Kapitola 5

Složitost rekurentních algoritmů

Co si zde procvičíme?

- Seznámíme se se třemi základními metodami úpravy rekurentních vztahů do nerekurzivní podoby
  - substituční metoda
  - metoda rekurzivních stromů
  - mistrovská věta
- Ukážeme si, jak provádět odhady složitostí algoritmů a jak dokázat jejich správnost
- Seznámíme se s tím, jak rozkreslit posloupnost rekurzivních volání do rekurentního stromu a naučíme se vypočíst charakteristiky těchto stromů - jejich výšku, počet listů apod.
- Vysvětlíme si mistrovskou větu a naučíme se ji používat pro úpravu složitostí rekurentních algoritmů

5.1 Substituční metoda

Výpočet složitosti rekurzivních algoritmů bývá s ohledem na vnořená volání často obtížný. Z vlastního algoritmu dokážeme obvykle vyjádřit vztah pro jeho složitost rekurzivně, probléme ale bývá specifikace složitosti nerekurzivně, pomocí asymptotických mezí.

K tomu se využívají různé metody a postupy. Zde se postupně seznámíme se třemi nejpoužívanějšími:

- substituční metoda,
- metoda rekurzivního stromu,
- mistrovská věta - Master Theorem.

Substituční metoda řeší problém ve dvou krocích - nejprve provedeme přesný odhad, jehož správnost následně potvrdíme matematickými metodami, obvykle důkazem pomocí matematické indukce.

Substituční metoda může být velice efektivní, pokud dokážeme udělat dobrý odhad. Ten se obvykle počítá zjišťováním složitostí pro různá n a následným zobecněním výsledku.

Úloha 5.1.1. Uvažujme rekurzivní verzi `insertion_sort_rec` algoritmu řazení vkládáním. Určíme jeho asymptotickou složitost pomocí substituční metody.

```
1 def insertion_sort_rec(a, n):
2     if n > 0:
3         insertion_sort_rec(a, n - 1)
4         key = a[n]
5         i = n - 1
6         while i >= 0 and a[i] > key:
7             a[i + 1] = a[i]
8             i = i - 1
9         a[i + 1] = key
```

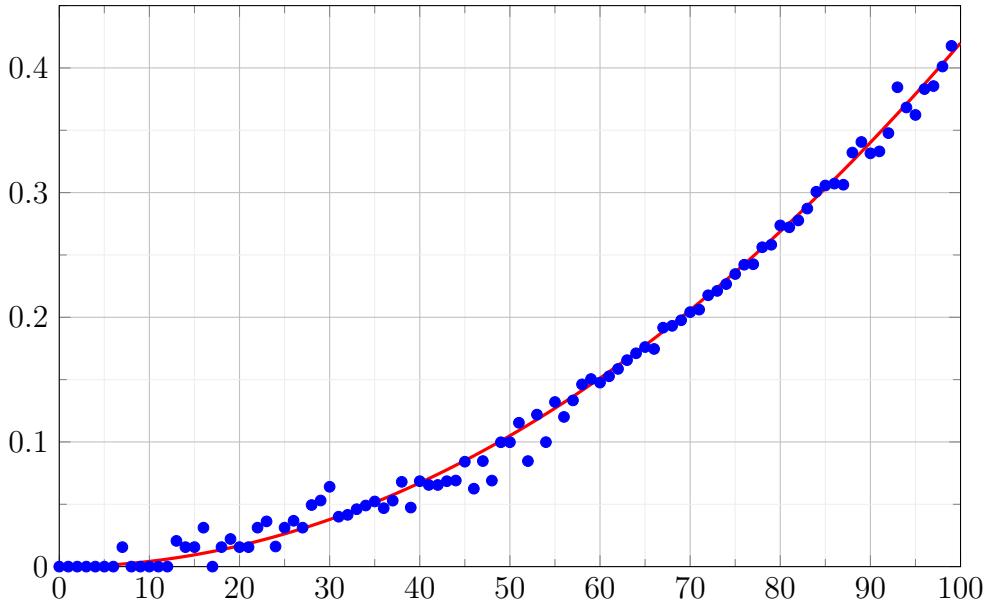
Řešení:
Z programu určíme rekurentní vztah pro složitost algoritmu `insertion_sort_rec`. Zdůvodněte.

