Digitální učební materiál

Číslo projektu:	CZ.1.07/1.5.00/34.0548
Název školy:	Gymnázium, Trutnov, Jiráskovo náměstí 325
Název materiálu:	VY_32_INOVACE_146_IVT
Autor:	Ing. Pavel Bezděk
Tematický okruh:	Algoritmy
Datum tvorby:	červenec 2013
Ročník:	4. ročník a oktáva
Anotace:	Algoritmus VI. – Asymptotická výpočetní složitost algoritmu
Metodický pokyn:	Při výuce nutno postupovat individuálně. Části DUM – „ Pro hloubavé“ jsou určeny pro zájemce o studium na technických a matematicko-fyzikálních oborech vysokých škol.

esf evropský sociální fond v ČR
EVROPSKÁ UNIE
MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY
OP Vzdělávání pro konkurenceschopnost
INVESTICE DO ROZVOJE VZDĚLÁVÁNÍ

Autor	**Ing. Pavel Bezděk**		
Vytvořeno dne	**23. 7. 2013**		
Odpilotováno dne	**16. 12. 2013**	**ve třídě**	**8.Y**
Vzdělávací oblast	**Informatika a informační a komunikační technologie**		
Vzdělávací obor	**Informatika a výpočetní technika**		
Tematický okruh	**Algoritmus**		
Téma	**Algoritmus VI. - Asymptotická složitost algoritmu**		
Klíčová slova	**Algoritmus, asymptotická složitost algoritmu**		

Asymptotická složitost algoritmu

Jen konečnost algoritmu vždy nestačí

Je třeba poznamenat, že abstraktní kritérium *konečnosti* je pro praktické použití ještě příliš slabé.
V praxi je třeba zajistit, aby algoritmus skončil „v rozumném“ čase. Za rozumný čas lze v praxi považovat takový čas, který nám umožní smysluplně využít výsledek.

Např. existuje **jednoduchý algoritmus**, který **dokáže určit, zda v dané šachové pozici může hráč na tahu vynutit vítězství a zároveň dokáže určit nejlepší možný tah.** Tento algoritmus se však ***nedá použít, protože by na svou činnost potřeboval ohromné množství času, i když je toto množství konečné***. Zároveň by takový algoritmus ***spotřeboval ohromné množství paměti***, což je další praktický zřetel, který se uplatňuje při volbě algoritmu. I když průměrná počítačová paměť stále narůstá, pro některé algoritmy jí nebude nikdy dost.

Asymptotická složitost algoritmu

Pro vyčíslení výpočetní složitosti algoritmů v závislosti na velikosti vstupních dat se používá asymptotický zápis závislosti výpočetního času na rozsahu úlohy (typicky na počtu vstupních údajů). Například ***O(log N)*** **znamená, že počet kroků algoritmu závisí logaritmicky na velikosti vstupních dat.**

