

# Různé algoritmy mají různou složitost

**Algoritmus a program není totéž**

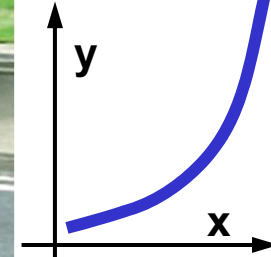
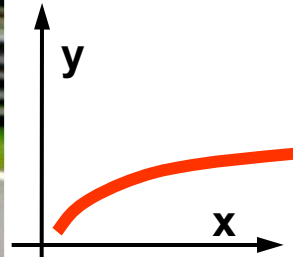
## Rychlost...



**Jeden algoritmus (program, postup, metoda...)  
je rychlejší než druhý.**

**Co ta věta znamená ??**

## Asymptotická složitost



Každému algoritmu lze jednoznačně přiřadit

**rostoucí funkci**

zvanou

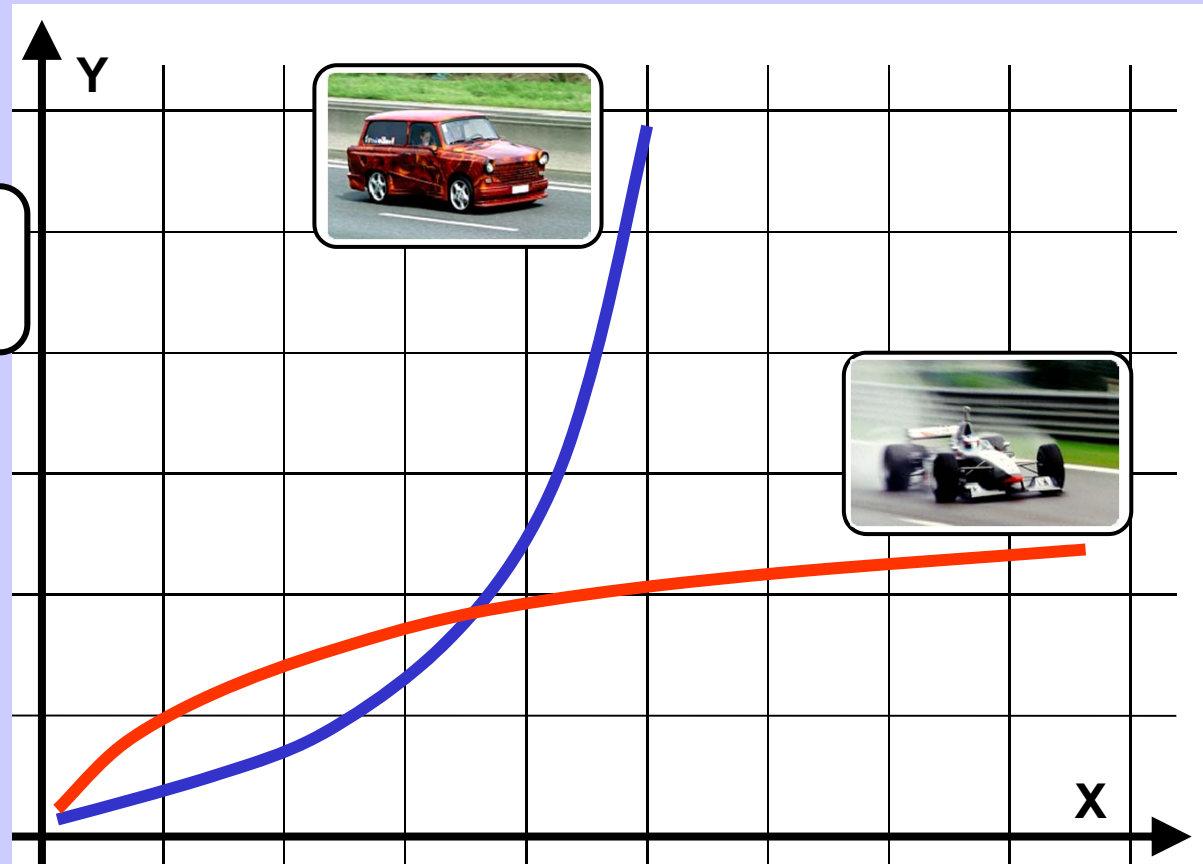
**asymptotická složitost,**

která charakterizuje počet operací algoritmu  
v závislosti na rostoucím rozsahu vstupních dat.

Čím pomaleji tato funkce roste, tím je algoritmus rychlejší.

# Asymptotická složitost

$Y \sim$  zatížení systému  
(trvání výpočtu)



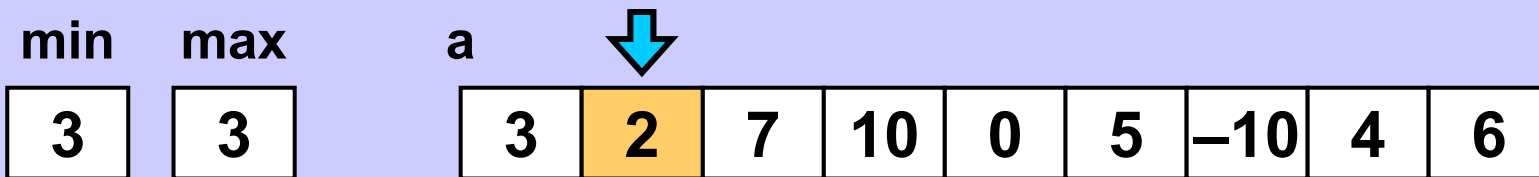
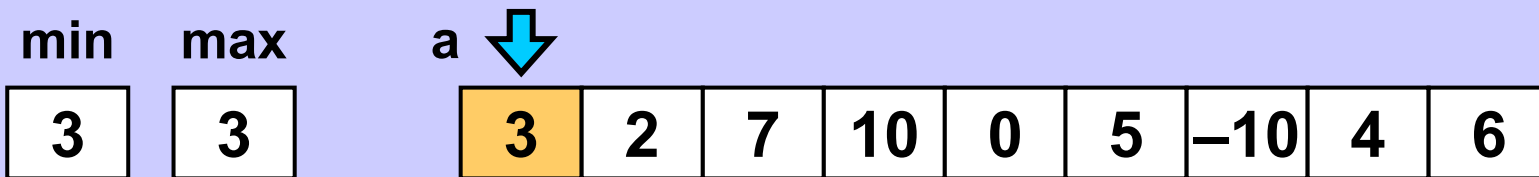
$X \sim$  naše požadavky  
(rozsah vstupních dat)



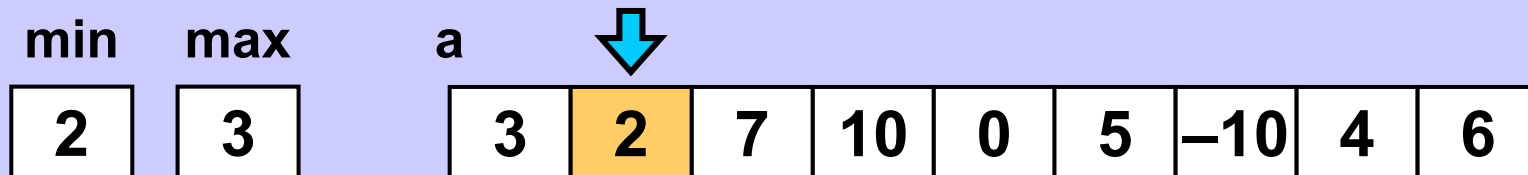
## Příklady



Najdi min and max hodnotu v poli — STANDARD



```
if (a[i] < min) min = a[i];  
if (a[i] > max) max = a[i];
```



## Příklady



Najdi min and max hodnotu v poli — STANDARD

min	max	a
2	7	3 2 7 10 0 5 -10 4 6

atd...

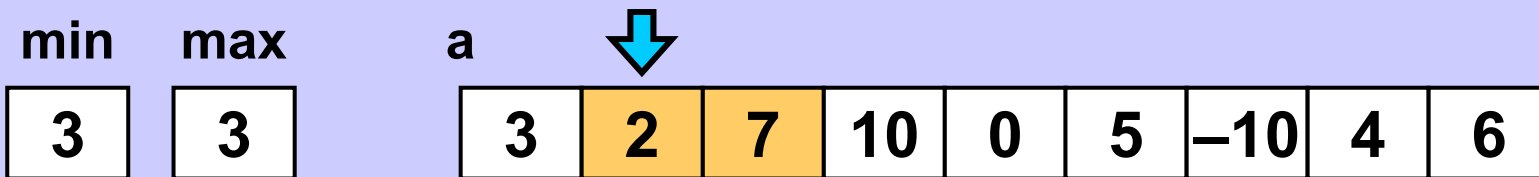
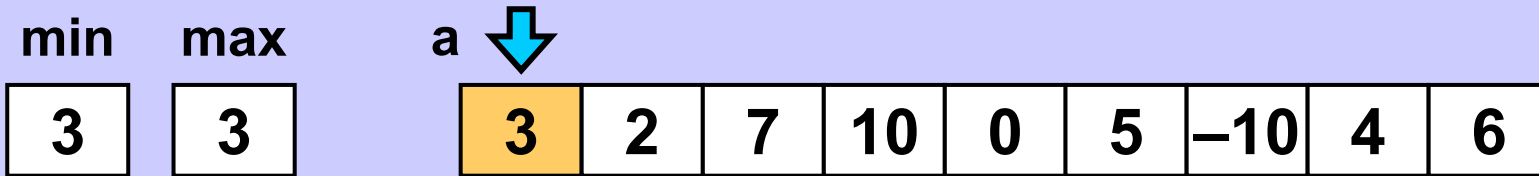
hotovo	min	max	a
	-10	10	3 2 7 10 0 5 -10 4 6

kód

```
min = a[0]; max = a[0];  
for ( i = 1; i < a.length; i++) {  
    if (a[i] < min) min = a[i];  
    if (a[i] > max) max = a[i]; }  
}
```

Příklady 

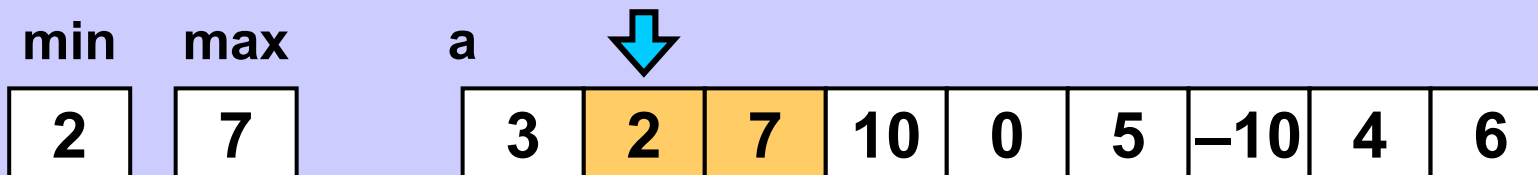
Najdi min and max hodnotu v poli — RYCHLEJI!



```

if (a[i] < a[i+1]) {
    if (a[i] < min) min = a[i];
    if (a[i+1] > max) max = a[i+1];
}

