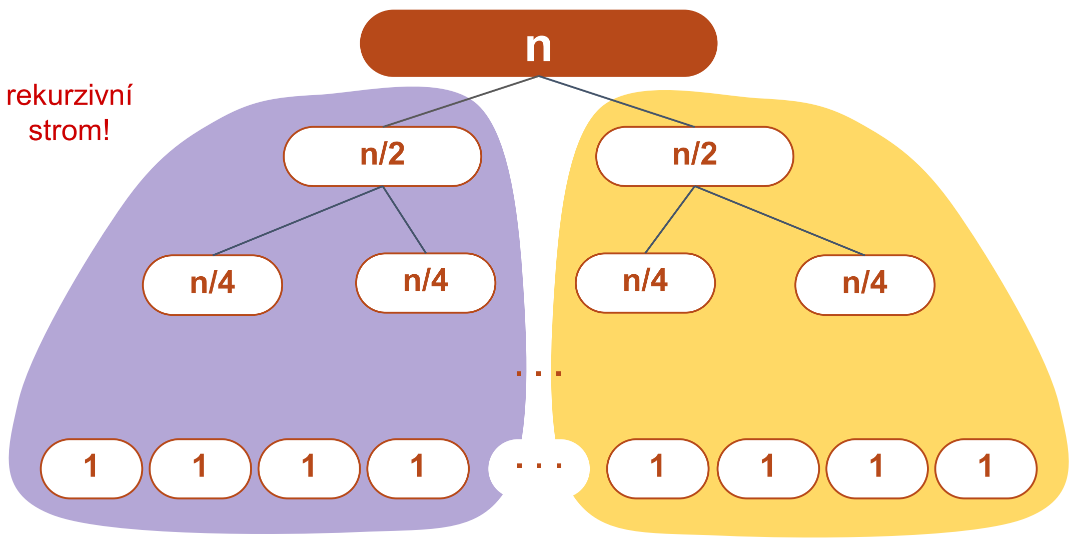
Práce v celém stromě =

celková práce v LEVÉM rekurzivním volání (levý podstrom)

+

Rekurentní vztahy

Abychom vytvořili rekurentní tvar pro Merge Sort, můžeme o jeho době běhu uvažovat následovně :



Práce v celém stromě

=

celková práce v LEVÉM rekurzivním volání (levý podstrom)

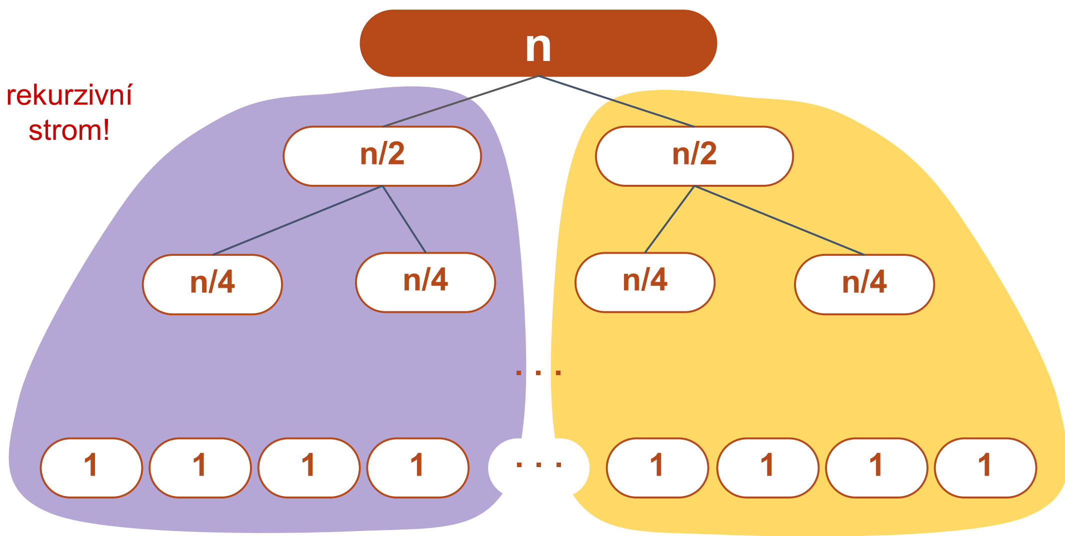
+

celková práce v PRAVÉM rekurzivním volání (pravý podstrom)