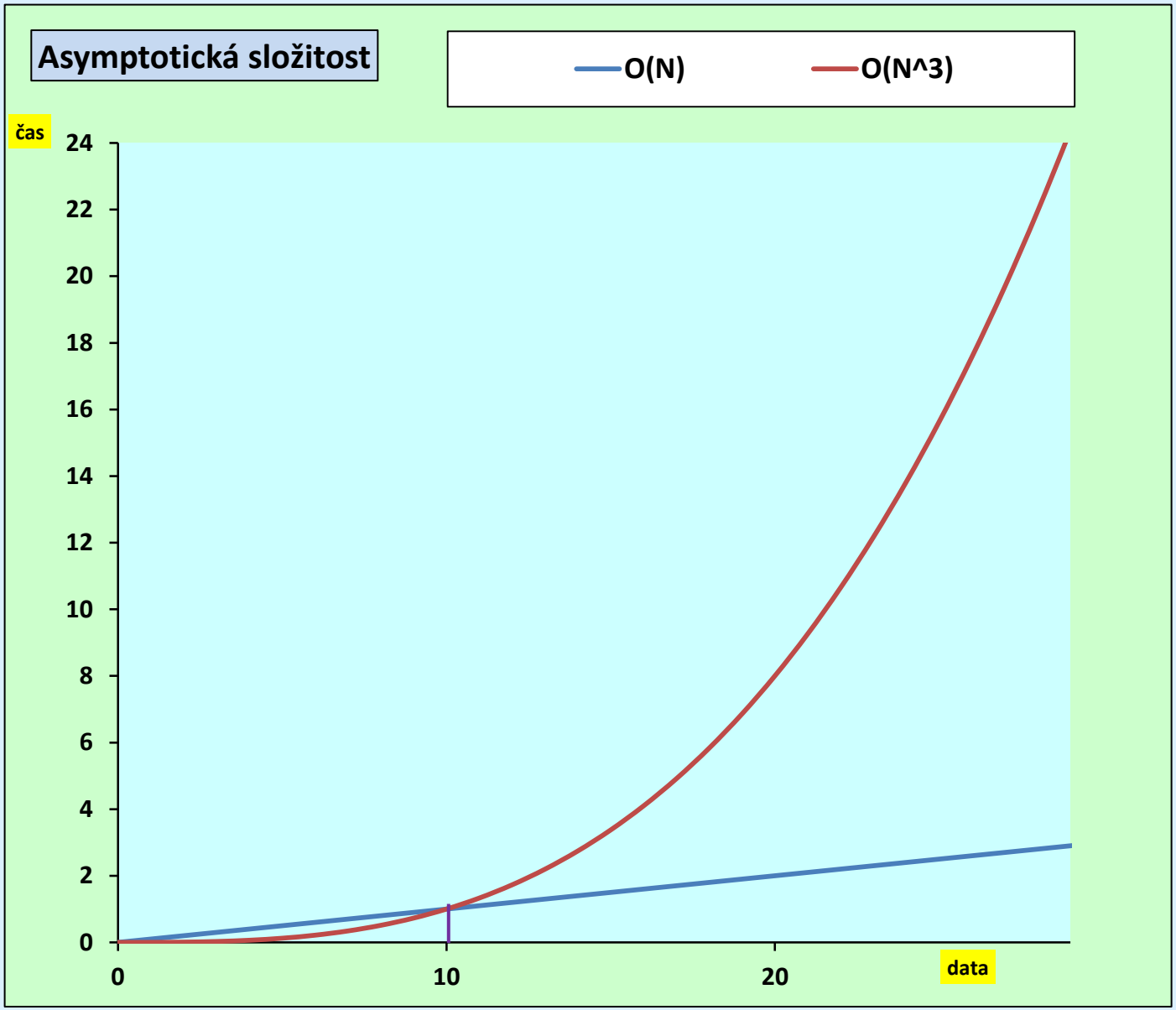
Pokud u takového algoritmu zdvojnásobíme rozsah vstupních údajů, doba výpočtu se zvýší o jednu jednotku času, pokud bude vstupních dat čtyřikrát více, doba výpočtu se prodlouží o dvě jednotky času, a tak dále. To je třeba případ nalezení jednoho prvku o určité hodnotě v seznamu prvků seřazeném podle hodnoty (např. nalezení jména v telefonním seznamu).

V této části textu se budeme věnovat pouze časové složitosti.

Všechna pravidla, která si řekneme, pak budou platit i pro paměťovou složitost.

U určování časové složitosti nás bude především zajímat, ***jak se algoritmy chovají pro velké vstupy***.

Závislost množství dat na délce běhu programu



Mějme například algoritmus **A** o časové složitosti N a algoritmus **B** o složitosti N^3. Sice pro malé hodnoty N bude algoritmus B rychlejší než A, ale pro všechna větší N(od 10 výše) ho už algoritmus A předběhne. Takže pokud bychom si měli mezi těmito algoritmy zvolit, vybereme si algoritmus A. U složitosti nás obvykle nebude zajímat, jak se chová na malých vstupech, protože na těch je rychlý téměř každý algoritmus. ***Rozhodující pro nás bude složitost na maximálních vstupech (pokud nějaké omezení existuje) anebo složitost pro „hodně velké vstupy“.*** *Proto si zavedeme tzv.* ***asymptotickou časovou složitost****.*

