# Různé algoritmy mají různou složitost

**Algoritmus a program není totéž**

# Složitost rekurzivních algoritmů

## Metody

- stromu rekurze
- substituční
- mistrovská

# Složitost rekurzivních algoritmů

**T(n) -- asymptotická složitost algoritmu při vstupu o velikosti n**

**Př.: Merge sort**

**Asymptotická složitost vyřešení triviální úlohy**

$$T(n) = \begin{cases} \Theta(1) & \text{pro } n = 1 \\ 2T(n/2) + \Theta(n) & \text{pro } n > 1 \end{cases}$$

**Jak asymptotická složitost při vstupu o velikosti n závisí na asymptotické složitosti při vstupu o velikosti n/2**

**Složitost rozdělení problému a spojení dílčích řešení (polovin pole v Merge sortu)**

# Složitost rekurzivních algoritmů

## Co lze zanedbat

Typickou hodnotu n si vhodně zvolíme
(v Merge sortu mocninu 2)

Konkrétní konstanta neovlivní výslednou asymptotickou složitost

$$T(n) = \begin{cases} \Theta(1) & \text{pro } n = 1 \\ 2T(n/2) + \Theta(n) & \text{pro } n > 1 \end{cases}$$

n/2 a obecně n/konst není celé číslo, mysleme si však, že (víceméně) je,
a použijme jej místo správného $\lceil n/2 \rceil$ či $\lfloor n/2 \rfloor$ apod.
Většinou výsledek nebude ovlivněn.

# Složitost rekurzivních algoritmů

## Příklad

**Pro algoritmus A platí**

$$T(n) = \begin{cases} \Theta(1) & \text{pro } n = 1 \\ 3T(\lfloor n/4 \rfloor) + \Theta(n^2) & \text{pro } n > 1 \end{cases}$$

**Rozdělí data na čtvrtiny. Vyřešení „čtvrtinové“ úlohy trvá $T(\lfloor n/4 \rfloor)$.**

**Jedna čtvrtina se nezpracovává*) ⇒ tři čtvrtiny se zpracují v čase $3T(\lfloor n/4 \rfloor)$.**

*) z důvodů zde nepodstatných :-)

**Čas potřebný na rozdělení na čtvrtiny a na spojení „čtvrtinových“ řešení je $\Theta(n^2)$.**

# Složitost rekurzivních algoritmů

## Příklad

$$T(n) = \begin{cases} \Theta(1) & \text{pro } n = 1 \\ 3T(\lfloor n/4 \rfloor) + \Theta(n^2) & \text{pro } n > 1 \end{cases}$$