$$T(n) = \begin{cases} 1 & n = 0 \\ T(n-1) + n & n > 0 \end{cases}$$

Vypočteme několik prvních hodnot $T(n)$:

n	1	2	3	4	5	6
$T(n)$	$T(0)+1=2$	$T(1)+2=4$	$T(2)+3=7$	$T(3)+4=11$	16	22

Výpočet naznačuje, že složitost algoritmu pro vstup délky n souvisí se součtem hodnot $1, \ldots, n$, tj. že se jedná o kvadratickou složitost. To potvrzuje i měření času potřebného pro řazení pole délky n - viz následující graf. Přeložená červená čára je kvadratická funkce, která naměřené časy poměrně přesně aproximuje.



Matematickou indukcí dokážeme odhad, a to, že algoritmus `insertion_sort_rec` má kvadratickou složitost, tj. že $T(n) = \mathcal{O}(n^2)$.
Nejprve dokážeme indukční krok. Podle definice asymptotické těsné meze předpokládáme, že existuje konstanta $c > 0$ taková, že $T(n) \leq cn^2$. Toto podle indukčního předpokladu platí pro všechna $m < n$. Speciálně pro $m = n - 1$ dostáváme

$T(n-1) \leq c(n-1)^2$. Po úpravách

$$T(n-1) + n \leq c(n-1)^2 + n$$
$$T(n) \leq c(n^2 - n + 1) \leq cn^2, \text{ pro } n \geq 2 \text{ a } c > 0.$$

Ověříme první krok matematické indukce pro $n = 2$, $T(2) = 4 \leq c \cdot 2^2 = 4$ pro $c \geq 1$. Předpoklad kvadratické složitosti je správný pro $n_0 = 2$ a $c \geq 1$.

Úloha 5.1.2. Uvažujme rekurzivní verzi `binary_search_rec` algoritmu binárního vyhledávání. Určete jeho asymptotickou složitost pomocí substituční metody.

```
1  def binary_search_rec(a, v, start, stop):
2      if start > stop:
3          return None
4      mid = (start + stop) // 2
5      if a[mid] == v:
6          return mid
7      if v < a[mid]:
8          return binary_search_rec(a, v, start, mid - 1)
9      else:
10         return binary_search_rec(a, v, mid + 1, stop)
```

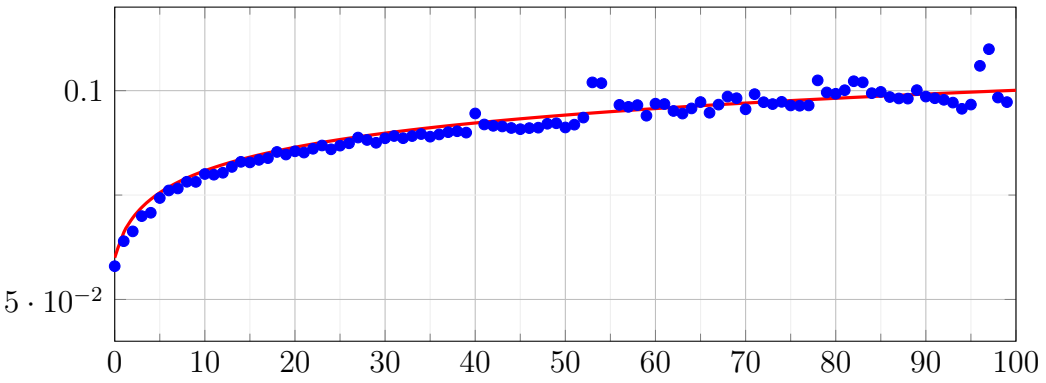
Řešení:
Z programu určíme rekurentní vztah pro složitost algoritmu `binary_search_rec`. Zdůvodněte.