Představme si, že máme algoritmus se složitostí $N^2/4 + 6*N + 12$.
Pod asymptotickou složitostí si můžeme představit, že nás zajímá jen nejvýznamnější člen výrazu, podle kterého se pak pro velké vstupy chová celý výraz. To znamená, že: Konstanty u jednotlivých členů můžeme škrtnout (např. 6n se chová podobně jako n). Tím dostáváme $N^2 + N + 1$.
Pro velká N je N + 1 oproti N^2 nevýznamné, tak ho můžeme také škrtnout. Dostáváme tak složitost $O(N^2)$. Obecně škrtáme všechny členy, které jsou pro dost velké N menší, než nějaký neškrtnutý člen.

Tahle pravidla sice většinou fungují, ale škrtat ve výpočtech přece nemůžeme jen tak. Proto zavádíme operátor **O** (velké O), díky kterému budeme umět popsat, co přesně naše „škrtání“ znamená, a používat ho korektně.

Zapisuje se pomocí „Omikron notace“ („velké **O** notace“ nebo „Big **O** notation“)

Asymptotický horní odhad časové složitosti - O(g(n))

Formálně f(***n***)= O(**g**(***n***)), **stejně** jako f(***n***)∈ O(**g**(***n***)). **„Big O notation“**

Pro danou funkci **g**(***n***) označme množinu funkcí O(**g**(***n***)), kde **O** je velké **omikron**:

O(**g**(***n***)) = {f(***n***): existují-li kladné konstanty *c* a n_0 takové, že
0 ≤ f(***n***) ≤ *c****g**(***n***) pro všechny ***n*** ≥ n_0}.

Zápis **f (n) = O(g(n))** lze popsat jako **f neroste řádově rychleji než funkce g**.

f je asymptoticky ohraničena funkcí g shora (až na konstantu)

Od určitého n_0 už leží graf funkce f (n) pod grafem funkce c*g(n).

Tedy f(n) roste řádově nejvýše tak rychle jako g(n). $\lim_{n \to +\infty} f(n)/g(n) = 0$

Náročnost algoritmu je vždy stejná nebo menší, než nějaký násobek g(n).

Jde o maximální možnou složitost algoritmu.

Říkáme, že g(*n*) je horným odhadem funkce f(*n*)