+

Rekurentní vztahy

Abychom vytvořili rekurentní tvar pro Merge Sort, můžeme o jeho době běhu uvažovat následovně :



Práce v celém stromě

=

celková práce v LEVÉM rekurzivním volání (levý podstrom)

+

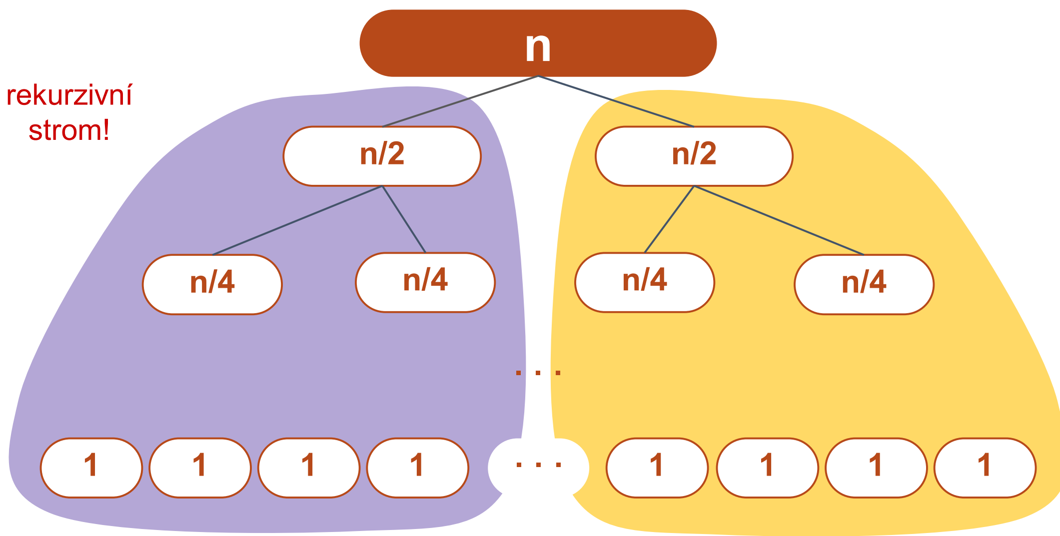
celková práce v PRAVÉM rekurzivním volání (pravý podstrom)

+

Práce odvedená v rámci nejvyššího problému

Rekurentní vztahy

Abychom vytvořili rekurentní tvar pro Merge Sort, můžeme o jeho době běhu uvažovat následovně :



Práce v celém stromě

=

celková práce v LEVÉM rekurzivním volání (levý podstrom)

+

celková práce v PRAVÉM rekurzivním volání (pravý podstrom)