```



# Příklady



Najdi min and max hodnotu v poli — RYCHLEJI!

min	max	a
2	7	3 2 7 10 0 5 -10 4 6

```
if (a[i] < a[i+1]) {  
    if ( a[i] < min) min = a[i];  
    if (a[i+1] > max) max = a[i+1];  
else {  
    if ( a[i] > max) max = a[i];  
    if (a[i+1] < min) min = a[i+1];  
}
```

min	max	a
0	10	3 2 7 10 0 5 -10 4 6



## Příklady



Najdi min and max hodnotu v poli — RYCHLEJI!

hotovo

min	max	a									
-10	10	3	2	7	10	0	5	-10	4	6	

kód

```
min = a[0]; max = a[0];  
for (i=1; i < a.length-1; i=i+2 ) {  
    if (a[i] < a[i+1]) {  
        if ( a[i] < min) min = a[i];  
        if (a[i+1] > max) max = a[i+1];  
    }  
    else {  
        if ( a[i] > max) max = a[i];  
        if (a[i+1] < min) min = a[i+1];  
    }  
}
```

## Počítání složitosti

### Elementární operace

aritmetická operace  
porovnání dvou čísel  
přesun čísla v paměti

### Složitost

**A**

celkový počet elementárních operací

zjednodušení

### Složitost

**B**

celkový počet elementárních operací nad daty

## Počítání složitosti

Složitost

**B**

celkový počet elementárních operací nad daty

další  
zjednodušení

Složitost

**C**

celkový počet porovnání čísel (znaků)  
v datech

Takto složitost mnohdy počítáme

# Počítání složitosti



Najdi min and max hodnotu v poli — STANDARD

A

Složitost

Všechny operace

Případ

nejlepší

nejhorší

```

    1           1           a.length = N
    min = a[0]; max = a[0];
    for ( i = 1; i < a.length; i++) {
        if (a[i] < min) min = a[i];
        if (a[i] > max) max = a[i]; }
  
```

Diagrammatic annotations for complexity analysis:

- Green boxes with '1' above the initial assignments of `min` and `max`.
- Green boxes with '1' above the loop start condition `i = 1` and `N` above `i < a.length`.
- Green boxes with `N-1` above the loop increment `i++`.
- Green boxes with `N-1` above the `if` conditions.
- Green boxes with `0...N-1` above the assignment `max = a[i]`.
- A red circle around the word "exkluzive!" with arrows pointing to the `if` conditions.

$$1 + 1 + 1 + N + N - 1 + N - 1 + 0 + N - 1 + 0 = 4N$$

$$1 + 1 + 1 + N + N - 1 + N - 1 + N - 1 + N - 1 = \underline{\underline{5N - 1}}$$



# Počítání složitosti



Najdi min and max hodnotu v poli — STANDARD

**B**

složitost

Operace  
nad daty

Případ

nejlepší

nejhorší

```

    1           1           a.length = N
    min = a[0]; max = a[0];
    for ( i = 1; i < a.length; i++ ) {
        if ( a[i] < min ) min = a[i];
        if ( a[i] > max ) max = a[i]; }
  
```

$$1 + 1 + N - 1 + 0 + N - 1 + 0 = 2N$$

$$1 + 1 + N - 1 + N - 1 + N - 1 + N - 1 = \underline{\underline{4N - 2}}$$

## Počítání složitosti



Najdi min and max hodnotu v poli — STANDARD

složitost

C

pouze  
testy v datech

```

a.length = N
min ≙ a[0]; max ≙ a[0];
for ( i ≙ 1; i < a.length; i++ ) {
    if ( a[i] < min ) min ≙ a[i];
    if ( a[i] > max ) max ≙ a[i]; }

```

vždy

$N-1 + N-1 = \underline{\underline{2N-2}}$  testů

# Počítání složitost

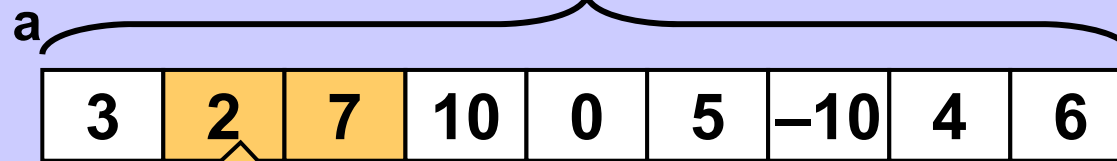


Najdi min and max hodnotu v poli — RYCHLEJI!

složitost **C**

pouze  
testy v datech

a.length = N



Jedna dvojice — 3 testy

$(N-1)/2$  dvojic

vždy

$3(N-1)/2 = \underline{\underline{(3N - 3)/2}}$  testů

## Počítání složitosti

Velikost pole N	Počet testů STANDARDNÍ $2(N - 1)$	Počet testů RYCHLEJŠÍ $(3N - 3)/2$	poměr STD./RYCHL.
11	20	15	1.33
21	40	30	1.33
51	100	75	1.33
101	200	150	1.33
201	400	300	1.33
501	1 000	750	1.33
1 001	2 000	1 500	1.33
2 001	4 000	3 000	1.33
5 001	10 000	7 500	1.33
1 000 001	2 000 000	1 500 000	1.33

Tab. 1



## Příklady

**data**

pole a:

1	-1	0	-2	5	1	0
---	----	---	----	---	---	---

pole b:

4	2	4	3	4	2	7
---	---	---	---	---	---	---

**úloha**Kolik prvků pole b je rovno součtu prvků pole a?**řešení**

pole a:

1	-1	0	-2	5	1	0
---	----	---	----	---	---	---

součet = 4

pole b:

4	2	4	3	4	2	7
---	---	---	---	---	---	---

výsledek = 3

## Příklady

funkce

```
int sumArr(int[] a) { /*snadné*/ }
```



POMALÁ  
metoda

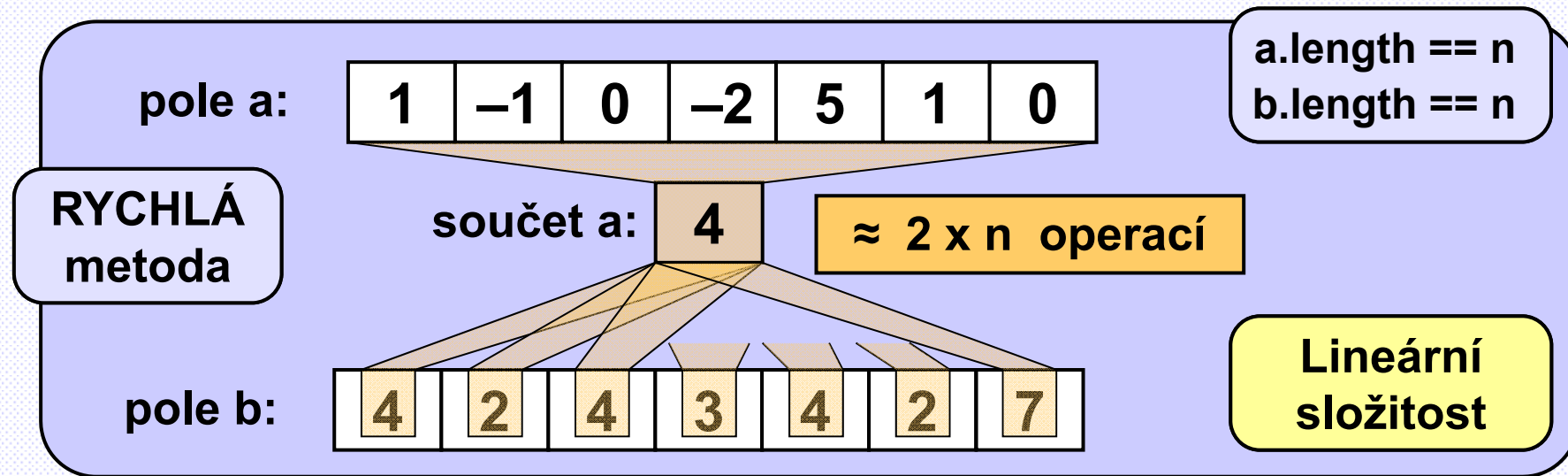
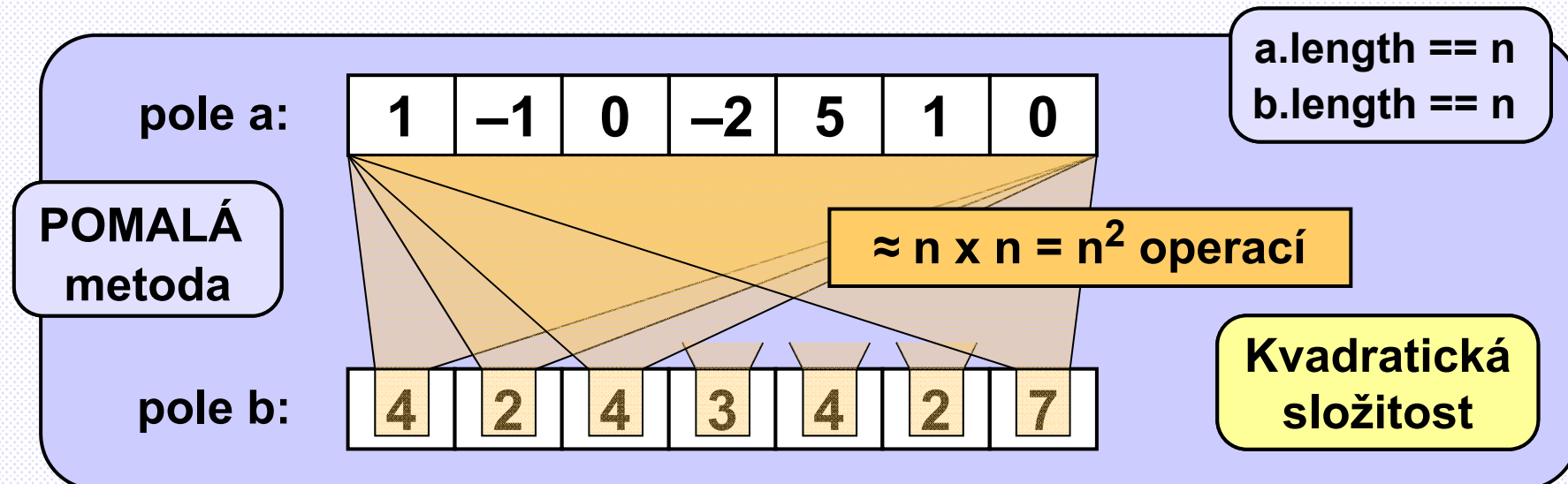
```
count = 0;  
for (int i = 0; i < b.length; i++)  
    if (b[i] == sumArr(a)) count++;  
return count;
```



