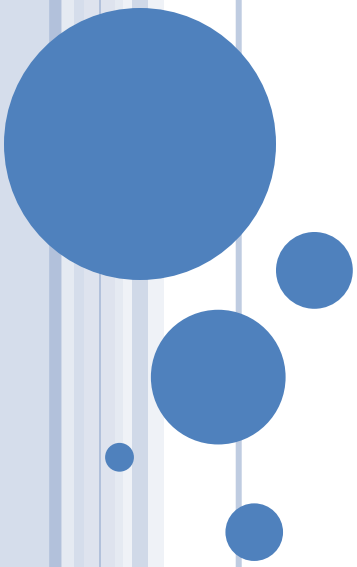


Výpočtová zložitosť algoritmov

Obsah predmetu

- Čo je výpočtová zložitosť algoritmu, problému.
- Určovanie výpočtovej zložitosti.
- Deterministické metódy riešenia problémov.
- Triedy zložitosti problémov.
- Nedeterministické metódy riešenia problémov.

Potreba efektívnych algoritmov

Potrebujeme vôbec efektívne algoritmy, keď máme stále výkonnejšie počítače?

Potreba efektívnych algoritmov

Efektívne algoritmy potrebujeme, lebo spolu s rastom výpočtovej sily bežných počítačov **rastie aj objem dát**, ktoré potrebujeme spracovať.

kapacita pamätí

- 1995 – cca 1GB pevné disky, 1,4 MB diskety
- dnes – pevné disky s kapacitou TB, USB kľúče niekoľko GB

webové stránky

- 1998 – 26 miliónov stránok indexovaných Googlom
- 2000 – miliarda stránok
- dnes – bilióny stránok

Časová výpočtová zložitosť

Ako porovnať dva algoritmy?

- Experimentálne – spustiť výpočet a zistiť, ktorý skončí skôr (len pre konkrétne vstupy)
- Spočítať počet krokov výpočtu – vyjadriť ako funkciu závislú od veľkosti vstupu

Počítanie počtu krokov

Problém

Koľko hviezdičiek sa vypíše podľa nasledujúceho algoritmu, ak premenná $n = 20$?

```
for (int i = 0; i < n; i++) {
  for (int j = i + 1; j < n; j++) {
     print('*');
  }
  println();
}
```

Koľko hviezdičiek sa vypíše, ak n je zadaný vstup?

Počítanie počtu krokov

```
for (int i = 0; i < n; i++) {
  for (int j = i + 1; j < n; j++) {
     print('*');
  }
  println();
}
```

počet hviezdičiek pre zadané *n*

$$(n-1)+(n-2)+\cdots+2+1=\frac{n\cdot(n-1)}{2}$$

Počítanie počtu krokov

Problém

Usporiadajme n čísiel od najmenšieho po najväčšie.
Koľko porovnaní sa vykoná?
Koľko výmen sa vykoná?

Jedno riešenie

```
for (int i = 0; i < n; i++) {
   for (int j = 0; j < n - 1; j++) {
      if (a[j] > a[j+1]) {
         p = a[j];
         a[j] = a[j+1];
         a[j+1] = p;
      }
   }
}
```

Časová výpočtová zložitosť

Nech $k(v)$ je počet krokov výpočtu, ktoré vykoná algoritmus pri spracovaní vstupu v.

- **Časová výpočtová zložitosť algoritmu** je funkcia T taká, že hodnota $T(n)$ je najväčší počet krokov výpočtu, ktoré algoritmus vykoná pri spracovaní ľubovoľného vstupu veľkosti n.

  $T(n) = \max \{ k(v);\ v \text{ je vstup veľkosti } n \}$

Časová výpočtová zložitosť

Nech W_n je množina všetkých vstupov veľkosti n.

Nech $k(v)$ je počet krokov, ktoré vykoná algoritmus pri spracovaní vstupu v veľkosti n.

- **Priemerná časová výpočtová zložitosť algoritmu** je funkcia $\overline{T}$ taká, že hodnota $\overline{T}(n)$ je priemerný počet krokov, ktoré algoritmus vykoná pri spracovaní ľubovoľného vstupu veľkosti n.

$$\overline{T}(n) = \frac{\sum_{v \in W_n} k(v)}{|W_n|}$$

Rády zložitosti

Predpokladajme, že za 1 s sa vykoná cca 1 miliarda inštrukcií. Aký najväčší vstup vie spracovať algoritmus s danou zložitosťou za daný čas?