Zanedbáme celé části,
$\Theta$ nahradíme úměrou

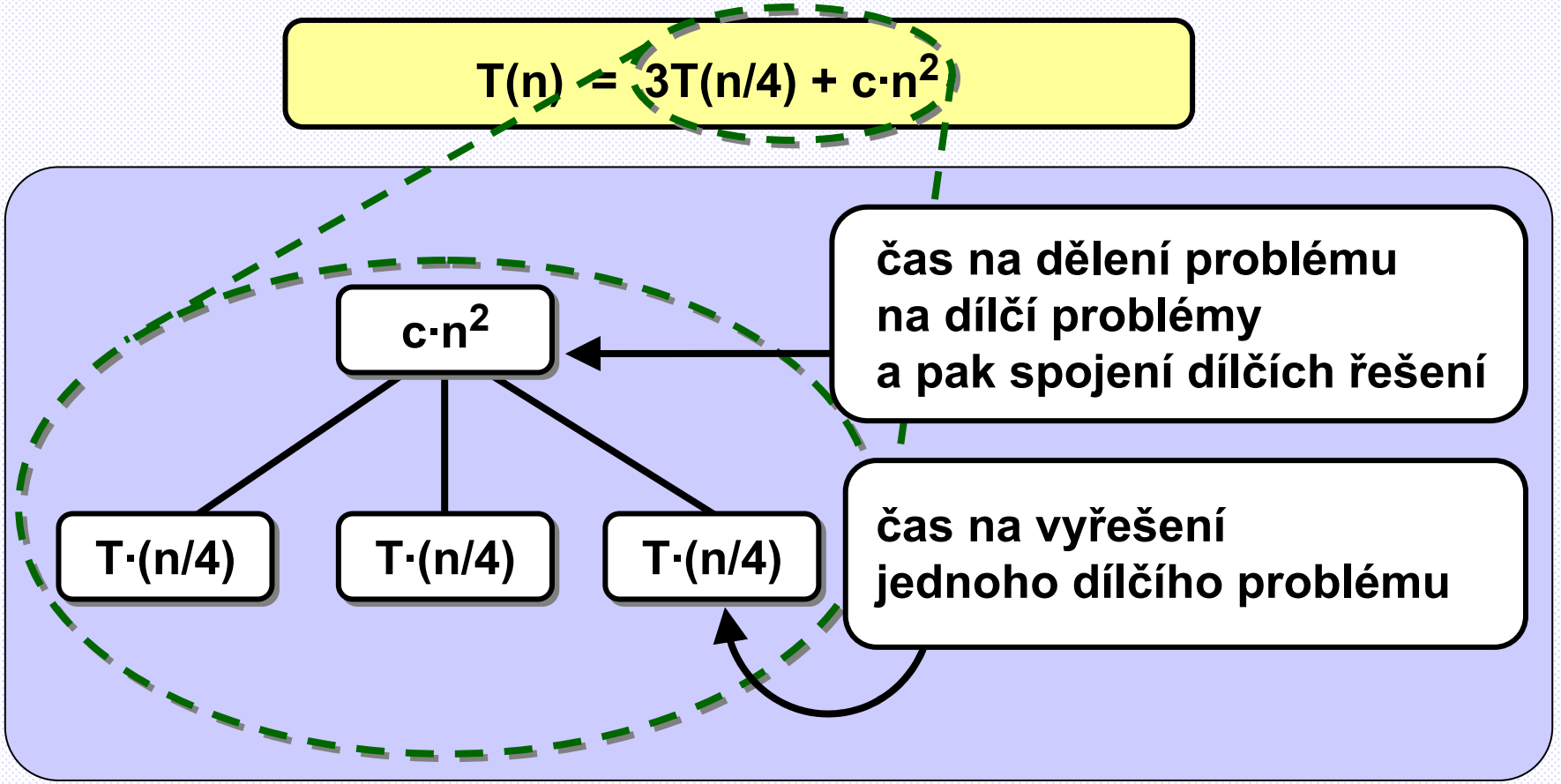
Vztah pro výpočet

$$T(n) = 3T(n/4) + c \cdot n^2$$

# Strom rekurze

## Metoda stromu rekurze

# Strom rekurze

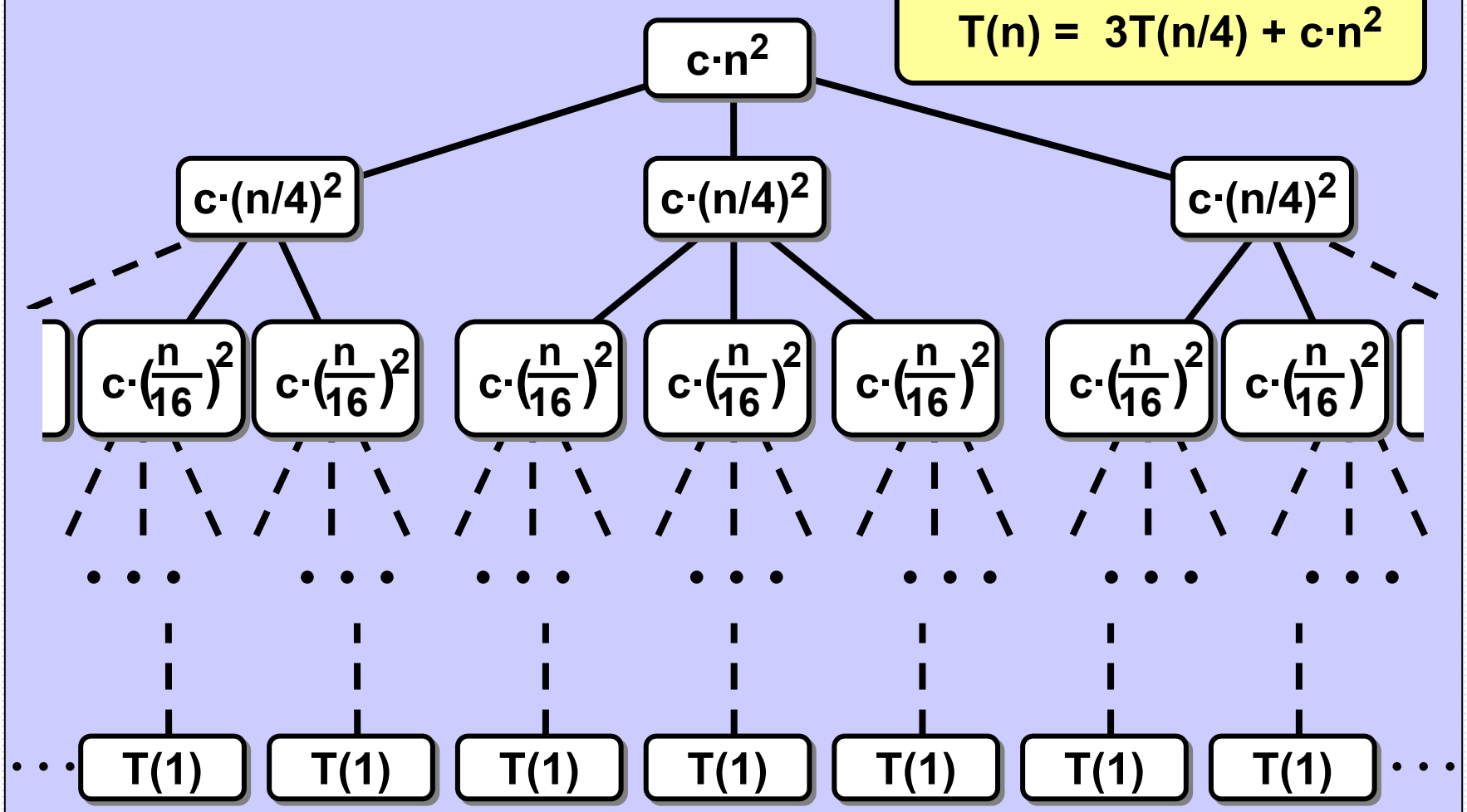



Strom rekurze je ale větší ...