RYCHLÁ  
metoda

```
count = 0;  
sumOf_b = sumArr(a);  
for (int i = 0; i < b.length; i++)  
    if (b[i] == sumOf_b) count++;  
return count;
```

## Počítání složitosti



## Počítání složitosti

Velikost pole N	POMALÁ metoda operací $N^2$	RYCHLÁ metoda operací $2N$	poměr POMALÁ/RYCHLÁ
11	121	22	5.5
21	441	42	10.5
51	2 601	102	25.5
101	10 201	202	50.5
201	40 401	402	100.5
501	251 001	1 002	250.5
1 001	1 002 001	2 002	500.5
2 001	4 004 001	4 002	1 000.5
5 001	25 010 001	10 002	2 500.5
1 000 001	1 000 002 000 001	2 000 002	500 000.5

Tab. 2



## Počítání složitosti

Velikost pole N	Poměr rychlostí řešení 1. úlohy	Poměr rychlostí řešení 2. úlohy
11	1.33	5.5
21	1.33	10.5
51	1.33	25.5
101	1.33	50.5
201	1.33	100.5
501	1.33	250.5
1 001	1.33	500.5
2 001	1.33	1 000.5
5 001	1.33	2 500.5
1 000 001	1.33	500 000.5

Tab. 3

## Příklady

### Hledání v seřazeném poli — lineární, POMALÉ

dané pole

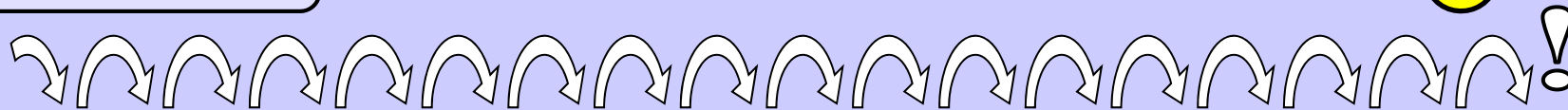
Seřazené pole: →

velikost = N

363	369	388	603	638	693	803	833	836	839	860	863	938	939	966	968	983	993
-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----

najdi 993 !

testů: N ☹️



363	369	388	603	638	693	803	833	836	839	860	863	938	939	966	968	983	993
-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----

najdi 363 !

testů: 1 😊

363	369	388	603	638	693	803	833	836	839	860	863	938	939	966	968	983	993
-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----

# Příklady

## Hledání v seřazeném poli — binární, RYCHLÉ



najdi 863 !

363	369	388	603	638	693	803	833	<del>836</del>	839	860	863	938	939	966	968	983	993
<del>363</del>	<del>369</del>	<del>388</del>	<del>603</del>	<del>638</del>	<del>693</del>	<del>803</del>	<del>833</del>		839	860	863	938	939	966	968	983	993

2 testy

2 testy

839	860	863	938	<del>939</del>	966	968	983	993
839	860	863	938		<del>966</del>	<del>968</del>	<del>983</del>	<del>993</del>

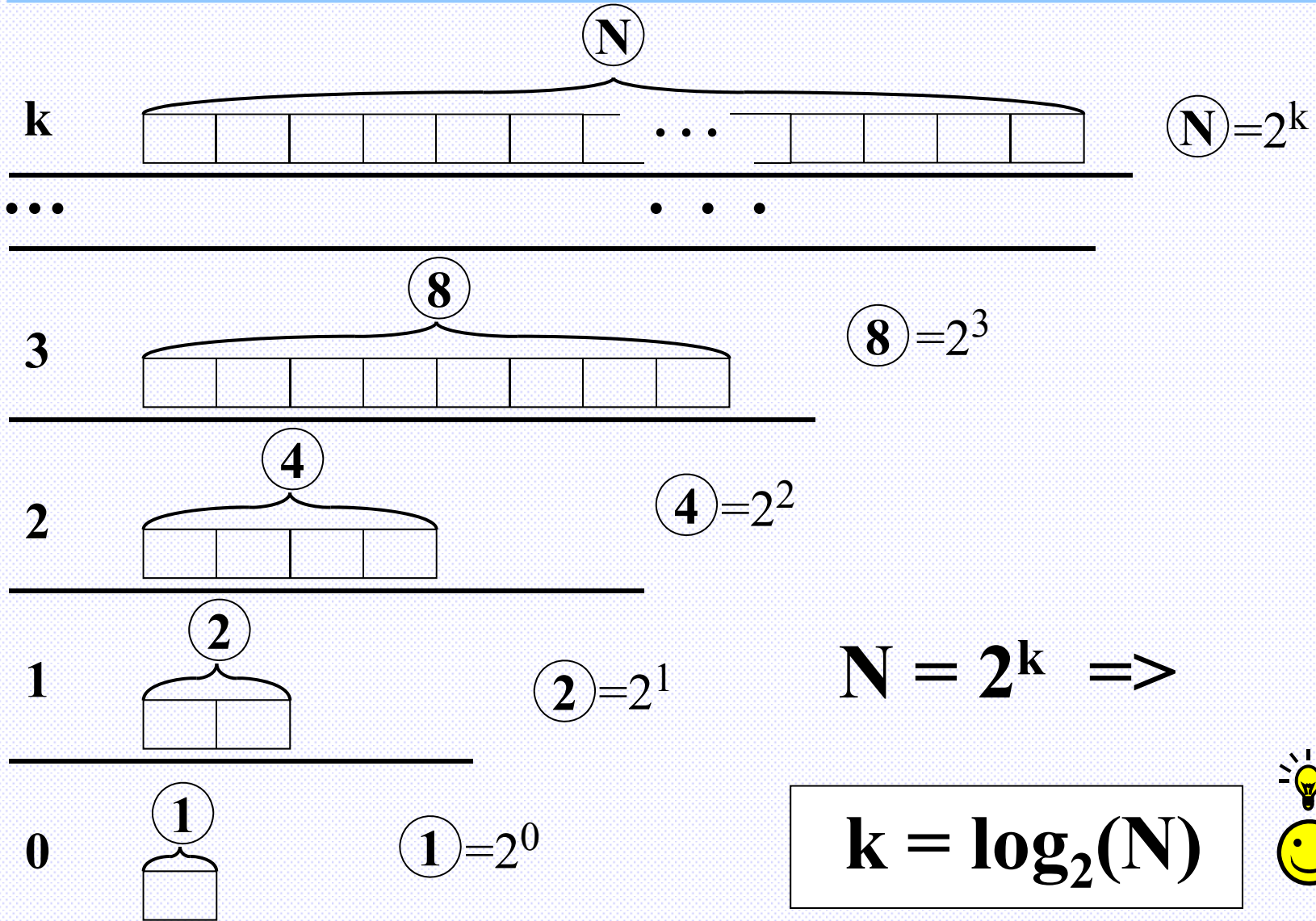
2 testy

839	<del>860</del>	863	938
<del>839</del>		863	938

1 test











<del>863</del>	938
----------------	-----

# Exponent, logarimus a půlení intervalu





## Počítání složitosti

Velikost pole	Počet testů					poměr   
	lineární hledání — případ			binární hledání nejhorší případ		
	nejlepší	nejhorší	průměrný			
5	1	5	3	5	0.6	
10	1	10	5.5	7	0.79	
20	1	20	10.5	9	1.17	
50	1	50	25.5	11	2.32	
100	1	100	50.5	13	3.88	
200	1	200	100.5	15	6.70	
500	1	500	250.5	17	14.74	
1 000		1000	500.5	 	19	26.34
2 000		 2000	 1000.5		21	47.64
5 000	1	5000	2500.5	25	100.02	
1 000 000	1	1 000 000	500 000.5	59	8 474.58	

Tab. 4

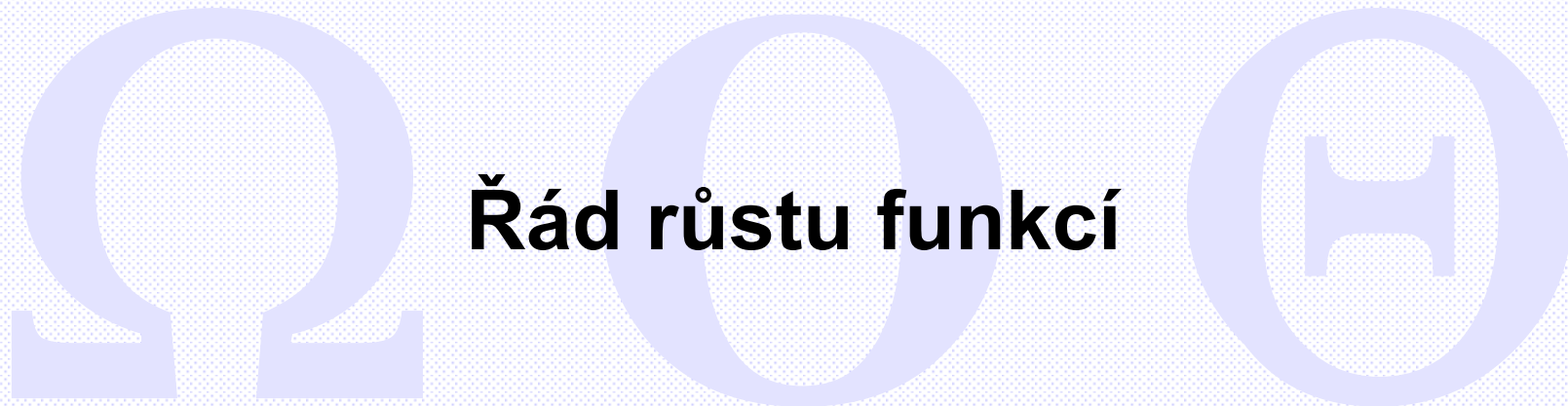
## Počítání složitosti

Doba výpočtu  
pro různé časové složitosti  
s předpokladem, že 1 operace trvá  $1 \mu\text{s}$  ( $10^{-6}$  sec)

složitost	Počet operací					
	10	20	40	60	500	1000
$\log_2 n$	3,3 $\mu\text{s}$	4,3 $\mu\text{s}$	5 $\mu\text{s}$	5,8 $\mu\text{s}$	9 $\mu\text{s}$	10 $\mu\text{s}$
$n$	10 $\mu\text{s}$	20 $\mu\text{s}$	40 $\mu\text{s}$	60 $\mu\text{s}$	0,5 ms	1 ms
$n \log_2 n$	33 $\mu\text{s}$	86 $\mu\text{s}$	0,2 ms	0,35 ms	4,5 ms	10 ms
$n^2$	0,1 ms	0,4 ms	1,6 ms	3,6 ms	0,25 s	1 s
$n^3$	1 ms	8 ms	64 ms	0,2 s	125 s	17 min
$n^4$	10 ms	160 ms	2,56 s	13 s	17 hod	11,6 dnů
$2^n$	1 ms	1 s	12,7 dnů	36000 let	$10^{137}$ let	$10^{287}$ let
$n!$	3,6 s	77000 let	$10^{34}$ let	$10^{68}$ let	$10^{1110}$ let	$10^{2554}$ let

