

KOMPLEKSNOST ALGORITMOV

- časovna kompleksnost
- prostorska kompleksnost

- Velikostni red (časovne enote)
- Dejanski pričakovani čas (sekunde)



KOMPLEKSNOŠT ALGORITMOV

- velikostni red kompleksnosti
 - asimptotična zgornja meja O
 - spodnja meja Ω
 - stroga meja θ



ZGORNJA MEJA ČASOVNE KOMPLEKSNOTI

- definicija

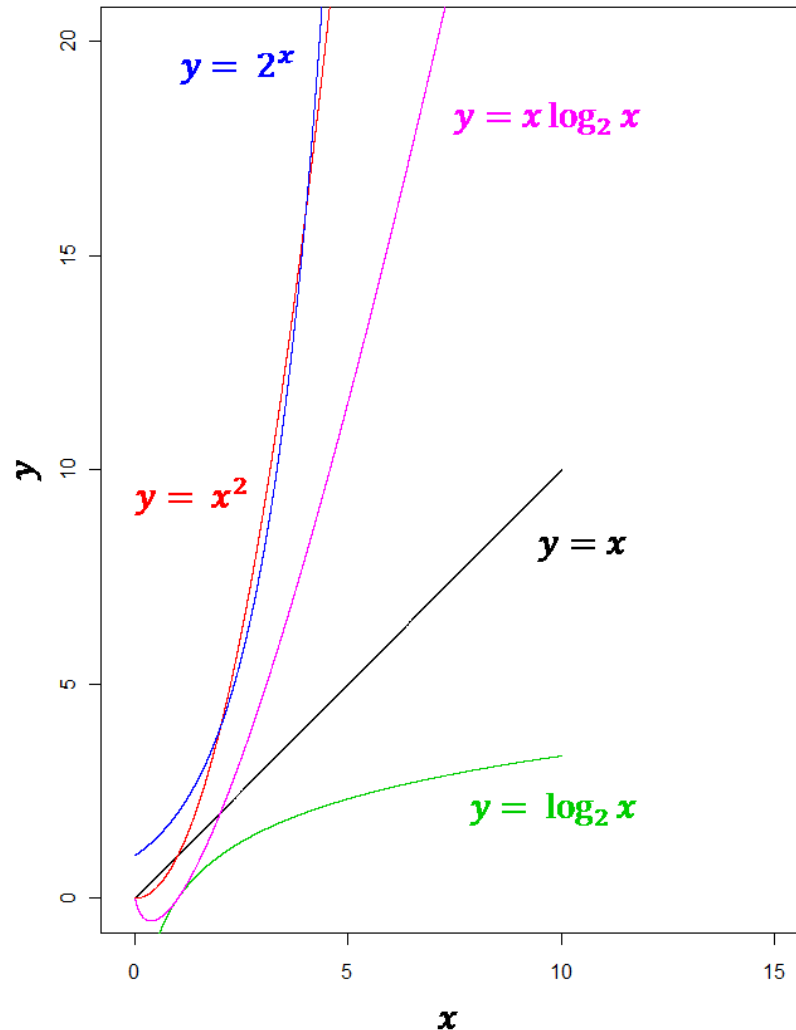
$$T(n) = O(g(n)) \leftrightarrow \exists c, n_0 > 0 : n > n_0 \rightarrow c \cdot g(n) \geq T(n) \geq 0$$

- pri dovolj velikem n je kompleksnost našega programa navzgor omejena s funkcijo $g(n)$
- poljubna konstanta c