# Strom rekurze

## Strom rekurze

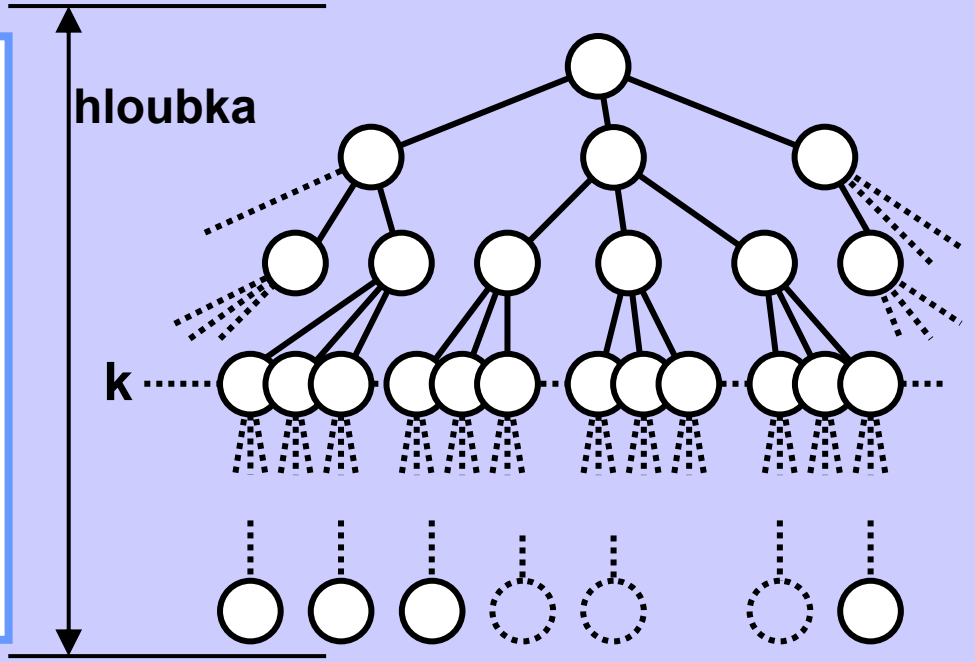
$T(n) = 3T(n/4) + c \cdot n^2$



# Strom rekurze

**Průběh výpočtu**

1. Nakresli strom rekurze
2. Spočti jeho hloubku
3. Spočti jeho šířku v patře k
4. Spočti cenu uzlu v patře k
5. Sečti ceny uzlů v patře k
6. Sečti ceny všech pater