Tab. 5

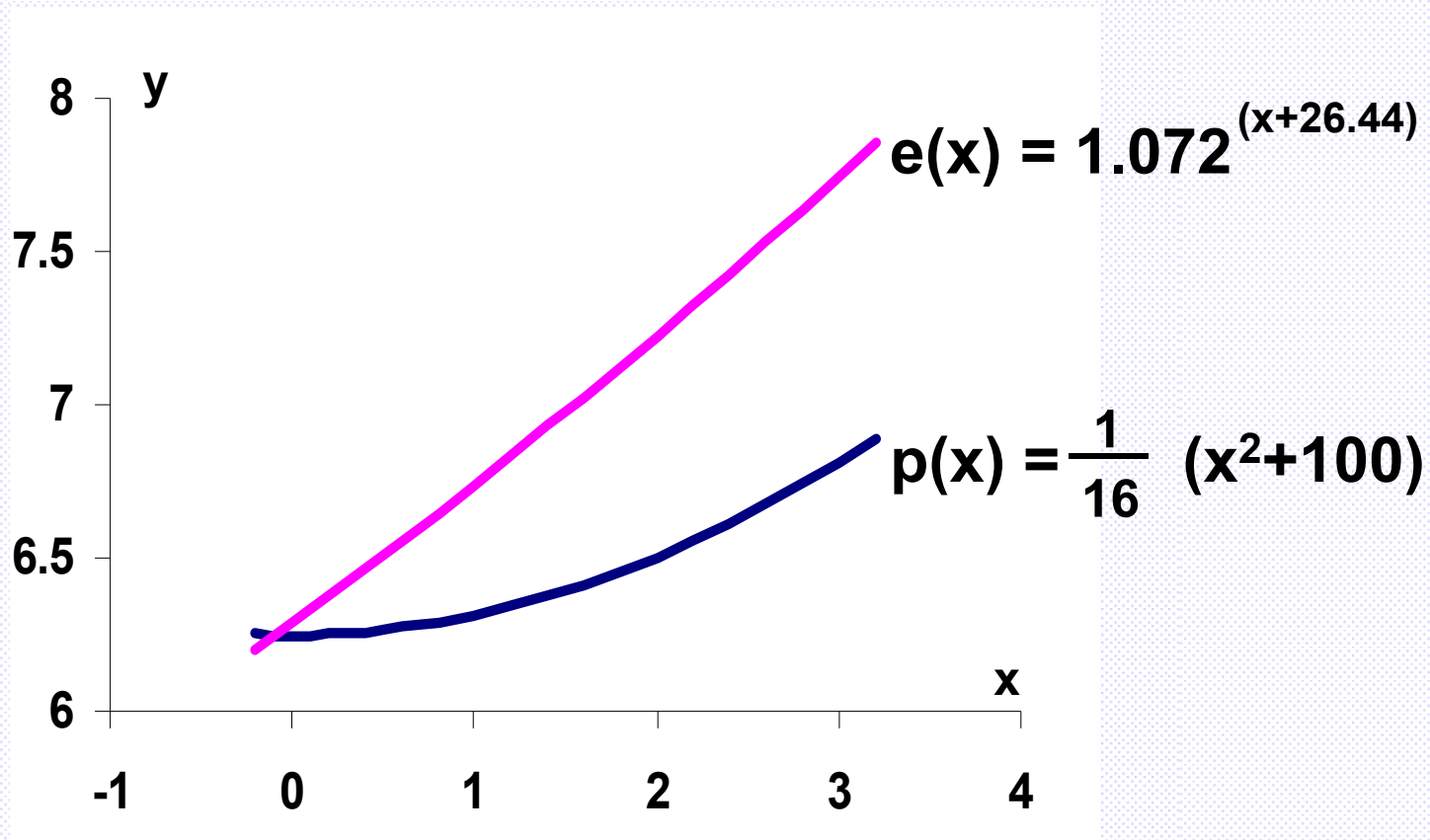
## Řád růstu funkcí



Řád růstu funkcí



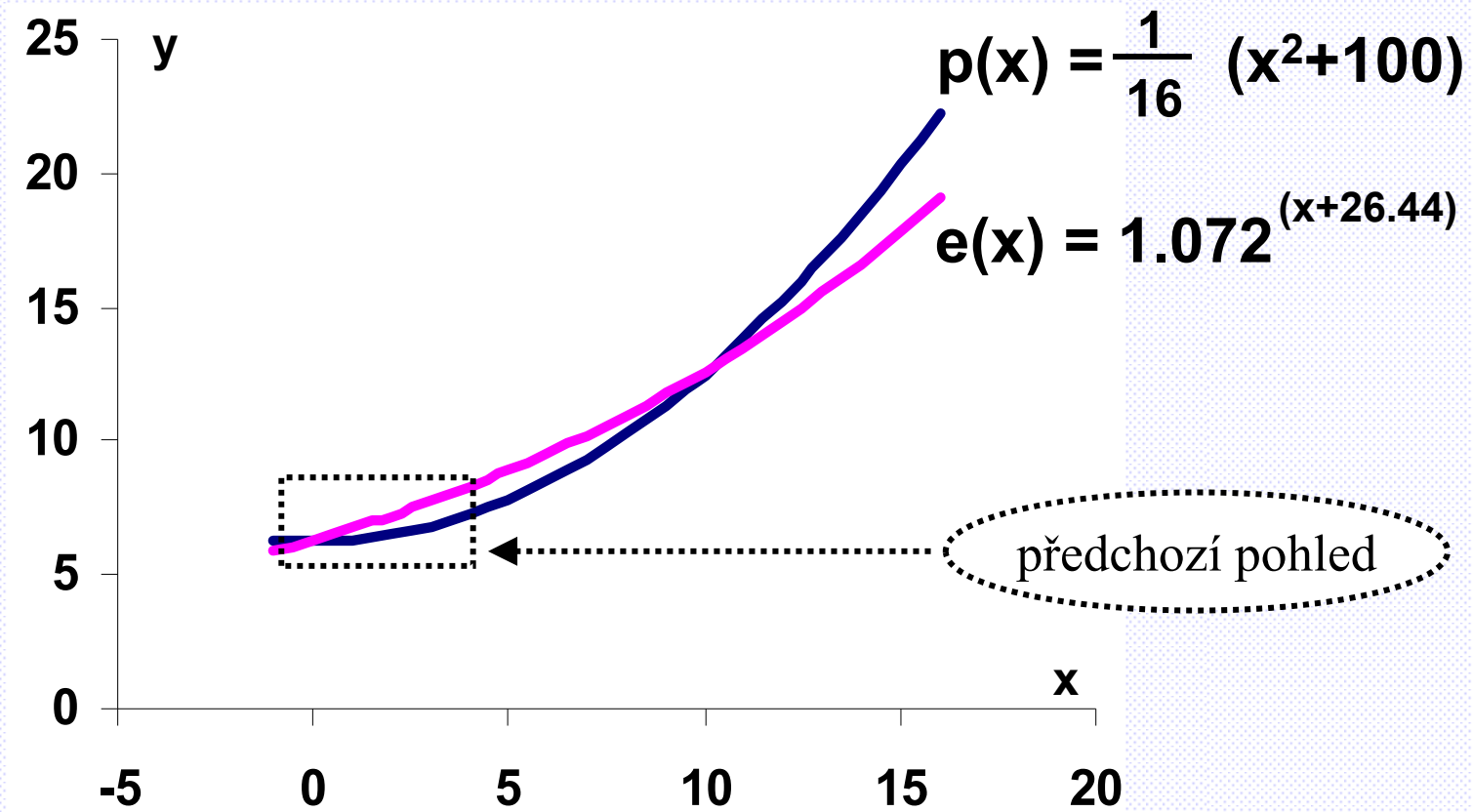
## Řád růstu funkcí





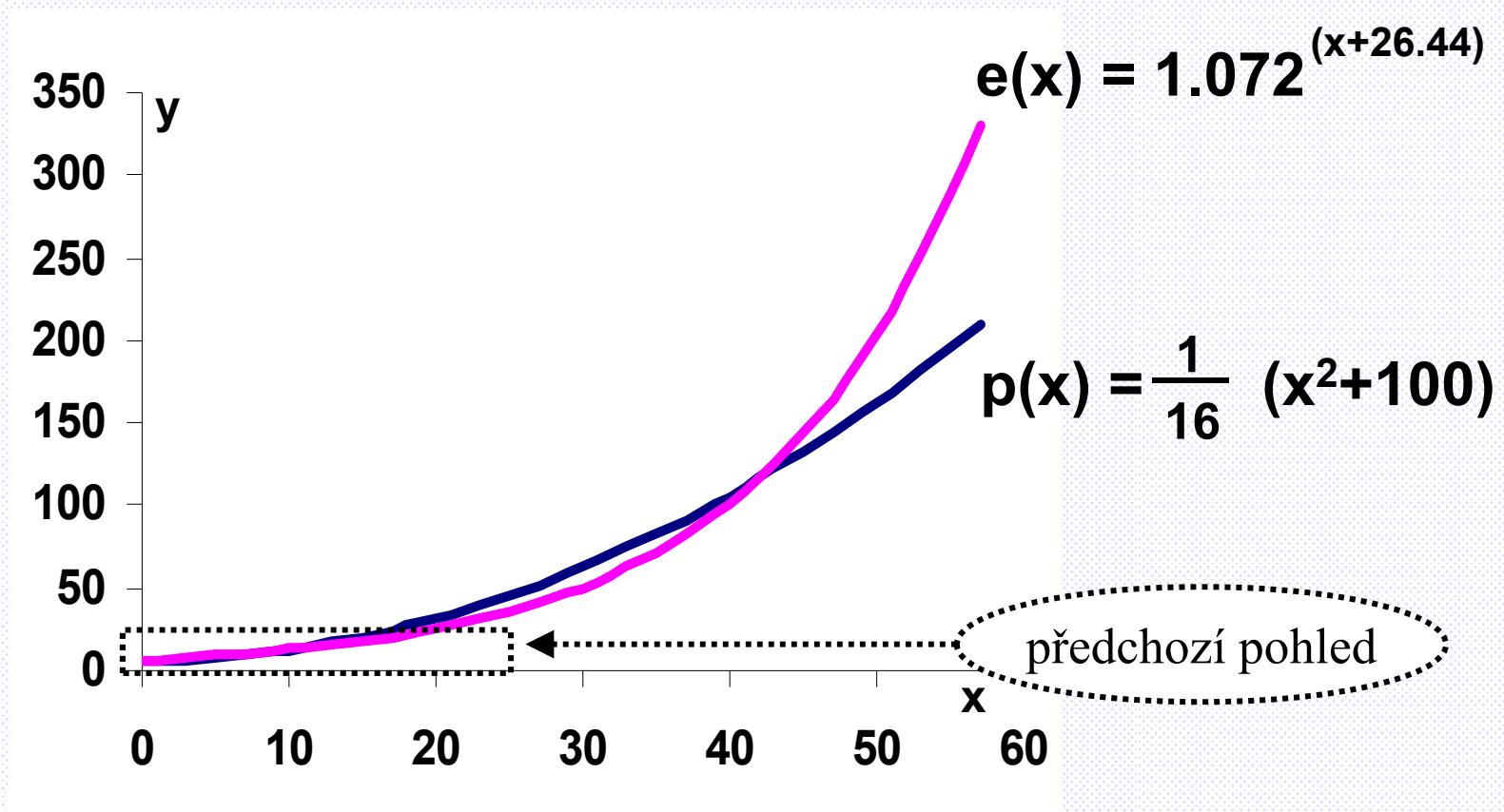
## Řád růstu funkcí

Zoom out! :



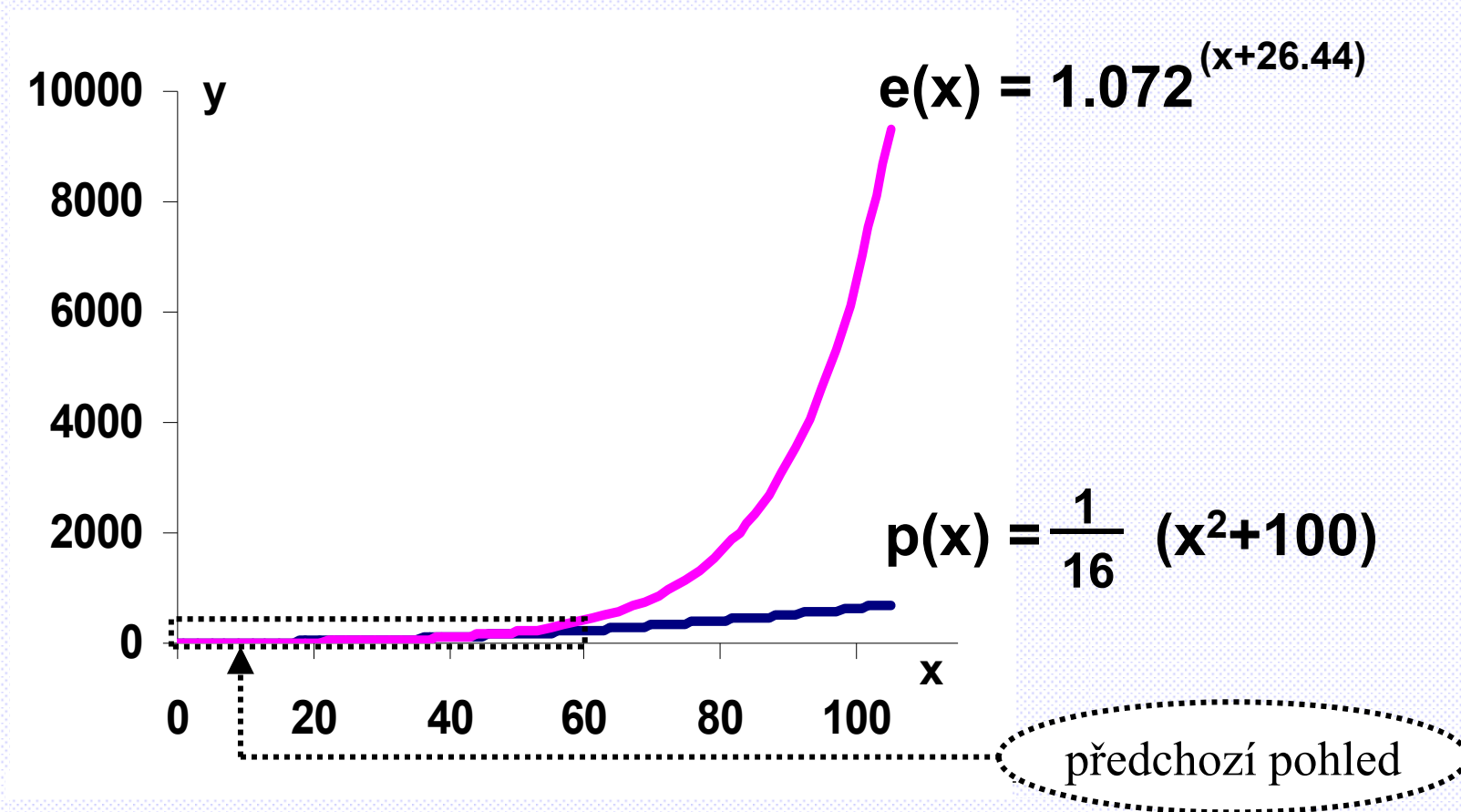
## Řád růstu funkcí

Zoom out! :



## Řád růstu funkcí

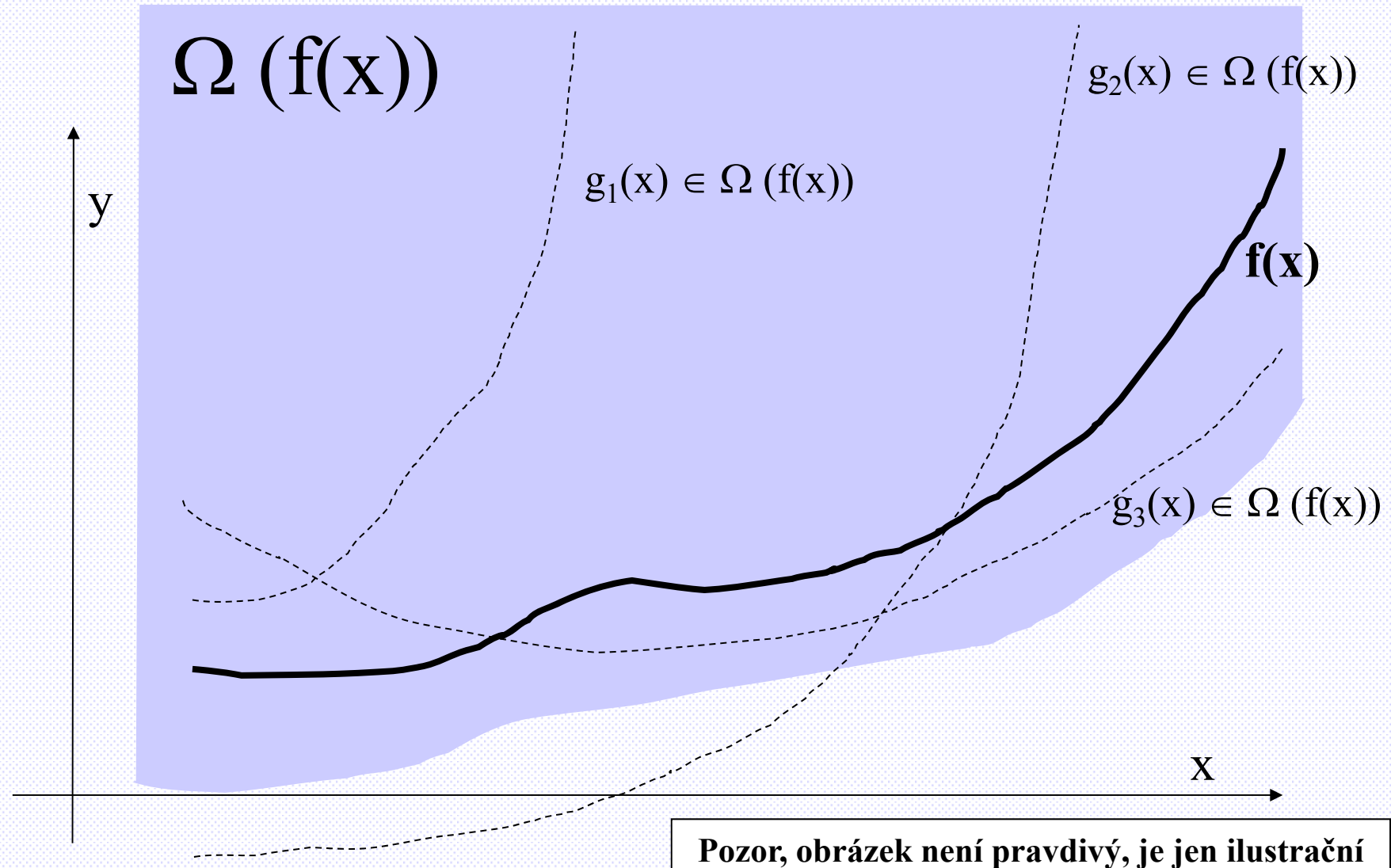
Zoom out! :



atd:...  $e(1000) = 9843181236605408906547628704342.9$

$p(1000) = 62506.25 \dots$

## Řád růstu funkcí





## Řád růstu funkcí

$\Omega(f(x))$

$\Omega$  Omega

**V množině  $\Omega(f(x))$   
se octne každá funkce  $g(x)$ , která od určitého bodu  $x_0$   
(není nijak předem předepsáno, kde by  $x_0$  měl být)**