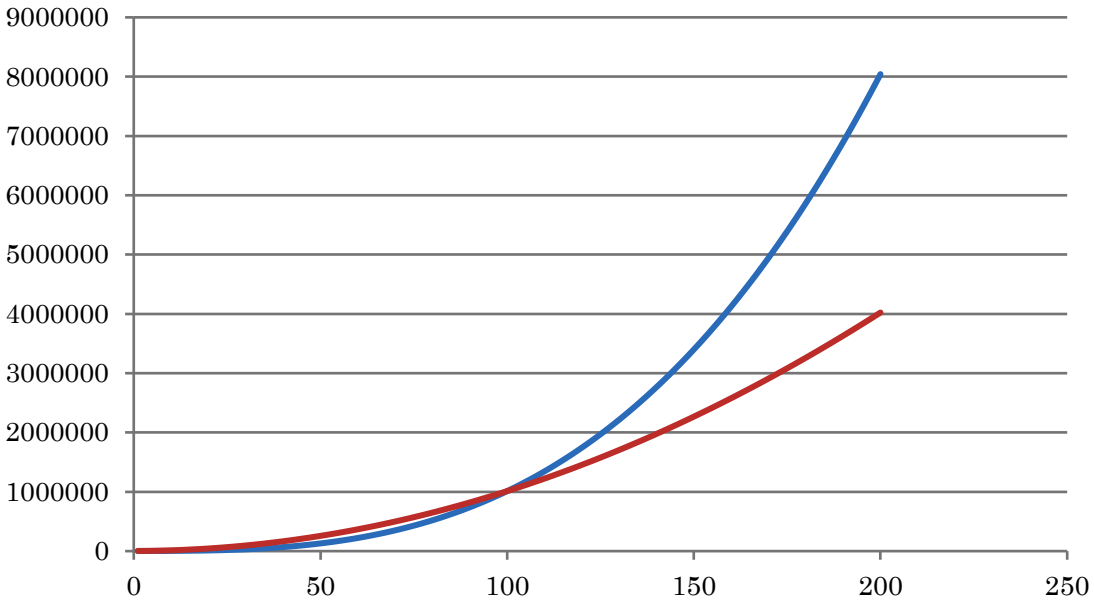
	n	n^2	n^3	2^n	$n!$
milisekunda	1 000 000	1 000	100	20	10
sekunda	1 000 000 000	30 000	1 000	30	12
minúta	veľa	250 000	4 000	35	14
hodina	veľa	2 000 000	15 000	41	15
deň	veľa	9 000 000	44 000	46	16
mesiac	veľa	51 000 000	130 000	51	17
rok	veľa	170 000 000	310 000	54	18
tisícročie	veľa	veľa	3 100 000	64	21