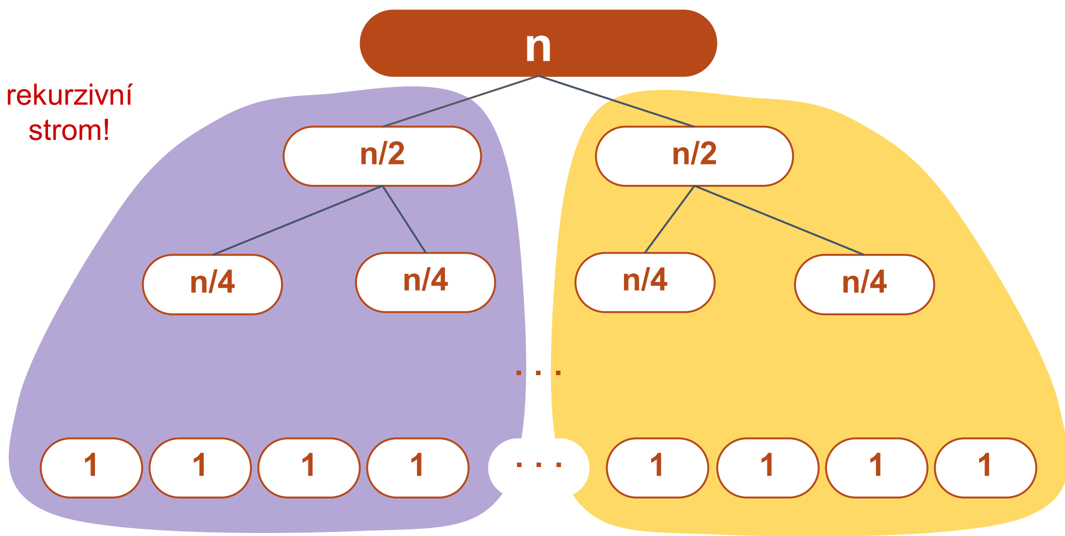
+

Práce odvedená v rámci nejvyššího problému

pracovat na vytváření subproblemů a „slučování“ jejich řešení

Rekurentní vztahy

Abychom vytvořili rekurentní tvar pro Merge Sort, můžeme o jeho době běhu uvažovat následovně:



T(n) =

T(n/2)

+

T(n/2)

+

O(n)

Rekurentní vztahy

Abychom vytvořili rekurentní tvar pro Merge Sort, můžeme o jeho době běhu uvažovat následovně :

Strom
rekurze

1

Poznámka:

Děláme zde zjednodušující předpoklad, že **n** je dokonalá mocnina dvou (jinak bychom měli používat horní celou část a dolní celou část).

Ukazuje se, že pokud začleníme horní celou část a dolní celou část, stále získáváme podproblémy konstantní velikosti na úrovni $\lfloor \log_b n \rfloor$ a obecně věci, které budeme dělat v této třídě se vztahy s opakováním, budou stále fungovat, pokud zde zapomeneme na horní celou část a dolní celou část.

T(n) =

T(n/2)

+

T(n/2)

+

O(n)

Rekurentní vztahy

Abychom vytvořili rekurentní tvar pro Merge Sort, můžeme o jeho době běhu uvažovat následovně :

$$\mathbf{T(n)} = \underbrace{\boxed{T(n/2)} + \boxed{T(n/2)}}_{} + \boxed{\mathbf{O(n)}}$$

*protože dílčí problémy mají stejnou velikost, můžeme to také zapsat jako **2 · T(n/2)***

Toto je rekurzivní definice pro T(n), takže potřebujeme také ZÁKLADNÍ PŘÍPAD:

$$\mathbf{T(1) = O(1)}$$

Bez ohledu na to, co je T, T(1) = O(1). Pokud by byla větší než O(1), velikost problému by ve skutečnosti nebyla 1.

Protože jsme již použili strom rekurze k výpočtu běhu programu Merge Sort, víme že **T(n) = O(n log n)**.

Příklad rekurentních vztahů

Násobení metodou rozděl a panuj (v 1. přednášce)

$$T(n) = 4 \cdot T(n/2) + O(n)$$

$$T(n) = O(n^{\log_2 4}) = \mathbf{O(n^2)}$$

Celočíselné násobení podle Karatsuby (v 1. přednášce)

$$T(n) = 3 \cdot T(n/2) + O(n)$$

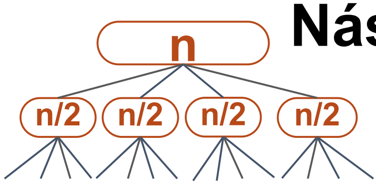
$$T(n) = O(n^{\log_2 3}) \approx \mathbf{O(n^{1.6})}$$

MergeSort

$$T(n) = 2 \cdot T(n/2) + O(n)$$

$$T(n) = \mathbf{O(n \log n)}$$

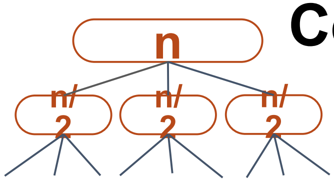
Příklad rekurentních vztahů



Násobení metodou rozděl a panuj (v 1. přednášce)

$$T(n) = 4 \cdot T(n/2) + O(n)$$

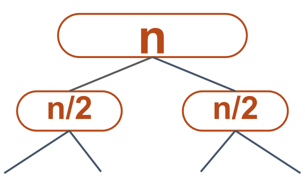
$$T(n) = O(n^{\log_2 4}) = \mathbf{O(n^2)}$$