**cena uzlu** = asymptotická složitost zpracování podproblému odpovídajícího uzlu ve stromu rekurze.

**cena stromu** = asymptotická složitost zpracování celé úlohy.

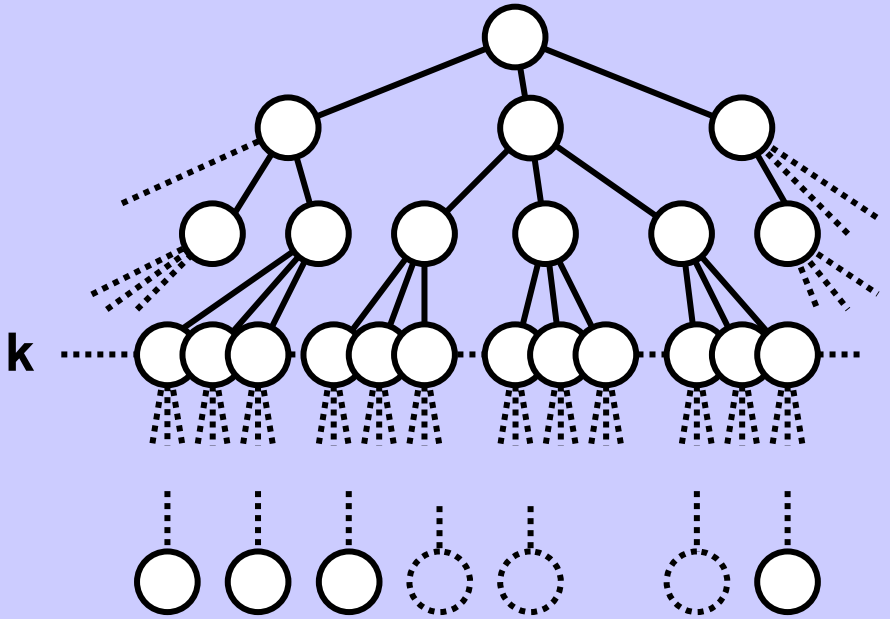
# Strom rekurze

$$T(n) = 3T(n/4) + c \cdot n^2$$

**Průběh výpočtu**

1. Nakresli strom rekurze
2. Spočti jeho hloubku

...




V hloubce k
je velikost podproblému $n/4^k$ .

Velikost podproblému je tedy = 1, když $n/4^k = 1$, tj $k = \log_4(n)$.

**hloubka stromu**

Takže **strom má $\log_4(n) + 1$ pater** (k=0 je hloubka kořene).

# Strom rekurze

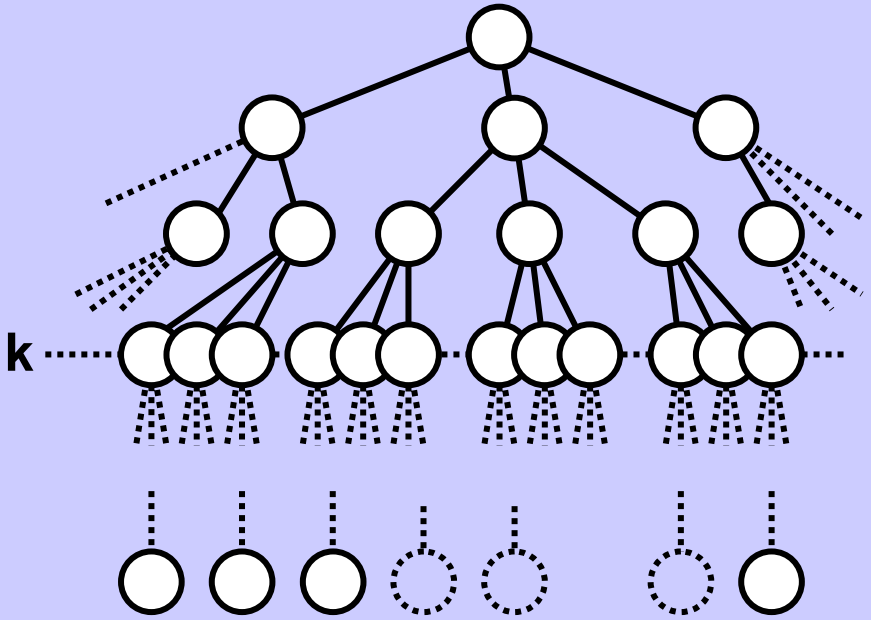
$T(n) = 3T(n/4) + c \cdot n^2$

**Průběh výpočtu**

...

3. Spočti jeho šířku v patře k

...