Porovnávanie výpočtovej zložitosti

$f(n) = n^3 + n^2 + 1$

$g(n) = 100\ n^2 + 100\ n$

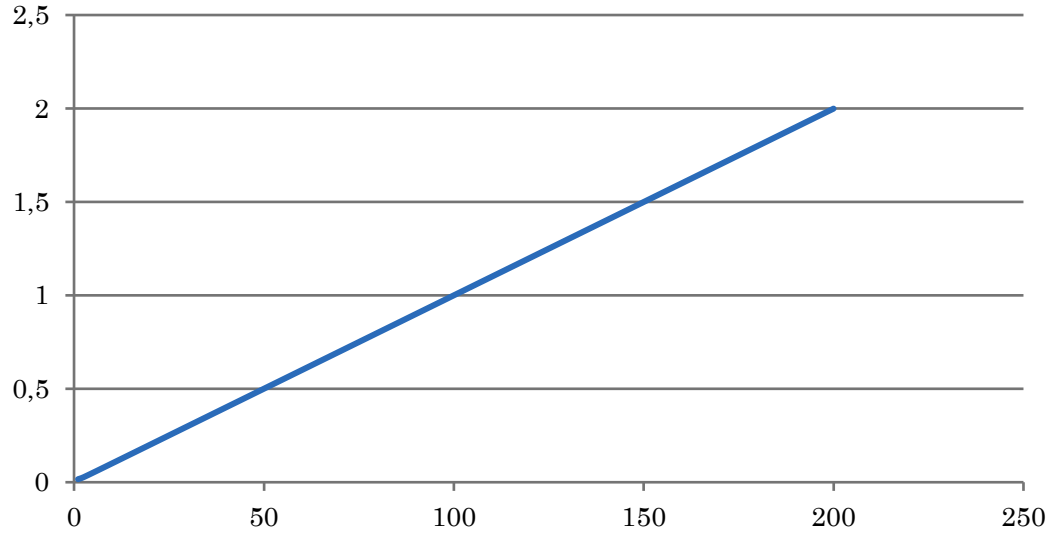
Porovnajme $\frac{f(n)}{g(n)}$

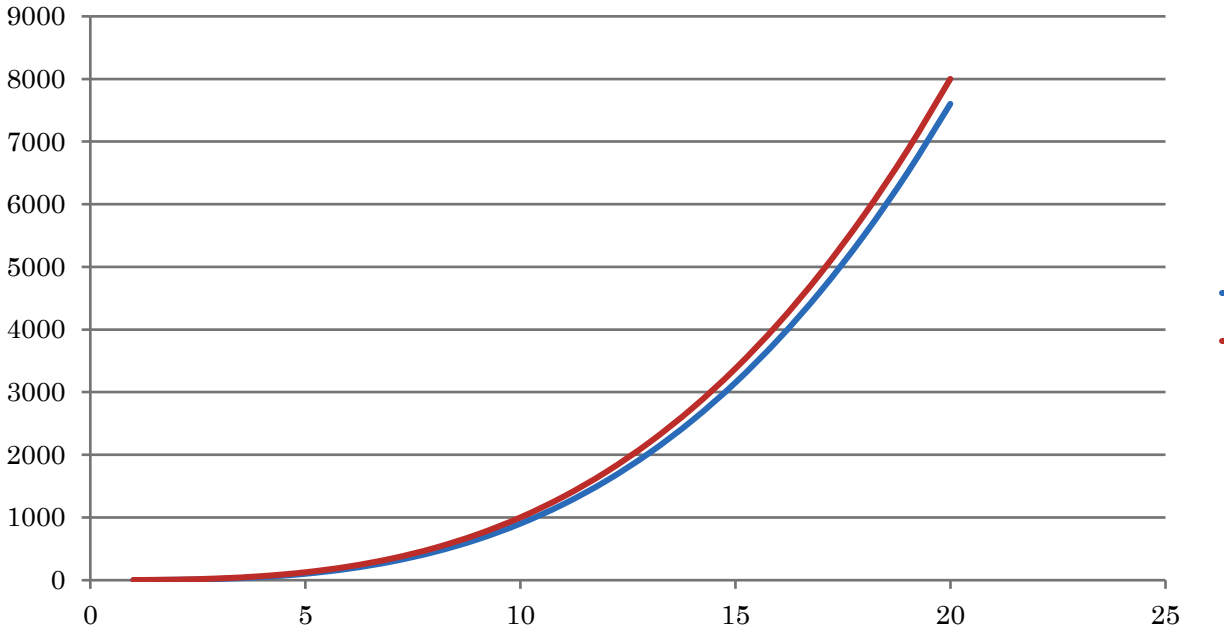


$f(n) = n^3 + n^2 + 1$

$g(n) = 100\ n^2 + 100\ n$

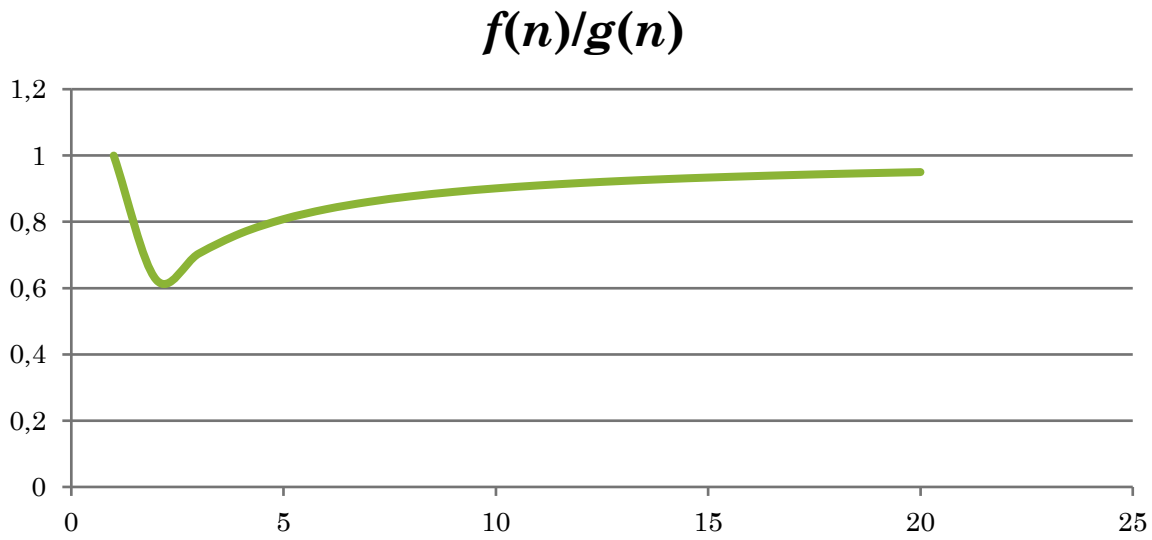
f(n)/g(n)