Celočíselné násobení podle Karatsuby (v 1. přednášce)

$$T(n) = 3 \cdot T(n/2) + O(n)$$

$$T(n) = O(n^{\log_2 3}) \approx \mathbf{O(n^{1.6})}$$

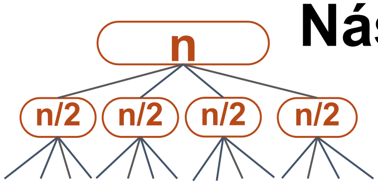


MergeSort

$$T(n) = 2 \cdot T(n/2) + O(n)$$

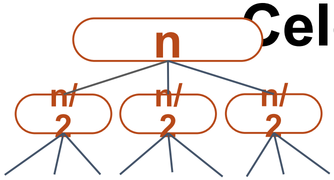
$$T(n) = \mathbf{O(n \log n)}$$

Příklad rekurentních vztahů



Násobení metodou rozděl a panuj (v 1. přednášce)

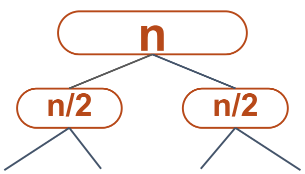
$$T(n) = 4 \cdot T(n/2) + O(n)$$
$$T(n) = O(n^{\log_2 4}) = \mathbf{O(n^2)}$$



Celočíselné násobení podle Karatsuby (v

$$T(n) = 3 \cdot T(n/2) + O(n)$$
$$T(n) = O(n^{\log_2 3}) \approx \mathbf{O(n^{1.6})}$$

MergeSort



$$T(n) = 2 \cdot T(n/2) + O(n)$$
$$T(n) = \mathbf{O(n \log n)}$$

Vidíme nějaký

VZOR???

Vyjádření složitosti rekurzivního algoritmu rekurentním tvarem

- Příklad vyjádření složitosti rekurzivního algoritmu rekurencí:

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 , \\ T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + \Theta(n) & \text{if } n > 1 . \end{cases}$$

  Kde $T(n)$ je celková složitost algoritmu.

  Na pravé straně jsou jednotlivé případy složitostí pro různá n.

- Okrajové případy (pro n < konstanta) můžeme opomenout, protože mají konstantní asymptotickou složitost. Zaokrouhlení rovněž většinou neovlivní celkový výsledek (Pozor existují i výjimky!).

  Z toho dostáváme rekurentní vztah:

$$T(n) = 2T(n/2) + \Theta(n) ,$$

Převod rekurence na přímé vyjádření

- Přímým vyjádřením složitosti myslíme vyjádření složitosti bez rekurence.
  - Např: $T(n) = \Theta(\log(n))$

- Jaké jsou možnosti řešení?
  - **Substituční metoda**
    - „Uhádneme“ řešení a potom dokážeme, že je správné indukcí.
  - **Metoda rekurzívního stromu**
    - Spočítáme složitost celého rekurzivního stromu.
  - **Použití „kuchařky“** (Master theorem – mistrovská věta)
    - Pro některé speciální tvary rekurentních vztahů známe předem vypočítané řešení dle mistrovské věty.

Substituční metoda

- Řešíme ve dvou krocích
  1. **Odhadneme přesný tvar řešení.**
     - Odhad lze stanovit například pomocí zjišťováním složitosti pro různá vstupní n.
  2. **Matematicky dokážeme, že je náš odhad správný.**
     - Obvykle se dokazuje pomocí matematické indukce.
- Metoda bývá zpravidla velmi účinná.
- Její nevýhodou je určování přesného tvaru řešení v kroku 1 pro které neexistuje obecný postup.

Substituční metoda - příklad

- Příklad:

  $T(n) = 2T(n/2) + n$

- Předpokládejme, že jsme odhadli přímé vyjádření vztahem:

  $T(n) = O(n \log(n))$

- Z definice horního odhadu O, chceme tedy dokázat, že

  $T(n) \leq cn \log(n)$

  pro nějaké vhodné $c > 0$.