$$T(n) = \begin{cases} \mathcal{O}(1) & n = 1 \\ T\left(\frac{n}{2}\right) + 1 & n = 2^k, k \geq 1 \end{cases}$$

Vypočteme několik prvních hodnot $T(n)$:

n	1	2	4	8	16	32
$T(n)$	1	$T(1)+1=2$	$T(2)+1=3$	$T(4)+1=4$	5	6

Z tabulky je vidět, že při exponenciálním růstu hodnoty n roste $T(n)$ lineárně. To napovídá, že by se mohlo jednat o logaritmickou složitost. V následujícím grafu je znázorněna závislost délky posloupnosti na času, který byl reálně naměřen při hledání čísla v posloupnosti. Přeložená červená čára je logaritmická funkce, která naměřené časy poměrně přesně aproximuje.



Matematickou indukcí dokážeme odhad, a to, že algoritmus `binary_search_rec` má logaritmickou složitost, tj. že $T(n) = \mathcal{O}(\log_2 n)$. Opět nejprve dokážeme indukční

krok. Podle definice asymptotické těsné meze předpokládáme, že existuje $c > 0$ takové, že $T(n) \leq c \log_2 n$. To podle indukčního předpokladu platí pro všechna $m < n$, speciálně i pro $m = \frac{n}{2}$, tj. $T(n/2) \leq c \log_2 (n/2)$. Po úpravách

$$T(n/2) + 1 \leq c \log_2 (n/2) + 1$$
$$T(n) \leq c \log_2 (n/2) + 1 = c (\log_2(n) - \log_2(2)) + 1 = c \log_2(n) - c + 1 \leq c \log_2(n)$$

pro $c \geq 1$.

Ověříme první krok matematické indukce pro $n = 1$. Máme ukázat, že pro $c \geq 1$ je $T(1) \leq c \log_2(1) = 0$. $T(1) \geq 0$, tj. uvedená nerovnost nemůže platit. Proto za počáteční n_0 zvolíme hodnotu větší než 1. Například pro $n_0 = 2$ dostáváme $T(2) = 2 \leq c \log_2(2) = c$, co platí pro $c \geq 2$. Předpoklad logaritmické složitosti algoritmu binárního vyhledávání je správný pro $n_0 = 2$ a $c \geq 2$.

Úloha 5.1.3. Uvažujme řadící algoritmus `merge_sort`. Jedná se o algoritmus, který využívá rekurzi - dělí posloupnost na poloviny a následně je ve funkci `merge` spojuje. Lze předpokládat, že funkce `merge` má složitost $\mathcal{O}(n)$. Určete jeho asymptotickou složitost algoritmu `merge_sort` pomocí substituční metody.

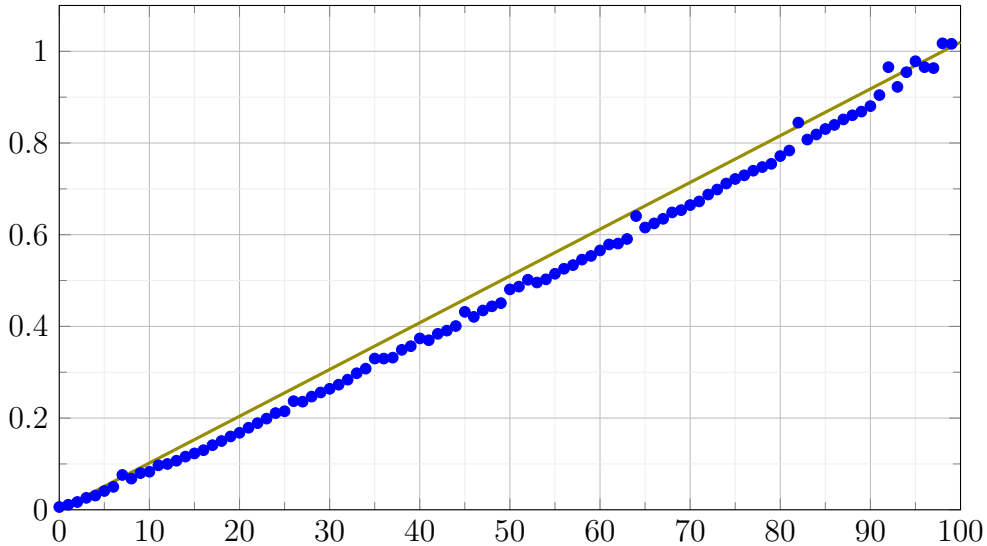
```
1 def merge_sort(a, p, r):
2     if p < r:
3         q = (p + r) / 2
4         merge_sort(a, p, q)
5         merge_sort(a, q + 1, r)
6         merge(a, p, g, r)
```