$$f(n) = n^3 - n^2 + 1$$

$$g(n) = n^3$$



Asymptotická výpočtová zložitosť

Nech f a g sú rastúce funkcie na prirodzených číslach.

- Potom funkcia g je **asymptotickou hornou hranicou** funkcie f, ak existujú kladné konštanty c, m také, že pre všetky $n \geq m$ platí $f(n) \leq c.g(n)$. Píšeme $f(n) = O(g(n))$.
- Potom funkcia g je **asymptotickou dolnou hranicou** funkcie f, ak existujú kladné konštanty c, m také, že pre všetky $n \geq m$ platí $f(n) \geq c.g(n)$. Píšeme $f(n) = \Omega(g(n))$.

Asymptotická výpočtová zložitosť

Nech f a g sú rastúce funkcie na prirodzených číslach.

Ak $f(n) = O(g(n)) = \Omega(g(n))$, tak g je rád rýchlosti stúpania funkcie f.

$f(n) = \Theta(g(n))$ (čítaj theta)

Určovanie časovej výpočtovej zložitosti

- Sekvencia príkazov

P_1; P_2; ...P_k

$$T(n) = T_{P_1}(n) + T_{P_2}(n) + \ldots + T_{P_k}(n)$$

- Vetvenie

if (*podmienka*) then P_1; else P_2;

$$T(n) = T_{\text{podmienka}}(n) + \max\{T_{P_1}(n), T_{P_2}(n)\}$$

- Cyklus

while (*podmienka*) do P_i;

for (*i* od *d* po *h*) P_i;

$$T(n) = \sum(T_{\text{podmienka}}(n) + T_{P_i}(n))$$

Príklady

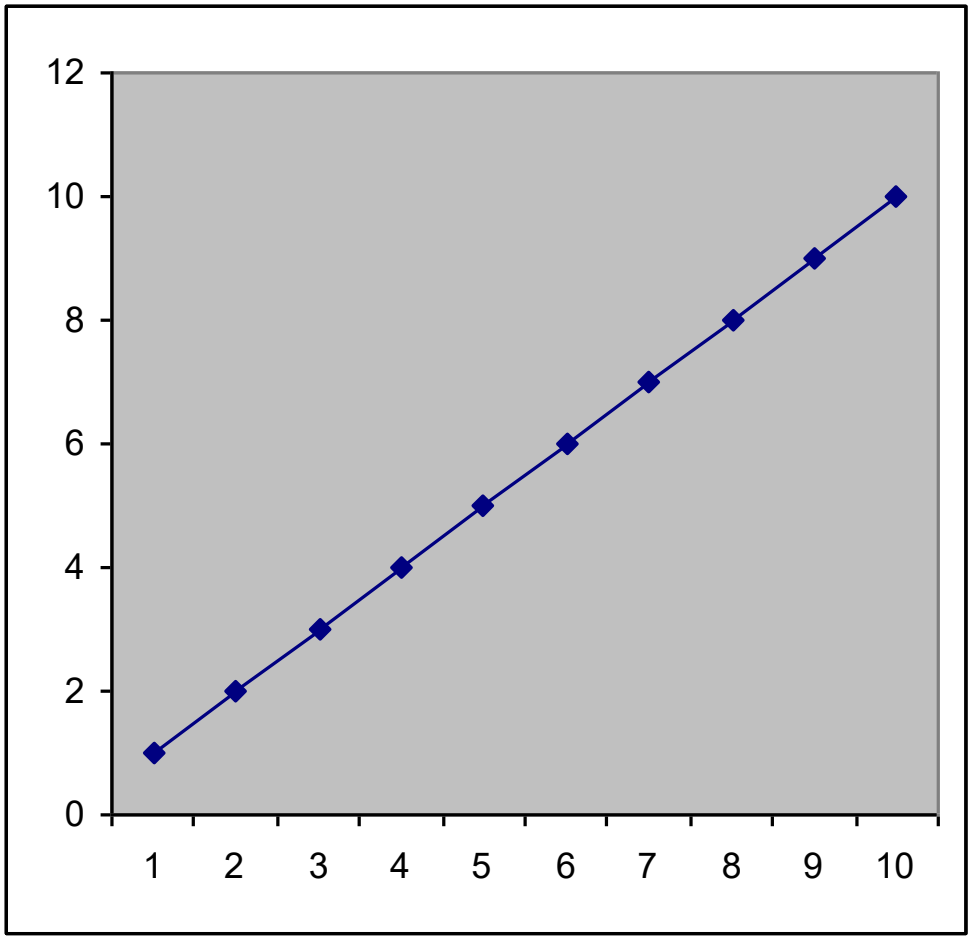
Vypočítajme mocninu m^n. Koľko násobení sa vykoná?