- Nyní stanovíme vhodný indukční předpoklad (tj. nechť odhad platí pro $n/2$):

  $T(n/2) \leq c(n/2) \log(n/2)$

Substituční metoda - příklad

- Nyní dosadíme indukční předpoklad do rekurentního vztahu a pokusíme se dokázat jeho platnost vyjádřením přímého (nerekurentního) původně odhadnutého vztahu pro n.

$$\begin{aligned} T(n) &\leq 2(c\,(n/2)\log(n/2)) + n \\ &\leq cn\log(n/2) + n \\ &= cn\log(n) - cn\log(2) + n \\ &= cn\log(n) - cn + n \\ &\leq cn\log(n) \end{aligned}$$

  kde poslední krok platí pro $c \geq 1$.

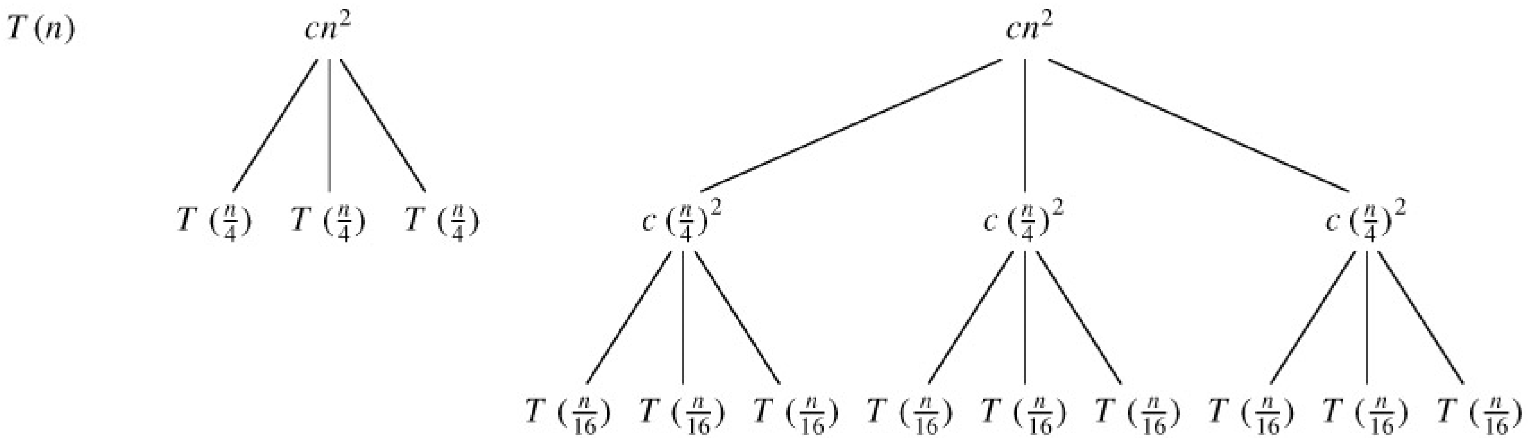
- Počáteční krok indukce platí triviálně. Díky asymptotické notaci stačí ukázat, že odhad platí pro nějaké n_0 a $c > 0$. V našem příkladě tedy platí pro $n_0=3$ a $c \geq 2$).
- Tím je důkaz hotov.

Metoda rekurzívního stromu - příklad

- Příklad:

  $T(n) = 3T(n/4) + cn^2$

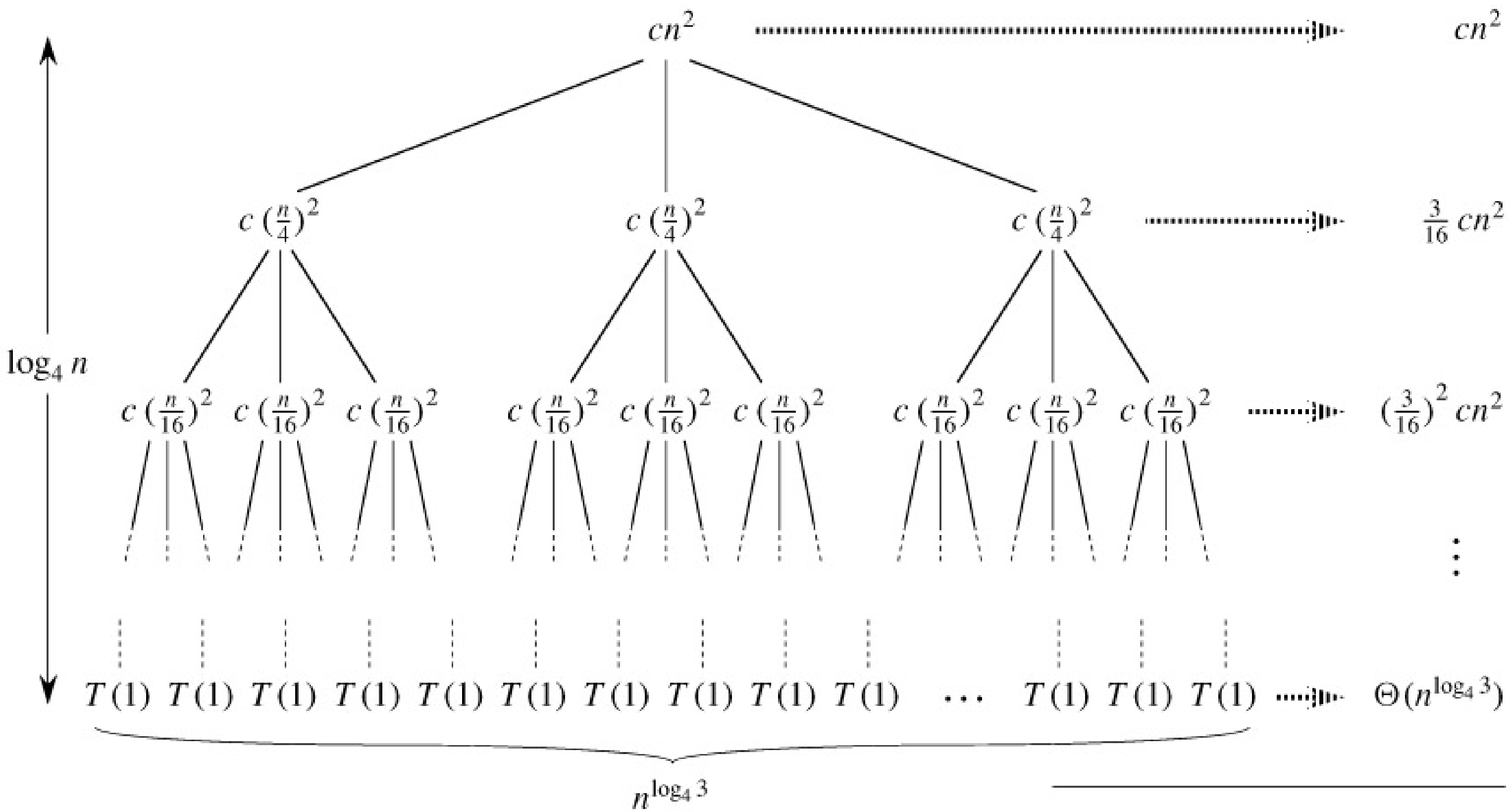
- Iterativně rozkládáme do rekurzivních stromů:



- Pro každý strom platí, že součet všech uzlů dá složitost $T(n)$ podle původního rekurentního vztahu.
- Rekurzivní stromy jsou pouze grafická vizualizace rozvoje rekurentního vztahu.

Metoda rekurzívního stromu - příklad

- Výsledný strom má následující tvar:



- Vyjádříme součty jednotlivých pater stromu.
- Všechna patra sečteme a dostaneme výslednou složitost:

$O\,(n^2)$

Metoda rekurzívního stromu - příklad

- Součet pater lze spočítat následovně:

$$
\begin{aligned}
T(n) &= cn^2 + \frac{3}{16}cn^2 + \left(\frac{3}{16}\right)^2 cn^2 + \cdots + \left(\frac{3}{16}\right)^{\log_4 n - 1} cn^2 + \Theta(n^{\log_4 3}) \\
&= \sum_{i=0}^{\log_4 n - 1} \left(\frac{3}{16}\right)^i cn^2 + \Theta(n^{\log_4 3}) \\
&< \sum_{i=0}^{\infty} \left(\frac{3}{16}\right)^i cn^2 + \Theta(n^{\log_4 3}) \\
&= \frac{1}{1-(3/16)} cn^2 + \Theta(n^{\log_4 3}) \\
&= \frac{16}{13} cn^2 + \Theta(n^{\log_4 3}) \\
&= O(n^2) .
\end{aligned}
$$