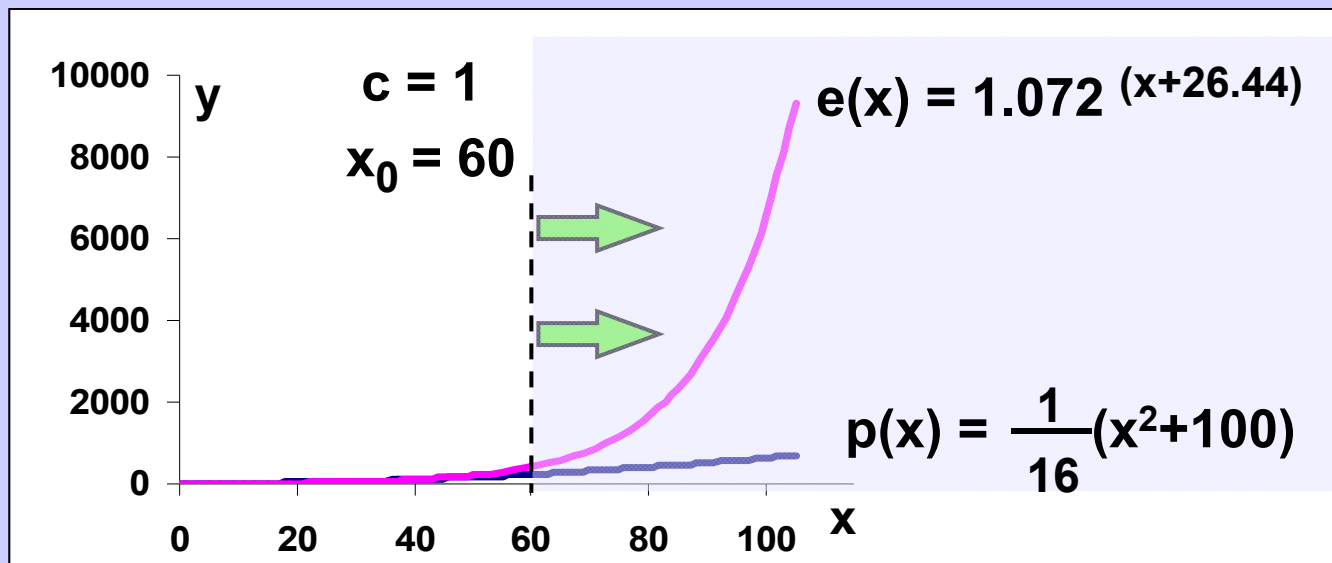
**a) – je už vždy větší než funkce  $f(x)$**

**b) – sice větší než  $f(x)$  není, ale po vynásobení  
nějakou kladnou konstantou  
(hodnota konstanty také není nijak předepsána)  
je už vždy větší než funkce  $f(x)$ .**

**Takže: pokud najdeme nějaké  $x_0$  a  $c > 0$  takové, že  
 $c \cdot g(x) > f(x)$  všude napravo od  $x_0$ , (někdy stačí  $c=1$ )  
je jisto, že  $g(x) \in \Omega(f(x))$**

## Řád růstu funkcí

Takže: pokud najdeme nějaké  $x_0$  a  $c > 0$  takové, že  $c \cdot g(x) > f(x)$  všude napravo od  $x_0$ , (někdy stačí  $c=1$ ) je jisto, že  $g(x) \in \Omega(f(x))$



$$x > 60 \Rightarrow e(x) > p(x), \text{ tj. } 1.072(x+26.44) > \frac{1}{16}(x^2+100)$$

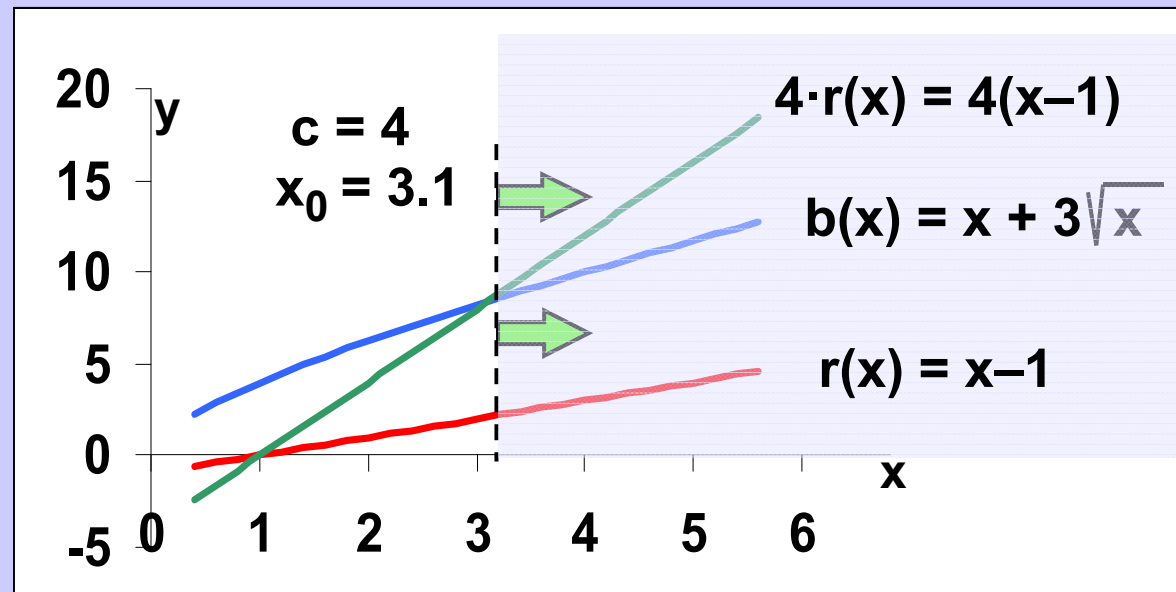
tudíž platí  $e(x) \in \Omega(p(x))$  (ověřte!)

## Řád růstu funkcí

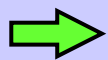
Takže: pokud najdeme nějaké  $x_0$  a  $c > 0$  takové, že  $c \cdot g(x) > f(x)$  všude napravo od  $x_0$ , (někdy stačí  $c=1$ ) je jisto, že  $g(x) \in \Omega(f(x))$

$$b(x) = x + 3\sqrt{x}$$

$$r(x) = x - 1$$



$$\begin{array}{l} \Rightarrow \\ x > 3.1 \Rightarrow 4 \cdot r(x) > b(x), \text{ tj. } 4(x-1) > x + 3\sqrt{x} \quad (\text{ověřte!}) \end{array}$$



tudíž platí  $r(x) \in \Omega(b(x))$

## Řád růstu funkcí

### Typické ukázky

$$x^2 \in \Omega(x)$$

$$x^3 \in \Omega(x^2)$$

$$x^{n+1} \in \Omega(x^n)$$

$$2^x \in \Omega(x^2)$$

$$2^x \in \Omega(x^3)$$

$$2^x \in \Omega(x^{5000})$$

$$x \in \Omega(\log(x))$$

$$x \cdot \log(x) \in \Omega(x)$$

$$x^2 \in \Omega(x \cdot \log(x))$$

$$2^x \in \Omega(x^{20000})$$

$$x^{20000} \in \Omega(x)$$

$$x \in \Omega(1)$$

vždy

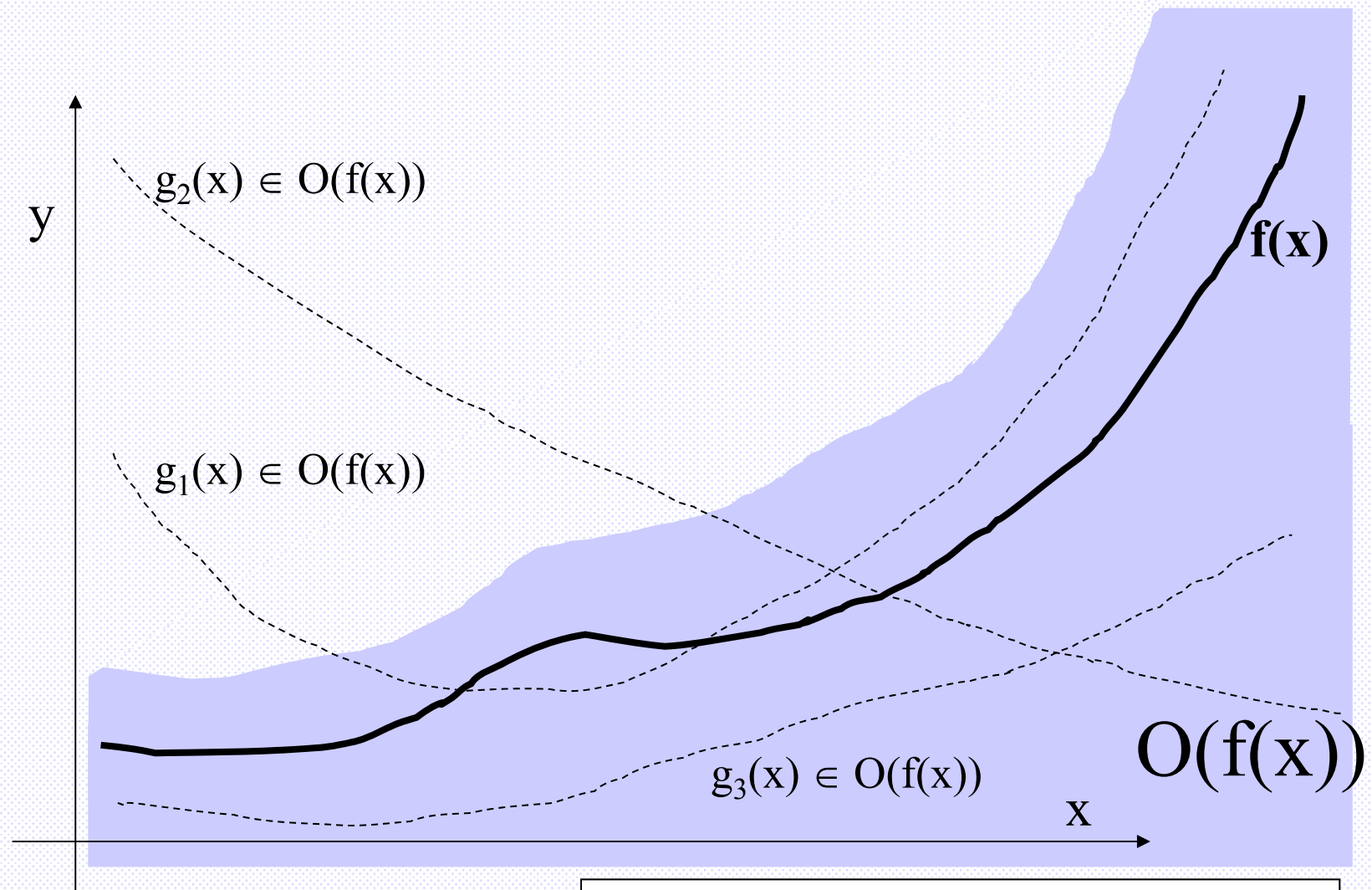
$$f(x) > 1 \Rightarrow f(x) \in \Omega(1)$$

těžko uvěřitelné

$$200\,000 \sqrt{x} \in \Omega(\log(x)^{200\,000})$$



## Řád růstu funkcí



**Pozor, obrázek není pravdivý, je jen ilustrační**

## Řád růstu funkcí

$O(f(x))$

**O Omikron**

**V množině  $O(f(x))$**

**se octne každá funkce  $g(x)$ , která od určitého bodu  $x_0$   
(není nijak předem předepsáno, kde by  $x_0$  měl být)**