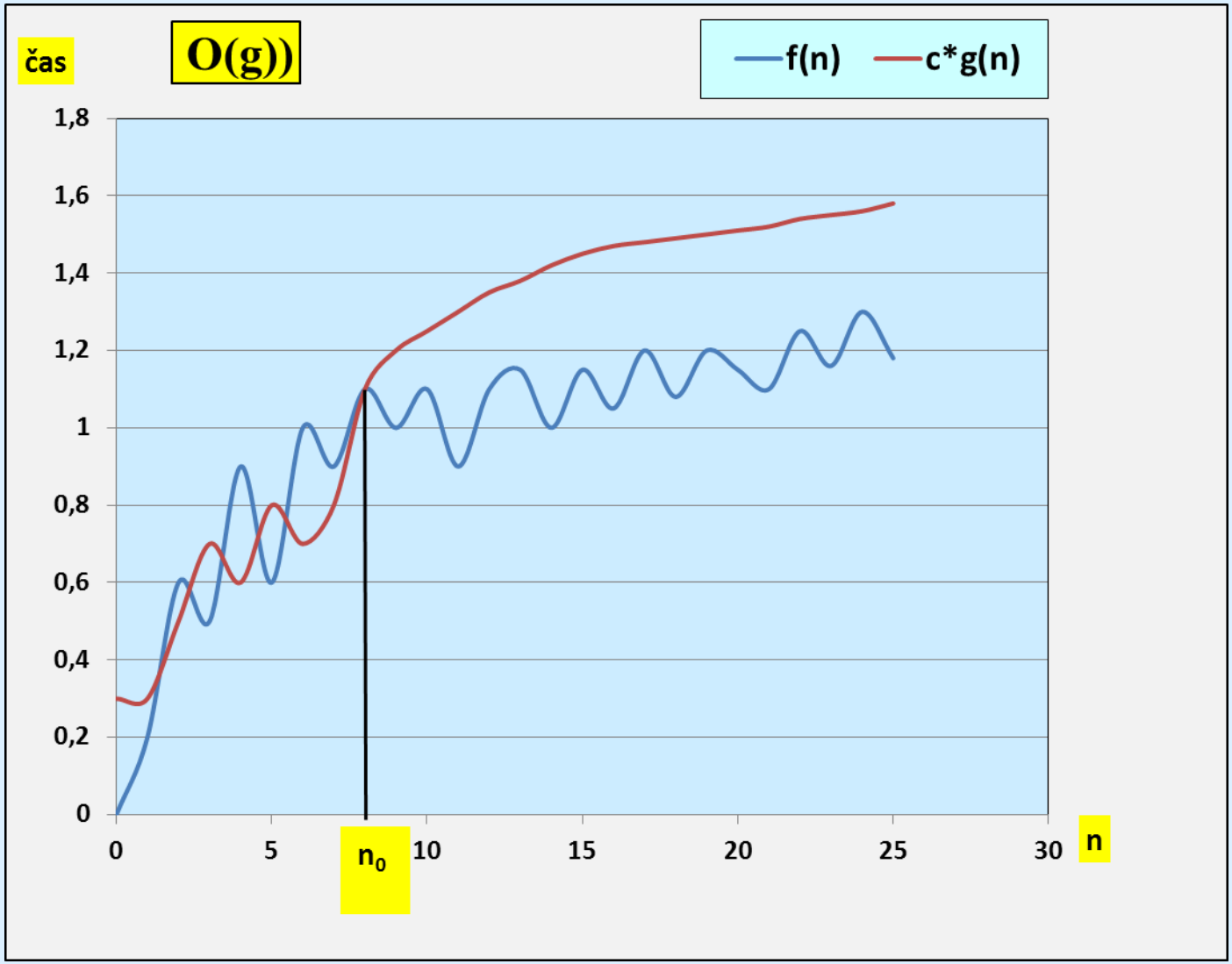
To znamená, že funkce g shora omezuje funkci f až na multiplikativní konstantu.
Proto také f = O(g) někdy čte jako **„f je asymptoticky menší nebo rovno g“**.

0 ≤ f(***n***) ≤ *c****g**(***n***) pro všechny ***n*** ≥ n_0

Asymptotický horní odhad časové složitosti - O(g(n))

Funkce f se dá shora odhadnout funkcí g

(až na multiplikativní konstantu a pro dostatečně velká n).



Pro hloubavé:

Asymptotický dolní odhad časové složitosti - Ω(g(n))

Formálně f(***n***)= Ω(**g**(***n***)), **stejně** jako f(***n***)∈ Ω(**g**(***n***)).

Pro danou funkci **g**(***n***) označme množinu funkcí Ω(**g**(***n***)), kde Ω je velká **omega**:

Ω **g**(***n***)) = {f(***n***): existují-li kladné konstanty *c* a n_0 takové, že

$0 \leq c^*\mathbf{g}(n) \leq f(n)$ pro všechny $n \geq n_0$}.

Zápis **f (n) = Ω(g(n))** znamená, že funkce **f** roste řádově aspoň tak rychle, jako funkce **g**

Od určitého n_0 už leží graf funkce f (n) nad grafem funkce c*g(n).

Říkáme, že **g(*n*) je dolním odhadem funkce f(*n*)**

Algoritmus pro alespoň jeden vstup bude této složitosti.