POGOSTE KOMPLEKSNOSTI

$\log n, n, n \log n, n^2, n^3, n^4, 2^n, n!, n^n$



RAČUNANJE Z $O()$

- eliminacija konstante

$$c > 0 \rightarrow O(c \cdot f(n)) = O(f(n))$$

- vsota

$$O(f(n)) + O(g(n)) = O(\max(f(n), g(n)))$$

- prevladujoča funkcija

$$\forall n > n_0 : f(n) > g(n) \rightarrow O(f(n)) + O(g(n)) = O(f(n))$$

- produkt

$$O(f(n)) \cdot O(g(n)) = O(f(n) \cdot g(n))$$

- tranzitivnost

$$f(n) = O(g(n)), g(n) = O(h(n)) \rightarrow f(n) = O(h(n))$$

- refleksivnost

$$f(n) = O(f(n))$$



PRIMERI

Določi asimptotično zgornjo mejo naslednjih funkcij:

$$17n^3 + 93n^2 + n = O(n^3)$$

$$2n^2 + n \log n = O(n^2)$$

$$3n + n \log n = O(n \log n)$$

$$2^n + 739n^9 = O(2^n)$$

$$(n + \sqrt{n}) \cdot (n \log n + n + 3) = O(n^2 \log n)$$



DOLOČANJE KOMPLEKSNOSTI V PROGRAMU - PRAVILA

Najprej določimo parametre kompleksnosti!

1. **osnovne operacije:** $O(1)$
2. **pri zaporedju ukazov** seštevamo zahtevnosti
3. **pri pogojih** štejemo kompleksnost izračuna pogoja in maksimum vseh možnih izbir
4. **pri zankah** seštejemo kompleksnost izračuna pogoja in enkratne izvedbe zanke ter pomnožimo s številom izvajanja zanke
5. **pri rekurziji** izrazimo zahtevnost kot rekurenčno enačbo

PRIMERI

Določi časovno zahtevnost:

```
s = 0;  
for (int i = 1; i <= n; i++)  
    for (int j = 1; j <= n; j++)  
        s = s + t[i][j];
```

n je velikost problema



$O(n^2)$



PRIMERI

Določi časovno zahtevnost:

```
void p(int n, int m, int k) {  
    s = 0;  
    for (int i = 1; i <= n; i++)  
        for (int j = 1; j <= m; j += k)  
            s = s + t[i][j];  
}
```



n , m in k določajo velikost problema

$$O\left(n \cdot \frac{m}{k}\right)$$



PRIMERI

Določi časovno zahtevnost:

```
int i = n;
int r = 0;
while (i > 1) {
    r = r + 1;
    i = i / 2;
}
```

$O(\log n)$





PRIMERI

Določi časovno zahtevnost:

kot če bi bilo
vgnezdenih m zank

```
void p(int n, int m) {  
    if (m > 0)  
        for (int i = 1; i <= n; i++)  
            p(n, m-1);  
}
```

$O(n^m)$



PRIMERI

Določi časovno zahtevnost:

```
void p(int n) {  
    if (n > 0)  
        for (int i = 1; i <= n; i++)  
            p(n-1);  
    else  
        for (int i = 1; i <= 10; i++)  
            System.out.println(i);  
}
```

$O(n!)$



PRIMER: IZRAČUN FAKULTETE

Iterativno:

```
fakulteta = 1;
for(int i=1; i <= n; i++)
    fakulteta = fakulteta*i;
```

Rekurzivno:

```
private int fakulteta(int n) {
    if (n==0) return 1;
    else { return n * fakulteta(n-1); }
}
```



POTENCIRANJE ŠTEVILA

```
private int potenca(int x, int p) {  
    if (p==0)  
        return 1;  
    else {  
        return x * potenca(x, p-1);  
    }  
}
```



HANOJSKI STOLPI

```
// premik n ploščic iz palice A na palico B z uporabo  
// pomožne palice C
```

```
static public void hanoi(char A, char B, char C,int n) {  
    if (n>0) {  
        hanoi(A,C,B,n-1) ;  
        System.out.println("premik_iz_" + A + "_na_" + B);  
        hanoi(C,B,A,n-1) ;  
    } // if  
} // hanoi
```



PRIMER FIBONACCI



Rekurzivno

```
public static int fib(int n) {  
    if (n <= 2)  
        return 1;  
    else  
        return fib(n-1)+fib(n-2);  
}
```

Iterativno

```
public static int fib(int n) {  
    int n1=1, n2=1, n3;  
    for(int i=2; i < n; i++) {  
        n3 = n1 + n2;  
        n1 = n2;  
        n2 = n3;  
    }  
    return n2;  
}
```