Řešení:
Z programu určíme rekurentní vztah pro složitost algoritmu `binary_search_rec`. Zdůvodněte.

$$T(n) = \begin{cases} 1 & n = 1 \\ 2T\left(\frac{n}{2}\right) + n & n = 2^k, k \geq 1 \end{cases}$$

Vypočtěte několik prvních hodnot $T(n)$ a odhadněte složitost algoritmu.

V následujícím grafu je znázorněna závislost délky posloupnosti na čase, který byl reálně naměřen při řazení posloupnosti. Zelená čára zobrazuje lineární křivku. Z grafu plyne, že časová náročnost algoritmu `merge_sort` roste rychleji než lineárně.



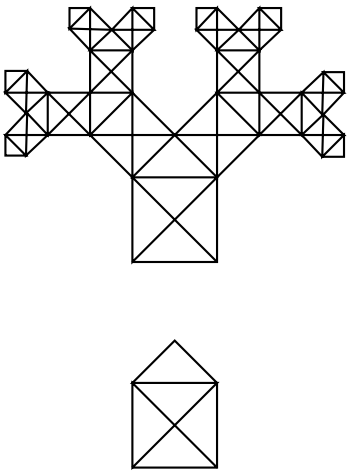
Pro algoritmus `merge_sort` dokažte matematickou indukcí, že $T(n) = \mathcal{O}(n \log_2 n)$.

Úloha 5.1.4. Uvažujeme algoritmus `hanoj` pro přesuny Hanojských věží. Pomocí substituční metody určete jeho asymptotickou složitost. Odhad složitosti dokažte.

```
1 def hanoj(n, odkud, kam, pomoc):
2     if n > 0:
3         hanoj(n-1, odkud, pomoc, kam)
4         print(odkud, '->', kam)
5         hanoj(n-1, pomoc, kam, odkud)
6
7 hanoj(3, 1, 2, 3)
```

Řešení: $T(n) = \mathcal{O}(2^n)$. Zdůvodněte.

Úloha 5.1.5. Celou škálu rekurzivních algoritmů představují algoritmy pro vykreslení rekurzivních křivek. Jednou z nich je křivka nazývaná Pythagorův strom - viz obrázek a algoritmus. Algoritmus vykresluje imaginární želva (`turtle`) při svém pohybu dopředu (`forward`).



```
 1 def house(pg, n):
 2     if n > 0:
 3         t.forward(pg)
 4         t.left(90+45)
 5         t.forward(pg*math.sqrt(2))
 6         t.left(90+45)
 7         t.forward(pg)
 8         t.left(90+45)
 9         t.forward(pg*math.sqrt(2))
10         t.left(90+45)
11         t.forward(pg)
12         t.right(90+45)
13         house(pg/math.sqrt(2),n-1)
14         t.right(90)
15         house(pg/math.sqrt(2),n-1)
16         t.right(45)
17         t.forward(pg)
18         t.left(90)
19     else:
20         t.forward(pg)
21
22 t = turtle.Turtle()
23 house(70, 4)
```

a) Pomocí substituční metody určete složitost algoritmu pro vykreslení Pythagorova stromu.