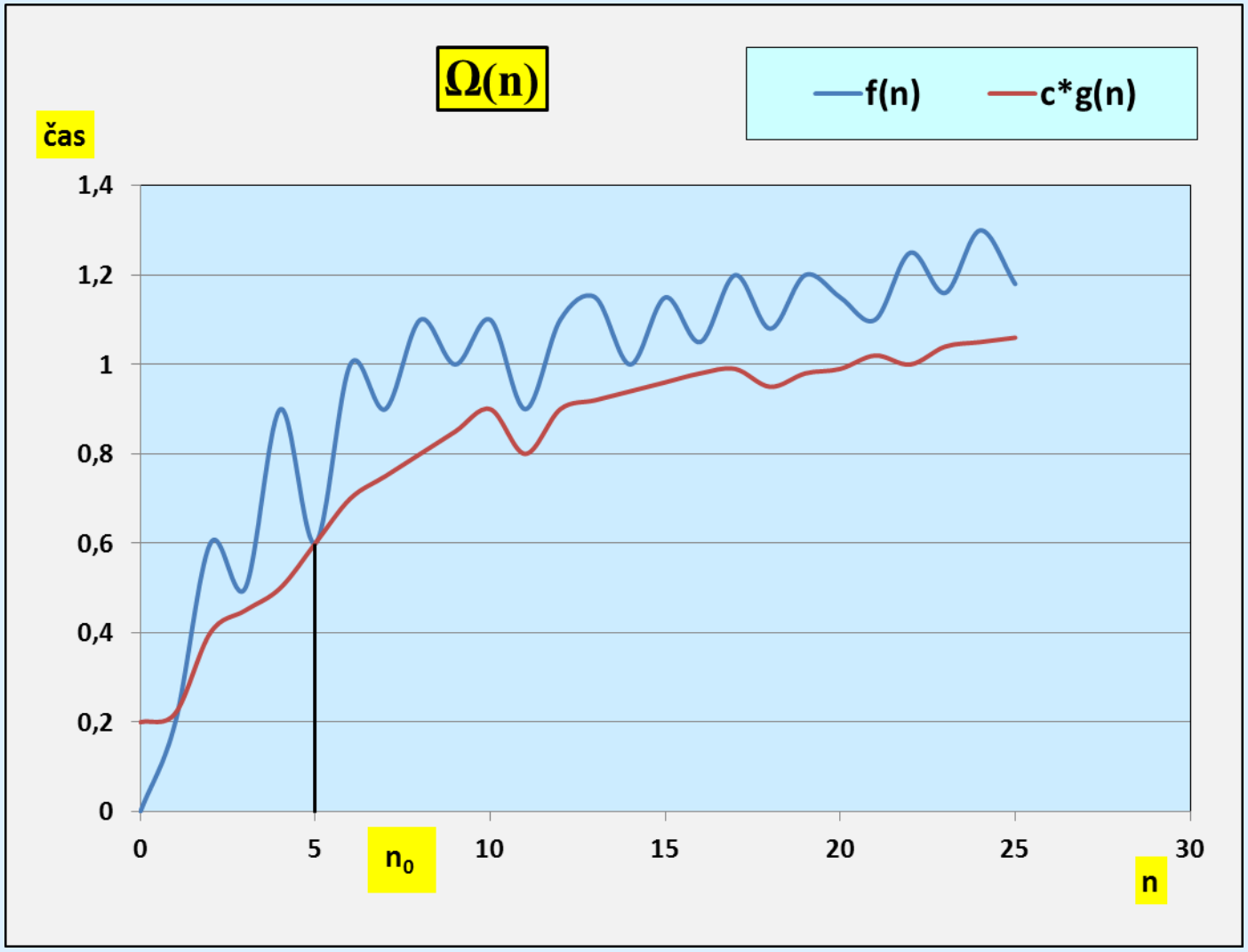
Náročnost algoritmu je vždy stejná nebo větší, než nějaký násobek g(n).

f je asymptoticky ohraničena funkcí **g** zdola (až na konstantu) $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$

Asymptotický dolní odhad časové složitosti - Ω(g(n))

Funkce f se dá zdola odhadnout funkcí g

(až na multiplikativní konstantu a pro dostatečně velká n).



Pro hloubavé:

Asymptotický střední odhad časové složitosti - Θ(g(n))

Formálně f(***n***)= Θ(**g**(***n***)), **stejně** jako f(***n***) ∈Θ(**g**(***n***)).