0. patro – 1 uzel
1. patro – 3 uzly
2. patro – 3·3 = 9 uzlů
3. patro – 3·3·3 = 27 uzlů

...

**počet uzlů v jednom patře**

k. patro – $3 \cdot 3 \cdot \ldots \cdot 3 \cdot 3 = 3^k$ uzlů

# Strom rekurze

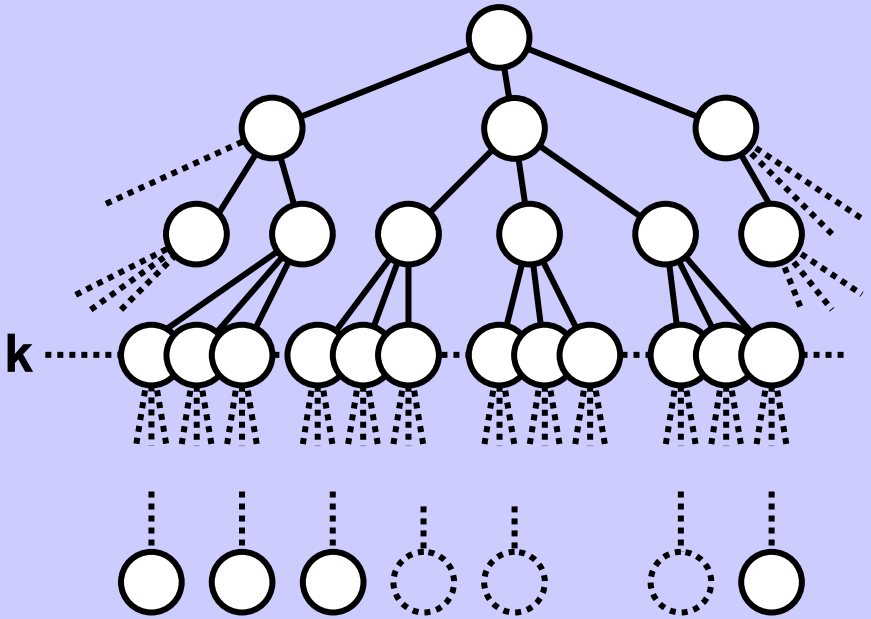
$T(n) = 3T(n/4) + c \cdot n^2$

**Průběh výpočtu**

...

4. Spočti cenu uzlu v patře k

...




0. patro – $c \cdot n^2$
1. patro – $c \cdot (n/4)^2$
2. patro – $c \cdot (n/16)^2$
3. patro – $c \cdot (n/64)^2$

...

**cena uzlu v jednom patře**

k. patro – $c \cdot (n/4^k)^2$

# Strom rekurze

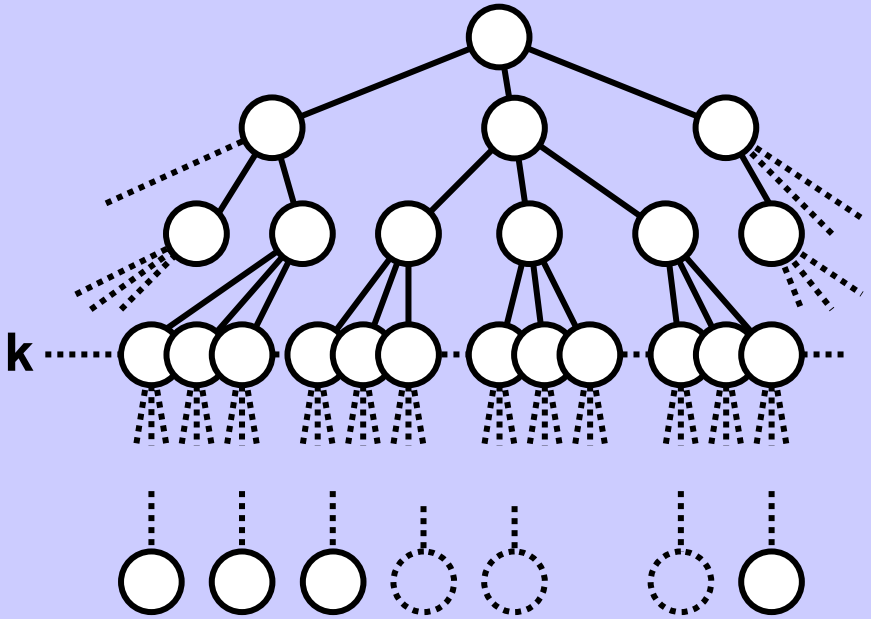
$T(n) = 3T(n/4) + c \cdot n^2$

**Průběh výpočtu**

...