b) Pythagorův strom se skládá z domečků, které lze nakreslit jedním tahem (domeček pod obrázkem stromu). Jaká je závislost mezi hloubkou vnoření rekurze n a počtem vykreslených domečků? Proveďte výpočet a výsledek zjednodušte s využitím asymptotické těsné meze - notace Θ.

c) Pythagorův strom lze nakreslit jedním tahem. Jaká je délka čáry, kterou v závislosti od hloubky vnoření n želva vykreslí? Udělejte odhad podle algoritmu a výsledek vyjádřete pomocí notace Θ.

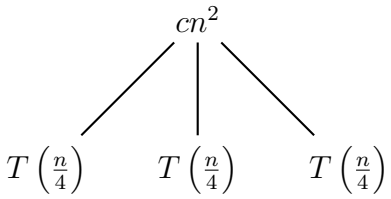
5.2 Metoda rekurzivního stromu

Úloha 5.2.1. Je dán rekurentní vztah:

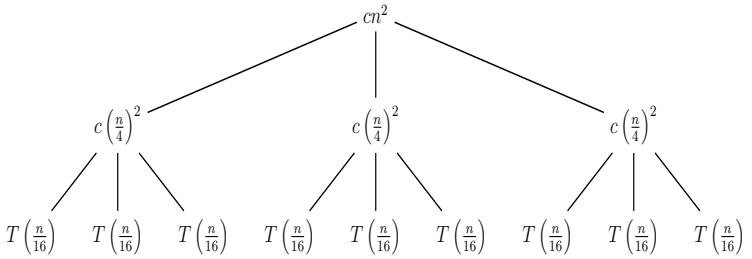
$$T(n) = \begin{cases} \mathcal{O}(1) & n = 1 \\ 3T\left(\frac{n}{4}\right) + cn^2 & n > 1, c > 0 \end{cases} \tag{1}$$

S využitím metody rekurzivního stromu vyjádřete vztah (1) nerekurzivně a určete třídu složitosti algoritmu.

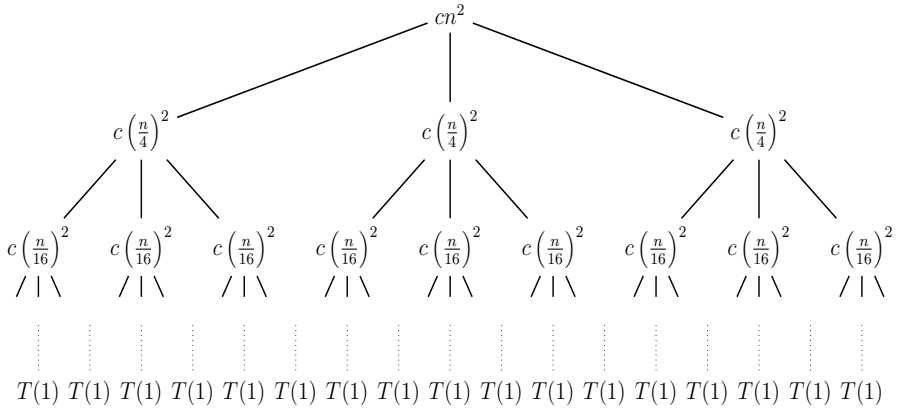
Řešení:
Rekurentní vztah (1) vyjadřuje skutečnost, že problém velikosti n (časová složitost $T(n)$) se v jednotlivých úrovních dělí na tři problémy velikosti $n/4$ a část algoritmu složitosti cn^2. Nakreslíme strom rekurzivních volání.



Každý z problémů $T\left(\frac{n}{4}\right)$ se dále dělí na tři problémy velikosti $n/16$ a část algoritmu se složitostí $c\left(\frac{n}{4}\right)^2$.