Riešenie 1

```
p = 1;
for (int i = 0; i < n; i++) {
    p = p * m;
}
```

$$T(n) = \sum_{i=0}^{n-1} 1 = n$$

Zložitosť riešenia 1



Príklady

Vypočítajme mocninu m^n. Koľko násobení sa vykoná?

Riešenie 2

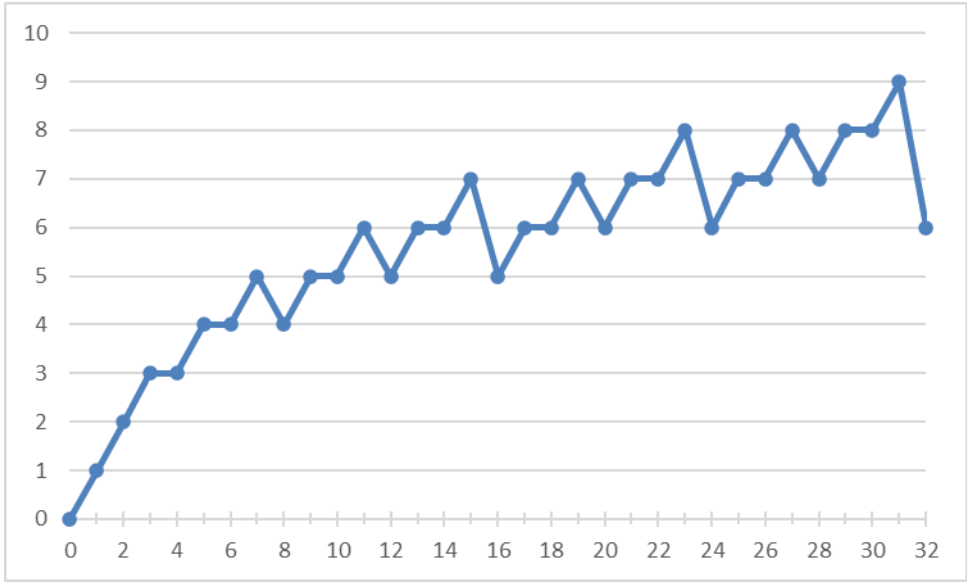
```
p = 1; b = m; e = n;
while (e > 0) {
   if nepárne(e) {
     p = p * b; e = e - 1;
   } else {
     b = b * b; e = e / 2;
   }
}
```

	m	n	p	b	e
0	2	10	1	2	10
1	2	10	1	4	5
2	2	10	4	4	4
3	2	10	4	16	2
4	2	10	4	256	1
5	2	10	1024	256	0

Zložitosť riešenia 2

```
p = 1; b = m; e = n;
while (e > 0) {
   if nepárne(e) {
     p = p * b; e = e - 1;
   } else {
     b = b * b; e = e / 2;
   }
}
```