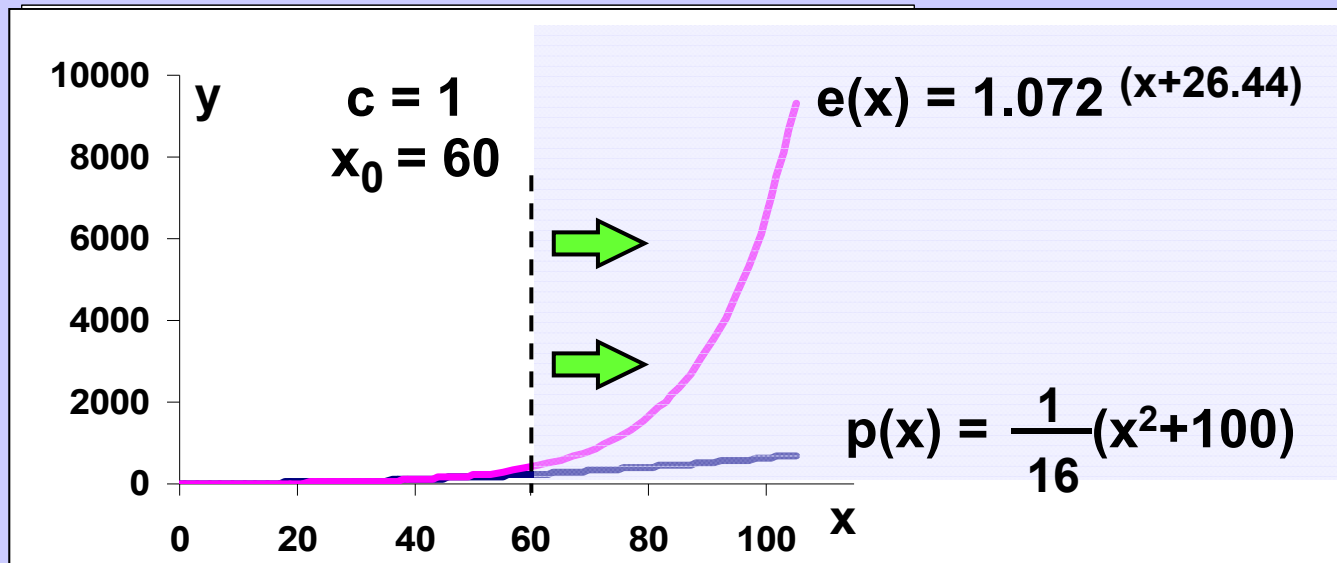
**a) – je už vždy menší než funkce  $f(x)$**

**b) – sice menší než  $f(x)$  není, ale po vynásobení  
nějakou kladnou konstantou (asi  $< 1$  😊)  
(hodnota konstanty také není nijak předepsána)  
je už vždy menší než funkce  $f(x)$ .**

**Takže: pokud najdeme nějaké  $x_0$  a  $c > 0$  takové, že  
 $c \cdot g(x) < f(x)$  všude napravo od  $x_0$ , (někdy stačí  $c=1$ )  
je jisto, že  $g(x) \in O(f(x))$**

## Řád růstu funkcí

Takže: pokud najdeme nějaké  $x_0$  a  $c > 0$  takové, že  $c \cdot g(x) < f(x)$  všude napravo od  $x_0$ , (někdy stačí  $c=1$ ) je jisto, že  $g(x) \in O(f(x))$



$x > 60 \Rightarrow p(x) < e(x)$ , tj.  $\frac{1}{16}(x^2+100) < 1.072^{(x+26.44)}$

tudíž platí

$$p(x) \in O(e(x))$$

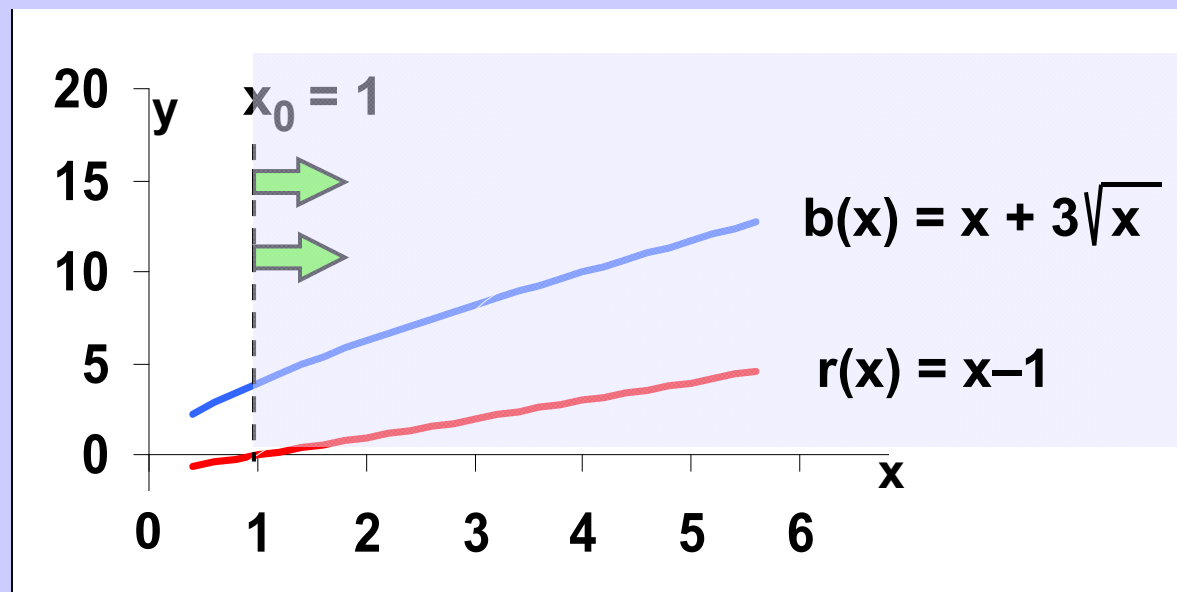
ověřte!

## Řád růstu funkcí

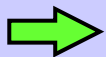
Takže: pokud najdeme nějaké  $x_0$  a  $c > 0$  takové, že  $c \cdot g(x) < f(x)$  všude napravo od  $x_0$ , (někdy stačí  $c=1$ ) je jisto, že  $g(x) \in O(f(x))$

$$b(x) = x + 3\sqrt{x}$$

$$r(x) = x - 1$$



$$x > 1 \Rightarrow r(x) < b(x), \text{ tj. } x - 1 < x + 3\sqrt{x}$$



tudíž platí  $r(x) \in O(b(x))$



## Řád růstu funkcí

$$f \in \Omega(g) \iff g \in O(f)$$

$$x \in O(x^2)$$

$$x^2 \in O(x^3)$$

$$x^n \in O(x^{n+1})$$

$$x^2 \in O(2^x)$$

$$x^3 \in O(2^x)$$

$$x^{5000} \in O(2^x)$$

$$\log(x) \in O(x)$$

$$x \in O(x \cdot \log(x))$$

$$x \cdot \log(x) \in O(x^2)$$

$$x^{20000} \in O(2^x)$$

$$x \in O(x^{20000})$$

$$1 \in O(x)$$

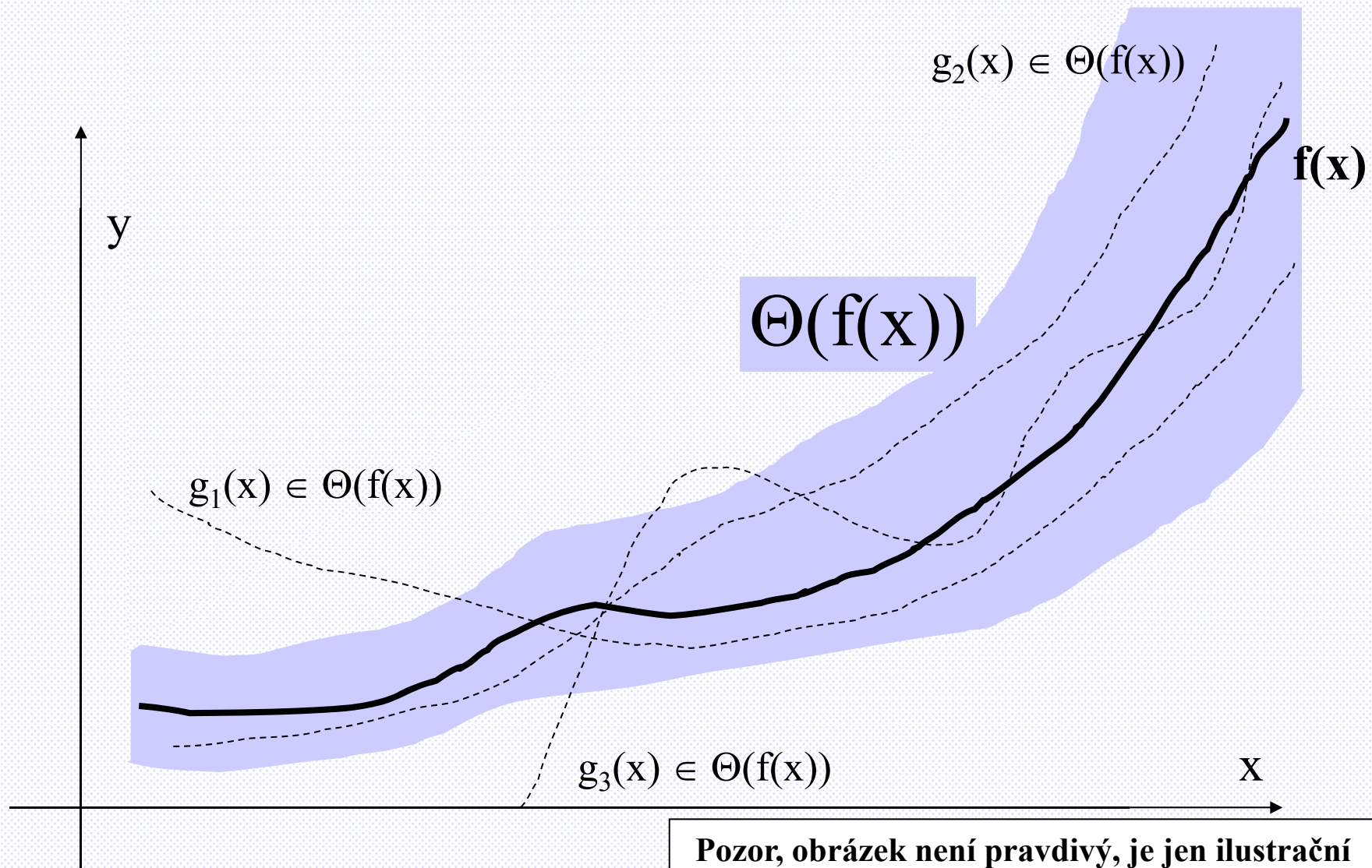
vždy

$$f(x) > 1 \implies 1 \in O(f(x))$$

těžko uvěřitelné

$$\log(x)^{200\,000} \in O(\sqrt[200\,000]{x})$$

## Řád růstu funkcí



Pozor, obrázek není pravdivý, je jen ilustrační

## Řád růstu funkcí

$$\Theta(f(x)) = \Omega(f(x)) \cap O(f(x))$$

### $\Theta$ Theta

V množině  $\Theta(f(x))$  se ocitne každá funkce  $g(x)$ , která spadá jak do  $\Omega(f(x))$  tak do  $O(f(x))$ .