Pro danou funkci **g**(***n***) označme množinu funkcí Θ(**g**(***n***)), kde Θ je **velká théta**:

Θ(**g**(***n***)) = {f(***n***): existují-li kladné konstanty c_1, c_2 a n_0 takové, že

$0 \le c_1 * \mathbf{g}(n) \le f(n) \le c_2 * \mathbf{g}(n)$ pro všechny $n \ge n_0$}.

Zápis **f (n) = Θ(g(n)) znamená**, že obě funkce rostou řádově stejně rychle.

Říkáme, že **g(*n*) je středním odhadem funkce f(*n*)**

Θ(g(n)) algoritmus probíhá asymptoticky stejně rychle jako g(n).

Náročnost algoritmu je stejná jako g(n)

To znamená, že problém lze řešit algoritmem, který nebude asymptoticky složitější než c_1*g(n), ale zároveň nikdy nebude asymptoticky lepší než c_2*g(n).

f je asymptoticky ohraničena funkcí **g** z obou stran (až na konstantu)

f(n) je zároveň v O(g(n)) a v Ω(g(n)). $0 < \lim_{n \to +\infty} f(n)/g(n) < +\infty$

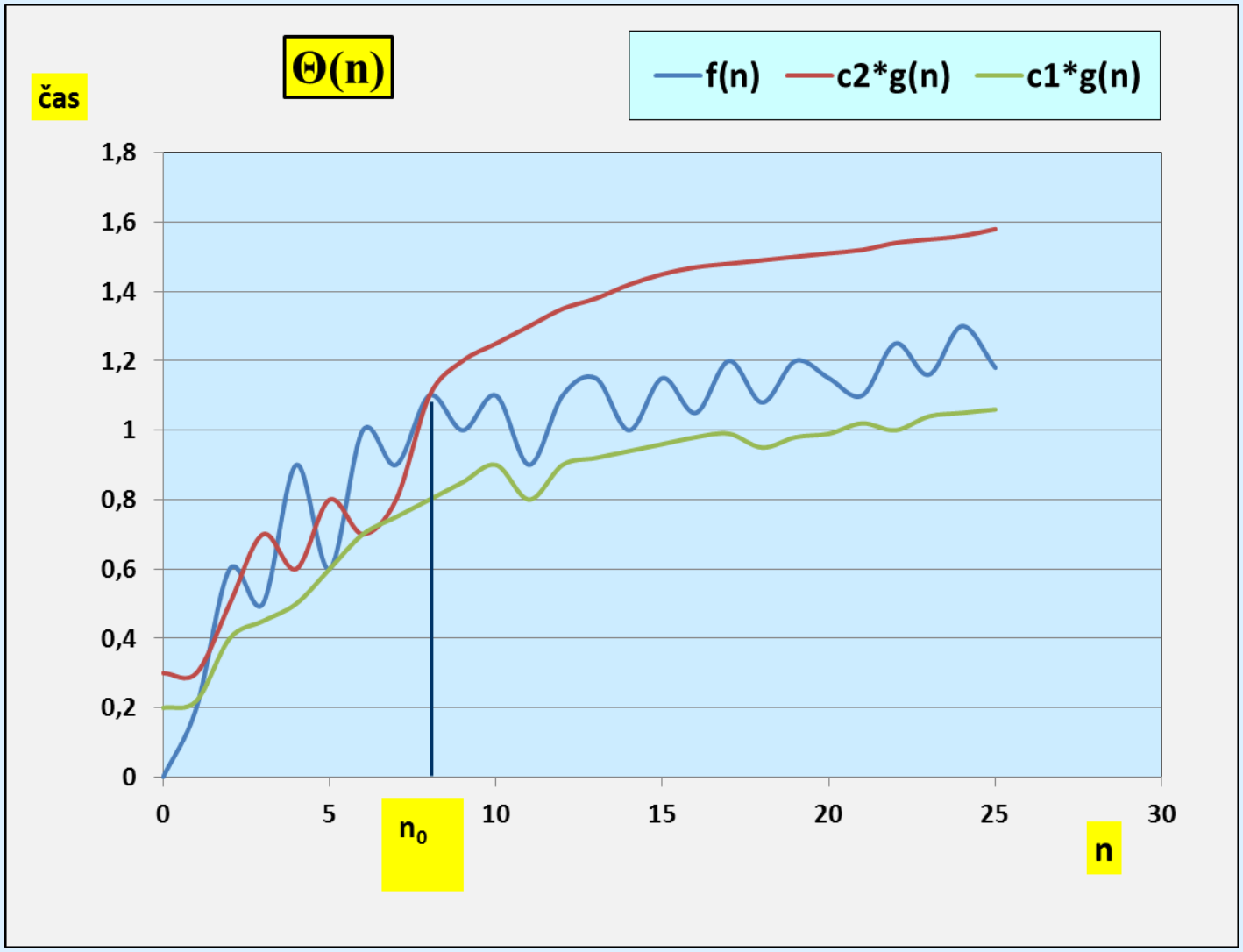
Pro libovolné funkce f(n) a g(n) platí f(n) = Θ(g(n)),