T(0) = 0
T(1) = 1
T(2) = 1+T(1) = 2
T(3) = 1+T(2) = 3
T(4) = 1+T(2) = 3
T(5) = 1+T(4) = 4
T(6) = 1+T(3) = 4
T(7) = 1+T(6) = 5
T(8) = 1+T(4) = 4
T(9) = 1+T(8) = 5
T(10) = 1+T(5) = 5
T(11) = 1+T(10) = 6
T(12) = 1+T(6) = 5



Zložitosť riešenia 2

Najhorší prípad:

$$n = 2^k - 1$$

$$k = \log_2(n+1)$$

```
p = 1; b = m; e = n;
while (e > 0) {
   if nepárne(e) {
     p = p * b; e = e - 1;
   } else {
     b = b * b; e = e / 2;
   }
}
```

$$T(2^k - 1) = 1 + T(2^k - 2) = 2 + T(2^{k-1} - 1) =$$
$$= 2 + 1 + T(2^{k-1} - 2) = 2 + 2 + T(2^{k-2} - 1) =$$
$$= 2 + 2 + 1 + T(2^{k-2} - 2) = 2 + 2 + 2 + T(2^{k-3} - 1) =$$
$$= \cdots = (k-1) \cdot 2 + T(2^{k-(k-1)} - 1) = (\mathrm{k} - 1) \cdot 2 + 1$$
$$= 2 \cdot \log_2(n+1) - 1 = O(\log n)$$

Príklady

Usporiadajme n čísiel od najmenšieho po najväčšie.
Koľko porovnaní sa vykoná?

Riešenie 1

```
for (int i = 0; i < n - 1; i++) {
    for (int j = 0; j < n - 1; j++) {
        if (a[j] > a[j+1]) {
            p = a[j];
            a[j] = a[j+1];
            a[j+1] = p;
        }
    }
}
```

Zložitosť riešenia 1

```
for (int i = 0; i < n - 1; i++) {
    for (int j = 0; j < n - 1; j++) {
        if (a[j] > a[j+1]) {
            p = a[j];
            a[j] = a[j+1];
            a[j+1] = p;
        }
    }
}
```

9	3	1	5	6	8	4	7	10	2
3	1	5	6	8	4	7	9	2	10
1	3	5	6	4	7	8	2	9	10
1	3	5	4	6	7	2	8	9	10
1	3	4	5	6	2	7	8	9	10
1	3	4	5	2	6	7	8	9	10
1	3	4	2	5	6	7	8	9	10
1	3	2	4	5	6	7	8	9	10
1	2	3	4	5	6	7	8	9	10
1	2	3	4	5	6	7	8	9	10

$$T(n) = \sum_{i=0}^{n-2} \left(\sum_{j=0}^{n-2} (1) \right) = \sum_{i=0}^{n-2} (n-1) = (n-1)^2 = O(n^2) = \Omega(n^2)$$

$T(n) = (n-1)^2 = O(n^2)$

$(n-1)^2 \leq n^2$ platí pre všetky n

$T(n) = (n-1)^2 = \Omega(n^2)$

$(n-1)^2 \geq c \,.\, n^2 \qquad c = ½$

$(n-1)^2 \geq 1/2 \,.\, n^2$

$2\,(n^2 - 2n+1) \geq n^2$

$2n^2 - 4n+2 - n^2 \geq 0$

$n^2 - 4n+2 \geq 0$

$(n-3{,}4)(n-0{,}6) \geq 0$

$n \geq 3{,}4 \wedge n \geq 0{,}6$

platí pre všetky $n \geq 4$

$$n_{1,2} = \frac{-b \pm \sqrt{b^2-4ac}}{2a}$$

$$n_{1,2} = \frac{4 \pm \sqrt{16-8}}{2} = \frac{4 \pm 2\sqrt{2}}{2}$$

$$= 2 \pm \sqrt{2} \approx \begin{cases} 3{,}4 \\ 0{,}6 \end{cases}$$

Príklady

Usporiadajme n čísiel od najmenšieho po najväčšie.
Koľko porovnaní sa vykoná?