Strom rekurzivního výpočtu lze postupně rozkreslit do dalších úrovní až k listům, které představují triviální problémy pro $n = 1$ - viz obrázek.



Zjistíme jednotlivé charakteristiky stromu.
Výška stromu h: Na nulté úrovni řešíme problém velikosti n, na první velikosti $n/4$, na druhé velikosti $n/16$ atd. Na poslední h-té úrovní řešíme problém velikosti 1. Tj. $\frac{n}{4^h} = 1$

$$h = \log_4(n)$$

Počet triviálních případů na úrovni h:

$$3^h = 3^{\log_4(n)} = n^{\log_4 3} \tag{2}$$

Složitost jednotlivého triviálního případu je $\Theta(1)$, tj. odhadovaná složitost všech triviálních případů je podle (2)

$$\Theta\left(n^{\log_4 3}\right) \tag{3}$$

. Na i-té úrovni stromu $0 \leq i < \log_4(n)$ (všechny úrovně stromu mimo nejnižší) se nachází 3^i vrcholů, které odpovídají problémům složitosti

$$c\left(\frac{n}{4^i}\right)^2 = c \cdot n^2 \cdot \left(\frac{1}{16}\right)^i \tag{4}$$

Celkovou složitost algoritmu dostaneme, pokud sečteme dílčí časové náročnost v jednotlivých vrcholech stromu - viz vztahy (3) a (4).

$$\sum_{i=0}^{\log_4 n-1} c \cdot n^2 \cdot \left(\frac{3}{16}\right)^i + \Theta\left(n^{\log_4 3}\right) \leq cn^2 \sum_{i=0}^{\infty} \left(\frac{3}{16}\right)^i + \Theta\left(n^{\log_4 3}\right) =$$

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

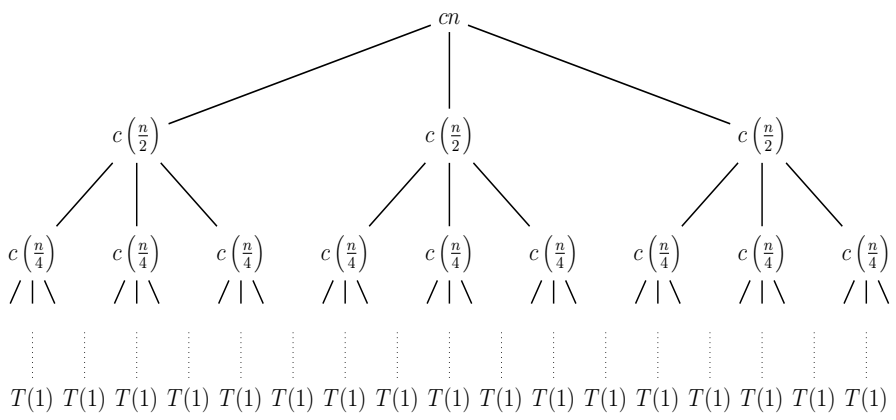
Vidíme, že v tomto případě převážila celková složitost na jednotlivých úrovních stromu nad složitostí triviálních případů (3). Algoritmus má kvadratickou složitost.

Úloha 5.2.2. Je dán rekurentní vztah:

$$T(n) = \begin{cases} \mathcal{O}(1) & n = 1 \\ 3T\left(\frac{n}{2}\right) + n & n > 1 \end{cases} \tag{5}$$

S využitím metody rekurzivního stromu vyjádřete vztah (5) nerekurzivně a určete třídu složitosti algoritmu.

Řešení:
Rekurentní vztah (5) vyjadřuje skutečnost, že problém velikosti n se v jednotlivých úrovních dělí na tři problémy velikosti $n/2$ a část algoritmu složitosti $cn, c > 0$. Nakreslíme strom rekurzivních volání.