podle vzorce $\sum_{k=0}^{\infty} kx^k = \frac{x}{(1-x)^2}$

pro $|x| < 1$

Použití „kuchařky“

- Použití „kuchařky“ nebo tzv. mistrovské věty (master theorem) řeší rekurentní složitost, která má následující tvar:

$$T(n) = a\,T(n/b) + f(n)$$

Kde $a \geq 1$ a $b > 1$ jsou konstanty

a $f(n)$ je asymptoticky kladná funkce.

- Zaokrouhlení u členu $T(n/b)$ na $T(\lfloor n/b \rfloor)$ nebo $T(\lceil n/b \rceil)$ neovlivní v tomto případě výslednou složitost.

Použití „kuchařky“

- **Master theorem** (mistrovská nebo také kuchařková věta)
  - Nechť jsou $a \geq 1$ a $b > 1$ konstanty, nechť je $f(n)$ funkce a nechť $T(n)$ je definováno pro nezáporná celá čísla rekurencí

$$T(n) = a\,T(n/b) + f(n)$$

kde n/b má význam buď $\lceil n/b \rceil$ nebo $\lfloor n/b \rfloor$. Potom lze asymptoticky vyjádřit následovně:

1. Pokud $f(n) \in O(n^{\log_b(a)-\varepsilon})$ pro nějakou konstantu $\varepsilon > 0$, potom

$$\boldsymbol{T(n) \in \Theta(n^{\log_b(a)})}.$$

2. Pokud $f(n) \in \Theta(n^{\log_b(a)})$, potom

$$\boldsymbol{T(n) \in \Theta(n^{\log_b(a)} \log(n))}.$$

3. Pokud $f(n) \in \Omega(n^{\log_b(a)+\varepsilon})$ pro nějakou konstantu $\varepsilon > 0$ a pokud $a\,f(n/b) \leq c\,f(n)$ pro nějakou konstantu $c < 1$ a všechna dostatečně velké n, potom

$$\boldsymbol{T(n) \in \Theta(f(n))}.$$

Použití „kuchařky“ – příklad 1

- Příklad 1:

  $T(n) = 9T(n/3) + n$
- Z toho dostáváme, že $a = 9$, $b = 3$, $f(n) = n \in \mathrm{O}(n^{\log_3(9)-1})$.

  Jedná se tedy o případ číslo 1.
- Dostáváme tedy složitost:

  $T(n) \in \mathbf{O}(n^{\log_3(9)}) = \mathbf{O}(\boldsymbol{n^2})$

Použití „kuchařky“ – příklad 2

- Příklad 2:

  $T(n) = T(2n/3) + 1$

- Z toho dostáváme, že $a = 1$, $b = 3/2$,

  $f(n) = 1 = n^{\log_{3/2}(1)} \in \mathbf{\Theta}(n^{\log_{3/2}(1)})$.

  Jedná se tedy o případ číslo 2.

- Dostáváme tedy složitost:

  $T(n) \in \mathbf{O}(n^{\log_{3/2}(1)} \log(n)) = \mathbf{O}(\mathbf{log}(\boldsymbol{n}))$

Použití „kuchařky“ – příklad 3

- Příklad 3:

  $T(n) = 3T(n/4) + n\log(n)$

- Z toho dostáváme, že $a = 3$, $b = 4$,

  $f(n) = n\log(n)$ a víme, že $n^{\log_4(3)} = O(n^{0.793})$.

  Platí tedy, že $f(n) \in \Omega(n^{\log_4(3)+0.2})$.

  Pokud by se mělo jednat o případ 3 musí ještě platit pro $c < 1$ a všechna dostatečně velká n, že $a\,f(n/b) \le c\,f(n)$ tedy $a\,f(n/b) = 3(n/4)\log(n/4) \le (3/4)n\log(n) = c\,f(n)$ pro $c = 3/4$.

- Dostáváme tedy složitost:

  $T(n) \in$ $\mathbf{O(n\log(n))}$

Q & A