Riešenie 2

```
for (int i = n – 1; i > 0; i--) {
    for (int j = 0; j < i; j++) {
        if (a[j] > a[j+1]) {
            p = a[j];
            a[j] = a[j+1];
            a[j+1] = p;
        }
    }
}
```

Zložitosť riešenia 2

```
for (int i = n - 1; i > 0; i--) {
    for (int j = 0; j < i; j++) {
        if (a[j] > a[j+1]) {
            p = a[j];
            a[j] = a[j+1];
            a[j+1] = p;
        }
    }
}
```

9	3	1	5	6	8	4	7	10	2
3	1	5	6	8	4	7	9	2	10
1	3	5	6	4	7	8	2	9	10
1	3	5	4	6	7	2	8	9	10
1	3	4	5	6	2	7	8	9	10
1	3	4	5	2	6	7	8	9	10
1	3	4	2	5	6	7	8	9	10
1	3	2	4	5	6	7	8	9	10
1	2	3	4	5	6	7	8	9	10
1	2	3	4	5	6	7	8	9	10

$$T(n) = \sum_{i=1}^{n-1} \left(\sum_{j=0}^{i-1} (1) \right) =$$

$$= \sum_{i=1}^{n-1} (i) = 1 + 2 + \cdots + n - 1 = \frac{n(n-1)}{2} = O(n^2)$$

Príklady

Nájdime najväčšie z n čísiel. Koľko porovnaní sa vykoná?

Riešenie 1

```
max = a[0];
for (int i = 1; i < n; i++) {
  if (a[i] > max) {
    max = a[i];
  }
}
```

$$T(n) = n - 1 = O(n)$$

Príklady

Nájdime najväčšie z n čísiel. Koľko porovnaní sa vykoná?

Riešenie 2 – rozdeľuj a panuj

```
function max(z, k) {
  int m1, m2, s;
  if (k == z) {
     return a[k];
  } else {
     s = (z + k) / 2;
     m1 = max(z, s);
     m2 = max(s + 1, k);
     if (m1 > m2) {
        return m1;
     } else {
        return m2;
     }
  }
}
```

Zložitosť riešenia 2

```
function max(z, k) {
  int m1, m2, s;
  if (k == z) {
     return a[k];
  } else {
     s = (z + k) / 2;
     m1 = max(z, s);
     m2 = max(s + 1, k);
     if (m1 > m2) {
        return m1;
     } else {
        return m2;
     }
  }
}
```

$$T(1) = 0$$
$$T(n) = T\left(\frac{n}{2}\right) + T\left(n - \frac{n}{2}\right) + 1$$

$$T(2) = T(1)+T(1)+1 = 1$$
$$T(3) = T(1)+T(2)+1 = 2$$
$$T(4) = T(2)+T(2)+1 = 3$$
$$T(5) = T(2)+T(3)+1 = 4$$
$$T(6) = T(3)+T(3)+1 = 5$$
$$T(7) = T(3)+T(4)+1 = 6$$
$$T(8) = T(4)+T(4)+1 = 7$$

Zložitosť riešenia 2

$$T(1) = 0$$
$$T(n) = T\left(\frac{n}{2}\right) + T\left(n - \frac{n}{2}\right) + 1$$

$$T(2^k) = 2 \cdot T(2^{k-1}) + 1$$
$$T(2^{k-1}) = 2 \cdot T(2^{k-2}) + 1$$
$$T(2^{k-2}) = 2 \cdot T(2^{k-3}) + 1$$

Pre $n = 2^k$

$$T(2^k) = 2 \cdot (2 \cdot T(2^{k-2}) + 1) + 1 = 2^2 T(2^{k-2}) + 2 + 1 =$$
$$= 2^2(2 \cdot T(2^{k-3}) + 1) + 2 + 1 = 2^3 T(2^{k-3}) + 2^2 + 2 + 1 = \cdots =$$
$$= 2^k T(2^{k-k}) + 2^{k-1} + 2^{k-2} + \cdots + 2^1 + 2^0 =$$
$$= 2^{k-1} + 2^{k-2} + \cdots + 2^1 + 2^0 =$$
$$= 2^0 \cdot \frac{2^k - 1}{2 - 1} = 2^k - 1 = n - 1 = O(n)$$

Pamäťová výpočtová zložitosť

Nech $p(v)$ je veľkosť pamäte, ktorá sa obsadí vykonávaním algoritmu pri spracovaní vstupu v.

- **Pamäťová (priestorová) výpočtová zložitosť algoritmu** je funkcia S taká, že hodnota $S(n)$ je najväčšia veľkosť pamäte, ktorá sa obsadí vykonávaním algoritmu pri spracovaní ľubovoľného vstupu veľkosti n.

  $S(n) = \max \{ p(v);\ v \text{ je vstup veľkosti } n \}$