PRIMER: PERMUTACIJE

```
static public void permutationsRec(int n0) {
    if (n0==0)
        writePermutation() ;
    else {
        for (int i=0, temp ; i < n0 ; i++) {
            temp = a[i] ; a[i] = a[n0-1] ; a[n0-1] = temp ;
            permutationsRec(n0-1) ;
            temp = a[i] ; a[i] = a[n0-1] ; a[n0-1] = temp ;
        }
    }
} // permutationsRec
```



WHO CARES TIME COMPLEXITY?

Control Panel > Windows Update

Windows Update



Downloading updates...



Downloading 188 updates (765,0 MB total, 90% complete)

Most recent check for updates: Today at 17:06

Updates were installed: Today at 15:33. [View update history](#)

You receive updates: For Windows and other products from Microsoft Update



OCENA DEJANSKEGA ČASA IZVAJANJA

- pri oceni velikostnega reda časovne zahtevnosti zanemarimo vse člene nižjega reda kot tudi vse konstante
- v realnosti ravno konstante lahko spremenijo sliko uporabnosti algoritma

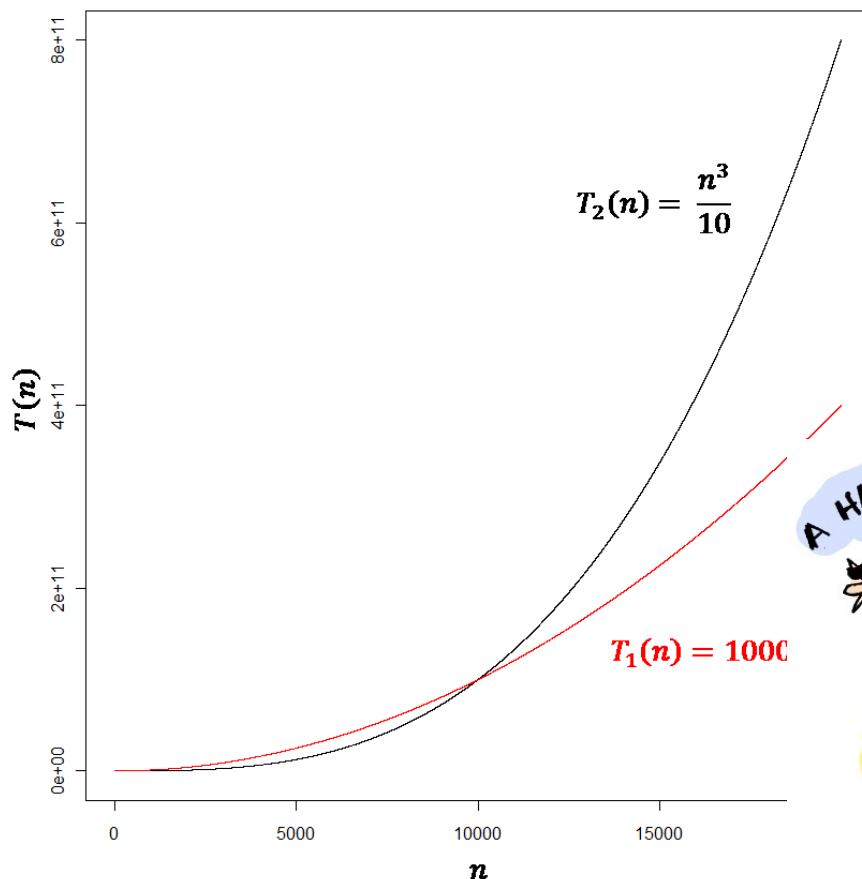


$$T_1(n) = 1000n^2 \quad T_2(n) = \frac{n^3}{10}$$

$O(n^2)$ < $O(n^3)$

ampak... za $n < 10000$ je drugi algoritem primernejši!