5. Sečti ceny uzlů v patře k

...




**V patře k je $3^k$ uzlů, každý má cenu $c \cdot (n/4^k)^2$.**

**celková cena patra**

$3^k \cdot c \cdot (n/4^k)^2 = (3/16)^k \cdot c \cdot n^2$

**Pozor na poslední patro:**

# Strom rekurze

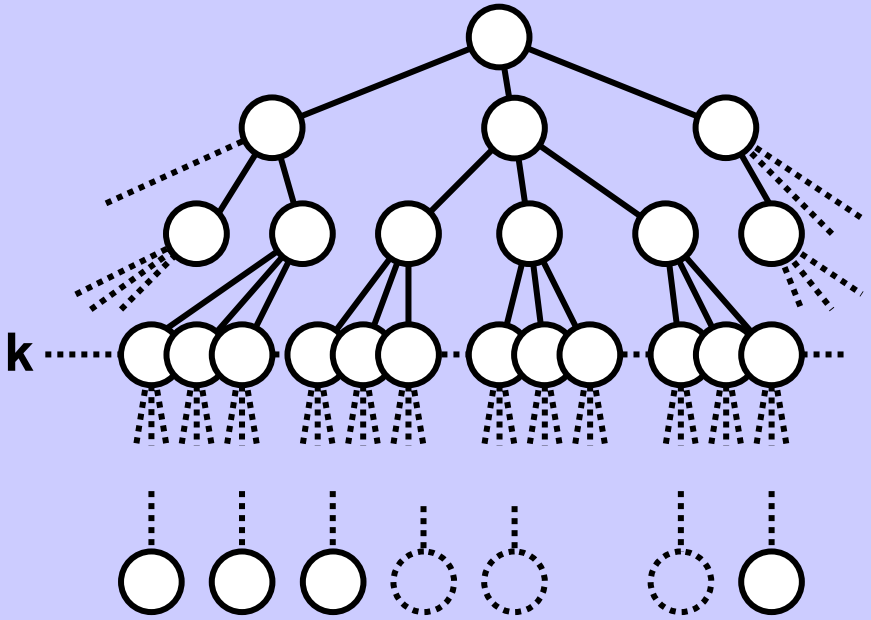
$T(n) = 3T(n/4) + c \cdot n^2$

**Průběh výpočtu**

...

5. Sečti ceny uzlů v patře k

...




Poslední patro je v hloubce $\log_4(n)$ a má tedy

$3^{\log_4(n)} = n^{\log_4(3)}$ uzlů.

## cena posledního patra

Každý přispívá uzel v posledním patře přispívá konstantní cenou, takže cena posledního patra je

$n^{\log_4(3)} \cdot konst = \Theta(n^{\log_4(3)})$

# Strom rekurze

**Průběh výpočtu**
**...**
**6. Sečti ceny všech pater**

$$T(n) = 3T(n/4) + c \cdot n^2$$

**Celková cena =**

$$cn^2 + 3/16 \cdot cn^2 + (3/16)^2 \cdot cn^2 + \ldots + (3/16)^{\log_4(n-1)} \cdot cn^2 + \Theta\left(n^{\log_4(3)}\right) =$$

$$\underbrace{(1 + 3/16 \cdot + (3/16)^2 + \ldots + (3/16)^{\log_4(n-1)}}) \cdot cn^2 + \Theta\left(n^{\log_4(3)}\right) \doteq$$

**Geometrickou posloupnost nahradíme přibližně geometrickou řadou (zbytek řady je zanedbatelný). Získáváme horní odhad součtu.**

$$(1 + 3/16 \cdot + (3/16)^2 + (3/16)^3 + \ldots \text{ ad inf. }) \cdot cn^2 + \Theta\left(n^{\log_4(3)}\right) =$$

$$(1 / (1 - 3/16)) \cdot cn^2 + \Theta\left(n^{\log_4(3)}\right) = 16/13 \cdot cn^2 + \Theta\left(n^{\log_4(3)}\right) =$$