$$f(x) \in \Theta(g(x)) \iff g(x) \in \Theta(f(x))$$

## Řád růstu funkcí

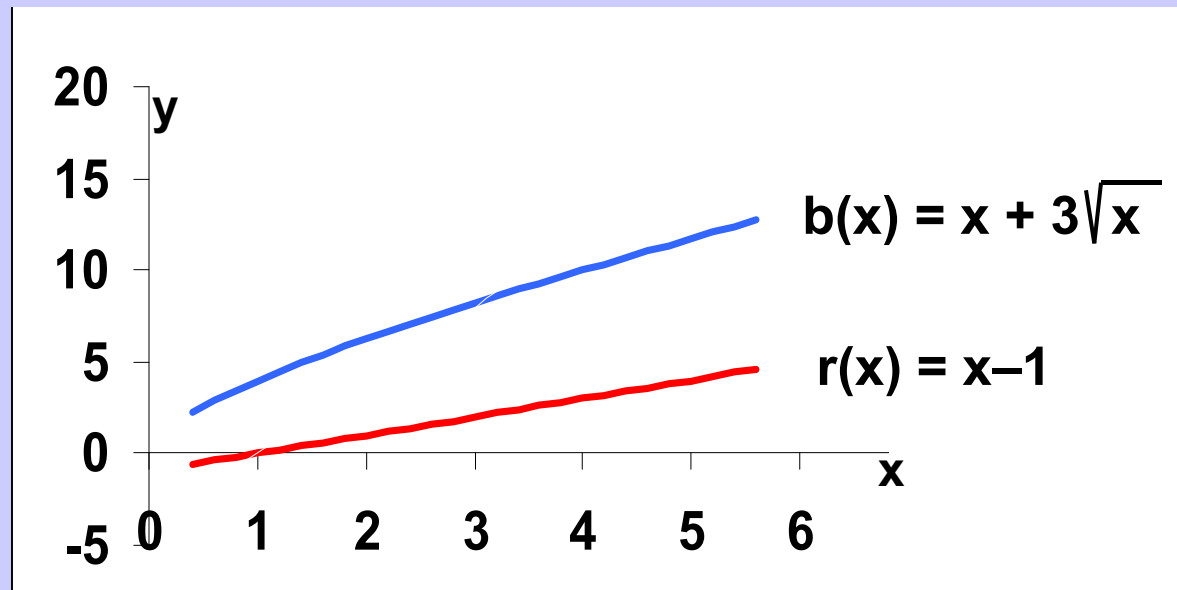
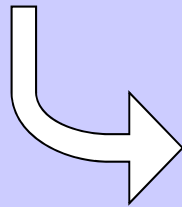
$$f(x) \in \Theta(g(x)) \Leftrightarrow g(x) \in \Theta(f(x))$$

$$b(x) = x + 3\sqrt{x}$$

$$r(x) = x - 1$$

$$r(x) \in \Omega(b(x))$$

$$r(x) \in O(b(x))$$



$$r(x) \in \Theta(b(x))$$

$$b(x) \in \Theta(r(x))$$



## Řád růstu funkcí

### Pravidla

1.  $(a > 0) \Leftrightarrow \Theta(f(x)) = \Theta(a \cdot f(x))$
2.  $g(x) \in O(f(x)) \Leftrightarrow \Theta(f(x)) = \Theta(f(x) + g(x))$

### Slovy

1. Multiplikativní konstanta  
nemá vliv na náležení do množiny  $\Theta(f(x))$ .
2. Přičtení/odečtení „menší“ funkce  
nemá vliv na náležení do množiny  $\Theta(f(x))$ .

### Příklady

$$1.8x + 600 \cdot \log_2(x) \in \Theta(x)$$

$$x^3 + 7x^{1/2} + 5(\log_2(x))^4 \in \Theta(x^3)$$

$$13 \cdot 3^x + 9x^{12} + 42x^{-4} + 29 \in \Theta(3^x)$$

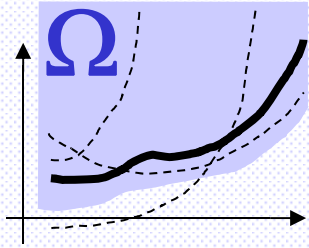
$$4 \cdot 2^n + 3 \cdot 2^{n-1} + 5 \cdot 2^{n/2} \in \Theta(2^n)$$

$$0.1x^5 + 200x^4 + 7x^2 - 3 \in \Theta(x^5)$$

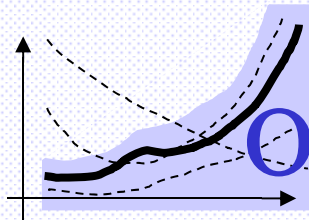
$$-''- \in O(x^5)$$

$$-''- \in \Omega(x^5)$$

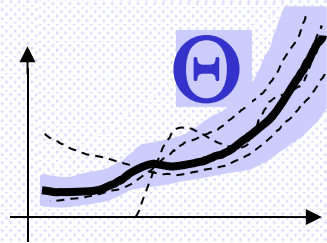
## Řád růstu funkcí



$$\Omega(f(x)) = \{ g(x) ; \exists x_0 > 0, c > 0 \forall x > x_0 : c \cdot f(x) < g(x) \}$$



$$O(f(x)) = \{ g(x) ; \exists x_0 > 0, c > 0 \forall x > x_0 : g(x) < c \cdot f(x) \}$$



$$\Theta(f(x)) = \{ g(x) ; \exists x_0 > 0, c_1 > 0, c_2 > 0 \forall x > x_0 : c_1 \cdot f(x) < g(x) < c_2 \cdot f(x) \}$$

Pozor, obrázky jsou jen ilustrační

## Řád růstu funkcí

### Porovnání rychlosti růstu funkcí

Funkce  $f(x)$  roste asymptoticky rychleji než funkce  $g(x)$ , když

$$f(x) \in \Omega(g(x)) \ \& \ f(x) \notin \Theta(g(x))$$

**Pozor!**

### Porovnání rychlosti algoritmů

Algoritmus A je asymptoticky pomalejší než algoritmus B, když

$$f_A(n) \in \Omega(f_B(n)) \ \& \ f_A(n) \notin \Theta(f_B(n)),$$

kde  $f_A(n)$ , resp.  $f_B(n)$  je funkce určující počet operací, které provede algoritmus A, resp. B, při zpracování dat o rozsahu  $n$ .

## Řád růstu funkcí

### Řád růstu funkce

Řád růstu funkce  $f$  je taková „co nejjednodušší“ funkce  $g$ , pro kterou platí  
$$g(x) \in \Theta(f(x))$$

### Manipulace

Řád růstu funkce  $f$  získáme většinou tak, že zanedbáme

1. Aditivní členy rostoucí pomaleji nebo stejně rychle
2. Multiplikativní konstantu

### Příklady

$ff(n) = 4 \cdot 2^n + 3 \cdot 2^{n-1} + 5 \cdot 2^{n/2} \in \Theta(2^n)$       Řád růstu  $ff(n)$  je  $2^n$

$hh(x) = x + \log_2(x) - \sqrt{x} \in \Theta(x)$       Řád růstu  $hh(x)$  je  $x$



# Asymptotická složitost

## Asymptotická složitost algoritmu

**Asymptotická složitost algoritmu A je řád růstu funkce  $f(n)$ , která charakterizuje počet elementárních operací algoritmu A při zpracování dat o rozsahu  $n$ .  
Přitom uvažujeme nejnáročnější možná data.  
(rozsah dat = celkový počet čísel a znaků v nich)**

**Ve většině případů nehraje roli, zda uvažujeme**

- A) počet všech elementárních operací,**
- B) počet všech elementárních operací nad daty,**
- C) počet testů nad daty.**

**Asymptotická složitost vycházívá táž.**

## Asymptotická složitost

### Asymptotická složitost předložených ukázek

Asymptotická složitost hledání minima v poli o  $n$  prvcích je  $\Theta(\underline{n})$ .  
V obou uvedených případech.

Asymptotická složitost pomalého zjišťování, kolik čísel v poli je rovno součtu jiného pole, je  $\Theta(\underline{n^2})$ .

Asymptotická složitost rychlého zjišťování, kolik čísel v poli je rovno součtu jiného pole, je  $\Theta(\underline{n})$ .

Za předpokladu, že obě pole mají délku  $\underline{n}$ .

Asymptotická složitost lineárního hledání prvku v poli je  $O(\underline{n})$ .

Asymptotická složitost hledání prvku uspořádaném poli pomocí půlení intervalu je  $O(\underline{\log(n)})$ .

Za předpokladu, že pole má délku  $\underline{n}$ .

# Asymptotická složitost

## Úmluvy

### Zjednodušení

Běžně se zkráceným termínem „složitost algoritmu“ rozumí právě výraz „asymptotická složitost algoritmu“

### Zmatení

Běžně se v literatuře neříká  $f(x)$  náleží do  $\Theta(g(x))$ ,  
ale  $f(x)$  je  $\Theta(g(x))$ .

Rovněž se to tak značí:  $f(x) = \Theta(g(x))$

namísto  $f(x) \in \Theta(g(x))$

(a stejně pro  $O$  a  $\Omega$ ).

Chápe se to ale beze změny, v původním smyslu náležení.



## Asymptotická složitost



$$\in \Theta(\text{station wagon})$$



$$\in \Theta(\text{Formula 1 car})$$



$$\in \Omega(\text{Formula 1 car})$$



$$\in O(\text{station wagon})$$



# Různé algoritmy mají různou složitost

**Algoritmus a program není totéž**