Charakteristiky stromu (jeho výšku, počet listů apod.) určíme podobně, jako v předchozím případě. Vypočteme celkovou složitost algoritmu sečtením dílčích složitostí podproblémů odpovídajících jednotlivým vrcholům stromu.

$$\sum_{i=0}^{\log_2 n-1} c \cdot n \cdot \left(\frac{3}{2}\right)^i + \Theta\left(n^{\log_2 3}\right) = cn\frac{\frac{3}{2}^{\log_2 n} - 1}{\frac{3}{2} - 1} + \Theta\left(n^{\log_2 3}\right) =$$

$$= 2cn\left(n^{\log_2 \frac{3}{2}} - 1\right) + \Theta\left(n^{\log_2 3}\right) \leq 2cn^{\log_2 3} + \Theta\left(n^{\log_2 3}\right) \sim \Theta\left(n^{\log_2 3}\right)$$

Úloha 5.2.3. Jsou dány následující rekurentní vztahy. S využitím metody rekurzivního stromu vyjádřete vztahy nerekurzivně.

- $T(n) = \begin{cases} \mathcal{O}(1) & n = 1 \\ 2T\left(\frac{n}{2}\right) + c & n > 1, c > 0 \end{cases}$
- $T(n) = \begin{cases} \mathcal{O}(1) & n = 1 \\ 2T\left(\frac{n}{2}\right) + n^2 & n > 1, c > 0 \end{cases}$
- $T(n) = \begin{cases} \mathcal{O}(1) & n = 1 \\ T\left(\frac{n}{3}\right) + T\left(\frac{2n}{3}\right) + n & n > 1 \end{cases}$
- $T(n) = \begin{cases} \mathcal{O}(1) & n = 1 \\ T\left(\frac{7n}{10}\right) + T\left(\frac{3n}{10}\right) + n & n > 1 \end{cases}$

5.3 Mistrovská věta - Master Theorem

Mistrovská věta - Master Theorem se aplikuje na rekurence ve tvaru

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

kde $a \geq 0$, $b > 0$ jsou konstanty, $f(n)$ je nezáporná funkce. Potom $T(n)$ můžeme asymptoticky ohraničit takto;

1. Je-li $f(n) = \mathcal{O}\left(n^{\log_b a - \epsilon}\right)$ pro nějaké $\epsilon > 0$, pak $T(n) = \Theta\left(\log_b a\right)$.
2. Je-li $f(n) = \Theta\left(n^{\log_b a} \log^k n\right)$ pro nějaké $k \geq 0$[1], pak $T(n) = \Theta\left(n^{\log_b a} \log^{k+1} n\right)$.
3. Je-li $f(n) = \Omega\left(n^{\log_b a + \epsilon}\right)$ pro nějaké $\epsilon > 0$ a jestli $f(n)$ splňuje podmínku regularity, pak $T(n) = \Theta\left(f(n)\right)$.
   Podmínka regularity: $af\left(\frac{n}{b}\right) \leq cf(n)$ pro nějakou konstantu $c < 1$ a všechna dostatečně velká n.

Úloha 5.3.1. Vyjádřete $T(n)$ nerekurzivně. Použijte mistrovskou větu.

$$T(n) = 9T\left(\frac{n}{3}\right) + n$$

Řešení: $a = 9, b = 3, f(n) = n$ - splňují podmínky MT
$n^{\log_b a} = n^{\log_3 9} = n^2$

[1] Často $k = 0$.

$\lim_{n\to\infty} \frac{n}{n^{2-\epsilon}} < \infty$, například pro $\epsilon = 0,5, \lim_{n\to\infty} \frac{n}{n^{2-0,5}} = 0$
Jedná se o 1. případ MT.

$$T(n) = \Theta(n^2)$$

.