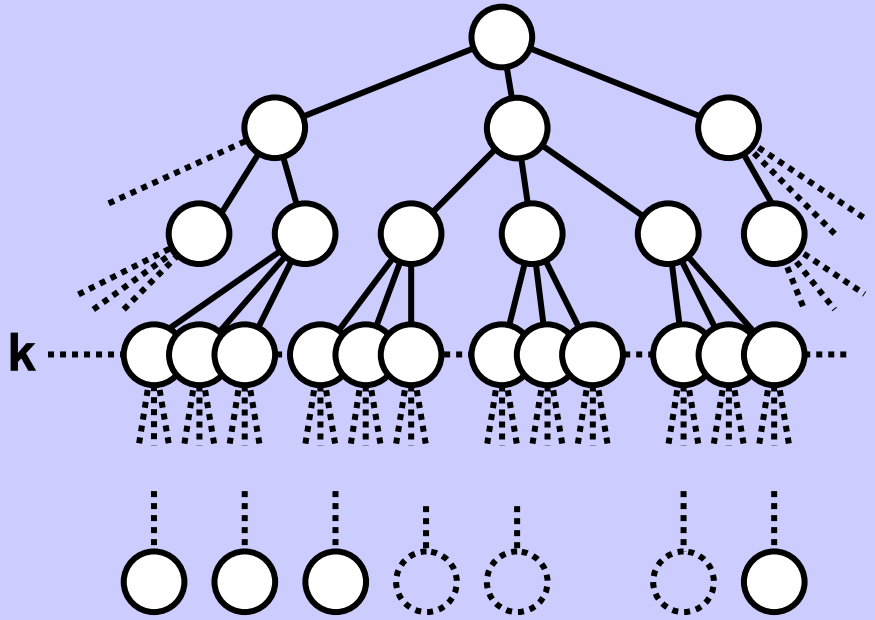
# Strom rekurze

**Průběh výpočtu**

**...**

**6. Sečti ceny všech pater**

$T(n) = 3T(n/4) + c \cdot n^2$




**Asymptotická složitost algoritmu A**

$2 > \log_4(3) \Rightarrow$

$= 16/13 \cdot cn^2 + \Theta(n^{\log_4(3)}) = \Theta(n^2)$

# Substituční metoda

Substituční metoda

# Substituční metoda

## Příklad

Rekurentní vztah popisující asymptotickou složitost algoritmu B

$$T(n) = 2T(\lfloor n/2 \rfloor) + n$$

Náš odhad složitosti

$$T(n) = O(n \cdot \log_2(n))$$

Zdroj odhadu: zkušenost, podobnost s jinými úlohami
úvaha, intuice ... :-)

Chceme dokázat: Náš odhad platí

Metoda: Běžná matematická indukce, do níž dosadíme (substituujeme) daný rekurentní vztah

# Substituční metoda

Chceme dokázat : $T(n) = O(n \cdot \log_2(n))$,
to jest : $T(n) \leq c \cdot n \cdot \log_2(n)$, pro vhodné $c > 0$

## Krok II (obecný krok) matematické indukce:

Dokážeme, že pokud nerovnost platí pro $\lfloor n/2 \rfloor$, platí i pro n.

Víme: $T(n) = 2T(\lfloor n/2 \rfloor) + n$

Předpokládáme: $T(\lfloor n/2 \rfloor) \leq c \cdot \lfloor n/2 \rfloor \cdot \log_2(\lfloor n/2 \rfloor)$

Substituce: $T(n) \leq 2 \cdot c \cdot \lfloor n/2 \rfloor \cdot \log_2(\lfloor n/2 \rfloor) + n$

úpravy: $\leq cn \cdot \log_2(n/2) + n = cn \cdot \log_2(n) - cn \cdot \log_2(2) + n$

$= cn \cdot \log_2(n) - cn + n$

$\leq cn \cdot \log_2(n)$, pokud $c \geq 1$

# Substituční metoda

## Krok I matematické indukce

Nerovnost $T(n) \leq c \cdot n \cdot \log_2(n)$, platí pro nějaké konkrétní malé n.

## Potíž

**Nelze dokazovat pro n = 1,**

neboť bychom dokazovali $T(1) \leq c \cdot 1 \cdot \log_2(1) = 0$, což neplatí, protože jistě je $T(1) > 0$.

## Pozorování

$T(n) = 2T(\lfloor n/2 \rfloor) + n$

Pokud n > 3, v rekurentním vztahu se T(1) neobjeví,
tedy pokud dokážeme indukční krok I pro n = 2 a n = 3,
je důkaz hotov pro všechna $n \geq 2$.

## Řešení

Jde nám ale o **asymptotickou složitost,** tudíž důkaz pro $n \geq 2$ stačí.