právě když f(n) = O(g(n)) a současně f(n) = Ω(g(n)).

Asymptotický střední odhad časové složitosti - Θ(g(n))

Funkce f se dá ohraničit funkcí g z obou stran

(až na multiplikativní konstanty a pro dostatečně velká n).



Pro hloubavé:

Limitní testy

Předpokládejme výše uvedené funkce f(***n***) a g(***n***).

Mějme limitu $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$

Potom f(***n***)=O(**g**(***n***)), ale f(***n***)≠Ω(**g**(***n***)). Říkáme, že: "f(***n***) roste **asymptoticky pomaleji než g(*n*)**".

Mějme limitu $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$

Potom f(***n***)= Ω(**g**(***n***)), ale f(***n***) ≠ O(**g**(***n***)). Říkáme, že: "f(***n***) roste **asymptoticky rychleji než g(*n*)**".

Mějme limitu $\lim_{n\to\infty} \frac{f(n)}{g(n)} = c$

Pro nějakou konstantu *c*>0, potom f(***n***)=Θ(**g**(***n***)). Říkáme, že: "f(***n***) a g(***n***) rostou **asymptoticky stejně rychle**".

Užití Big O notace

Pro nás má význam hlavně O(g(n))

Tedy, že algoritmus nebude mít složitost pro velká **n** větší než nějaká konstanta krát g(n). **Máme tu mez, kterou složitost algoritmu už nepřekročí.** Vždy se snažíme vyjádřit relaci O(**g**(***n***)) tak, aby **g**(***n***) byla co nejjednodušší. Ω(g(n)) a Θ(g(n)) vždy nepočítáme, většinou nám stačí O(g(n)).

Notaci O(**g**(***n***)) je nutné užívat ve správném smyslu.

Jestliže máme algoritmus, který má dobu běhu 10^{100}n, což je funkce která je O(n), potom je to asymptoticky lepší než 10n*log n, která jeO(n.log n) i když samozřejmě je lepší využít ten druhý algoritmus s O(n.log n). U asymptotických charakteristik je nutné si uvědomit, že je nelze přeceňovat, respektive nevhodně užívat.

Podobně, jestliže máme algoritmus s dobou běhu **O(*n* log(*n*))** s rozumným konstantním faktorem, pak je takový algoritmus vždy efektivnější než **O(n^2)**, který však může být lepší pro malé hodnoty ***n***.

Použité zdroje

BÖHM, Martin. *Programátorská kuchařka: Recepty z programátorské kuchařky* [online]. Praha: KSP MFF UK Praha, 2011/2012 [cit. 2013-07-23]. KSP, Korespondenční seminář z programování: Programátorské kuchařky, 24. ročník KSP. Dostupné z: http://ksp.mff.cuni.cz/tasks/24/cook1.html

Algoritmy a datové struktury [online]. 20. 07. 2002 [cit. 2013-07-23]. Dostupné z: http://*Algoritmy a datové struktury* [online]. 20. 07. 2002 [cit. 2013-07-23]. Dostupné z: http://www.alg.webzdarma.cz/diplomka/kap2/asymptot.html