Úloha 5.3.2. Vyjádřete $T(n)$ nerekurzivně. Použijte mistrovskou větu.

$$T(n) = T\left(\frac{2n}{3}\right) + 1$$

Řešení: $a = 1, b = \frac{3}{2} > 0, f(n) = 1$
$n^{\log_b a} = n^{\log_{3/2} 1} = n^0 = 1$
$\lim_{n\to\infty} \frac{1}{1} = 1$
Jedná se o 2. případ MT pro $k = 0$.

$$T(n) = \Theta(\log_2 n)$$

.

Úloha 5.3.3. Vyjádřete $T(n)$ nerekurzivně. Použijte mistrovskou větu.

$$T(n) = 3T\left(\frac{n}{4}\right) + n\log_2 n$$

Řešení: $a = 3, b = 4, f(n) = n\log_2 n$ - splňují podmínky MT
$n^{\log_b a} = n^{\log_4 3}$
$\lim_{n\to\infty} \frac{n\log_2 n}{n^{\log_4 3+\epsilon}} > 0$, například pro $\epsilon = 0,2, \lim_{n\to\infty} \frac{n\log_2 n}{n^{0,99}} = \infty$
Jedná se o 3. případ MT. Ověříme podmínku regularity:
$3f\left(\frac{n}{4}\right) \leq cf(n), 3\frac{n}{4}\log_2 \frac{n}{4} \leq cn\log_2 n, \frac{3}{4}n(log_2 n - 2) \leq cn\log_2 n, \frac{3}{4} < c < 1$

$$T(n) = \Theta(n\log_2 n)$$

Úloha 5.3.4. S využitím mistrovské věty upravte vztahy pro $T(n)$.

- $T(n) = 3T\left(\frac{n}{4}\right) + n^2$
- $T(n) = 3T\left(\frac{n}{2}\right) + n^2$
- $T(n) = 4T\left(\frac{n}{2}\right) + n^2$
- $T(n) = 3T\left(\frac{n}{2}\right) + n$
- $T(n) = 16T\left(\frac{n}{4}\right) + n$
- $T(n) = 16T\left(\frac{n}{4}\right) + n!$
- $T(n) = 2T\left(\frac{n}{2}\right) + n\log_2 n$
- $T(n) = 2T\left(\frac{n}{2}\right) + \Theta(n)$
- $T(n) = 7T\left(\frac{n}{3}\right) + \Theta(n^2)$
- $T(n) = 2T\left(\frac{n}{4}\right) + 1$
- $T(n) = 0,5T\left(\frac{n}{2}\right) + \frac{1}{n}$
- $T(n) = 2T\left(\frac{n}{4}\right) + \sqrt{n}$
- $T(n) = 2T\left(\frac{n}{4}\right) + n^{0,51}$
- $T(n) = 2T\left(\frac{n}{4}\right) + n$
- $T(n) = 2T\left(\frac{n}{4}\right) + n\log_2 n$
- $T(n) = T\left(\frac{n}{2}\right) + 2^n$
- $T(n) = 2^n T\left(\frac{n}{2}\right) + n^n$

- $T(n) = \sqrt{2}T\left(\frac{n}{2}\right) + \log_2 n$
- $T(n) = 4T\left(\frac{n}{2}\right) + \frac{n}{\log_2 n}\log_2 n$
- $T(n) = 64T\left(\frac{n}{8}\right) - n^2 \log_2 n$
- $T(n) = 3T\left(\frac{n}{3}\right) + \frac{n}{2}$

Úloha 5.3.5. Pro které z příkladů v úloze 5.2.3 lze využít Master Theorém? Tam, kde lze mistrovskou větu použít, ji aplikujte a srovnejte výsledky získané metodou rekurzivního stromu a s výsledky výpočtu podle Master Theorem.

Úloha 5.3.6. Pro úlohy 5.1.2 a 5.1.3 jsme odečetli rekurentní vztahy přímo z algoritmů a jejich nerekurzivní podobu jsme získali odhadem pomocí substituční metody. Pomocí mistrovské věty ukažte, že odhad byl správný.
Lze pro řešení úlohy 5.1.1 využít Master Theorem? Zdůvodněte.