# Substituční metoda

## Krok I matematické indukce pro n = 2 a n = 3

At' $T(1) = k$.

$$T(n) = 2T(\lfloor n/2 \rfloor) + n$$

Z rekurentního vztahu plyne
$T(2) = 2k+2$
$T(3) = 2k+3$

Chceme mít:
$T(2) \leq c \cdot 2 \cdot \log_2(2)$
$T(3) \leq c \cdot 3 \cdot \log_2(3)$

Stačí tedy volit $c \geq \max \{ (2k+2)/2,\ (2k+3)/(3 \cdot \log_2(3)) \}$.

Tudíž, kdyby k bylo např. 10, stačilo by volit
$c \geq \max \{ (2 \cdot 10 + 2)/2,\ (2 \cdot 10 + 3)/(3 \cdot \log_2(3)) \} = \max \{11, 4.84 \} = 11$.

# Mistrovská metoda

## Mistrovská metoda

## (The master method)

# Mistrovská metoda

## Věta

Nechť $a \geq 1$ a $b > 1$ jsou konstanty, $f(n)$ je funkce a nechť asymptotická složitost daného algoritmu je definována rekurentním vztahem

$$T(n) = aT(n/b) + f(n),$$

kde podíl $n/b$ lze libovolně interpretovat jako $\lfloor n/b \rfloor$ nebo $\lceil n/b \rceil$.
Potom platí

1. Pokud $f(n) = O(n^{\log_b(a) - e})$ pro nějakou konstantu $e > 0$, pak
   $$T(n) = \Theta(n^{\log_b(a)}).$$
2. Pokud $f(n) = \Theta(n^{\log_b(a)})$, pak
   $$T(n) = \Theta(n^{\log_b(a)} \cdot \log_2(n)).$$
3. Pokud $f(n) = \Omega(n^{\log_b(a) + e})$ pro nějakou konstantu $e > 0$, a pokud $a \cdot f(n/b) \leq c \cdot f(n)$ pro pro nějaké $c < 0$ a pro všechna dostatečně velká $n$, pak
   $$T(n) = \Theta(f(n)).$$

# Mistrovská metoda

## Komentáře k podmínkám věty

1. Pokud $f(n) = O(n^{\log_b(a) - e})$ pro nějakou konstantu $e > 0$, pak

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

// Podmínka 1. říká, že f(n) musí růst polynomiálně pomaleji
// než funkce $n^{\log_b(a)}$ .

3. Pokud $f(n) = \Omega(n^{\log_b(a) + e})$ pro nějakou konstantu $e > 0$,
a pokud $a \cdot f(n/b) \leq c \cdot f(n)$ pro pro nějaké $c < 0$
a pro všechna dostatečně velká n, pak

$$T(n) = \Theta(f(n)).$$

// Podmínka 3. říká, že f(n) musí růst polynomiálně rychleji
// než funkce $n^{\log_b(a)}$ .

# Mistrovská metoda

...

1. Pokud $f(n) = O(n^{\log_b(a) - e})$ pro nějakou konstantu $e > 0$, pak

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

...

**Příklad.**

$T(n) = 9T(n/3) + n$

$a = 9, b = 3, f(n) = n.$

Platí $n^{\log_b(a)} = n^{\log_3(9)} = \Theta(n^2)$

$f(n) = O(n^{\log_3(9) - e})$, kde $e = 1$

**Celkem tedy $T(n) = \Theta(n^2)$**

# Mistrovská metoda

...

3. Pokud $f(n) = \Omega(n^{\log_b(a)+e})$ pro nějakou konstantu $e > 0$, a pokud $a \cdot f(n/b) \leq c \cdot f(n)$ pro pro nějaké $c < 0$ a pro všechna dostatečně velká n, pak

$T(n) = \Theta(f(n))$.

**Příklad.**

$T(n) = 2T(n/2) + n \cdot \log_2(n)$

$a = 2$, $b = 2$, $f(n) = n \cdot \log_2(n)$ .

Platí $n^{\log_b(a)} = n$

$f(n) = n \cdot \log_2(n)$ roste asymptoticky rychleji než $n^{\log_b(a)} = n$.

Pozor, neroste ale polynomiálně rychleji. Poměr $n \cdot \log_2(n) / n = \log_2(n)$ roste pomaleji než každá rostoucí polynomiální funkce. $n \cdot \log2(n) \notin \Omega(n^{1+e})$ pro každé kladné e. Předpoklad 3. není splněn.

# Různé algoritmy mají různou složitost

**Algoritmus a program není totéž**