OCENA DEJANSKEGA ČASA IZVAJANJA

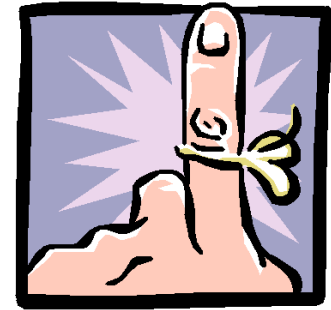


za $n < 10000$ je drugi algoritem primernejši!

OCENA DEJANSKEGA ČASA IZVAJANJA

- predpostavimo neko odvisnost med časovno zahtevnostjo in dejanskim časom izvajanja, npr:

$$T(n) = a * O(g(n)) + c$$



- oceno konstant izvajamo z meritvami
- za določitev konstant v enačbi je potrebno opraviti toliko meritev, kolikor je konstant
- meritve opravljamo pri velikih vrednostih vhodnih parametrov
- rešimo dobljeni sistem enačb
- dobljeno funkcijo lahko uporabimo za napovedovanje časa izvajanja programa



PRIMER 1

Za dani program so bili izmerjeni naslednji časi izvajanja za različne velikosti vhodnih podatkov:

velikost podatkov	5	6	7	8	9	10
čas	13.8	83.6	388.6	1796.2	8753.6	44421.4

Zanima nas, kako hitro raste funkcija:

- polinomske: $T(n) = c n^e$
- eksponentno: $T(n) = c 2^{en}$



PRIMER 1

Za dani program so bili izmerjeni naslednji časi izvajanja za različne velikosti vhodnih podatkov:

velikost podatkov	5	6	7	8	9	10
čas	13.8	83.6	388.6	1796.2	8753.6	44421.4
c		1.7 E-06	1.5E-06	8.0E-08	1.3E-09	1.7E-11
e		9.88	9.97	11.46	13.45	15.42

- polinomsko: $T(n) = c n^e$

Konstante NISO konstantne...



PRIMER 1

Za dani program so bili izmerjeni naslednji časi izvajanja za različne velikosti vhodnih podatkov:

velikost podatkov	5	6	7	8	9	10
čas	13.8	83.6	388.6	1796.2	8753.6	44421.4
c		1.7 E-03	8.3E-03	8.6E-03	5.6E-03	3.9E-03
e		2.60	2.22	2.21	2.28	2.34

- eksponentno: $T(n) = c 2^{en}$

Konstante SO vsaj do neke mere konstantne...



PRIMER 2

Za dani program so bili izmerjeni naslednji časi izvajanja za različne velikosti vhodnih podatkov:

velikost podatkov	5	10	15	30
čas	500	501	502	509

Katera funkcija najbolj ustreza časovni zahtevnosti tega programa v odvisnosti od vhodnih podatkov?

- a) $\log(n)$
- b) n
- c) $n \log(n)$
- d) n^2



PRIMER 2

$$T(n) = a * O(g(n)) + c$$

Preverimo funkcijo $\log(n)$:

$$T(n) = a * \log(n) + c$$

velikost podatkov	5	10	15	30
čas	500	501	502	509

$$a * \log(30) + c = 509$$

$$a * \log(15) + c = 502$$

$$a = 7$$

$$c = 474.7$$

Preverimo ustreznost rešitve:

$$7 * \log(10) + 474.7 = 497.94 \neq 501$$



PRIMER

$$T(n) = a * O(g(n)) + c$$

Preverimo funkcijo n:

$$T(n) = a * n + c$$

velikost podatkov	5	10	15	30
čas	500	501	502	509

$$a * 30 + c = 509$$

$$a * 15 + c = 502$$

$$a = 0.47$$

$$c = 494.9$$

Preverimo ustreznost rešitve:

$$0.47 * 10 + 494.9 = 499.6 \neq 501$$



PRIMER 2

$$T(n) = a * O(g(n)) + c$$

Preverimo funkcijo n^2 :

$$T(n) = a * n^2 + c$$

velikost podatkov	5	10	15	30
čas	500	501	502	509

$$a * 30^2 + c = 509$$

$$a * 15^2 + c = 502$$

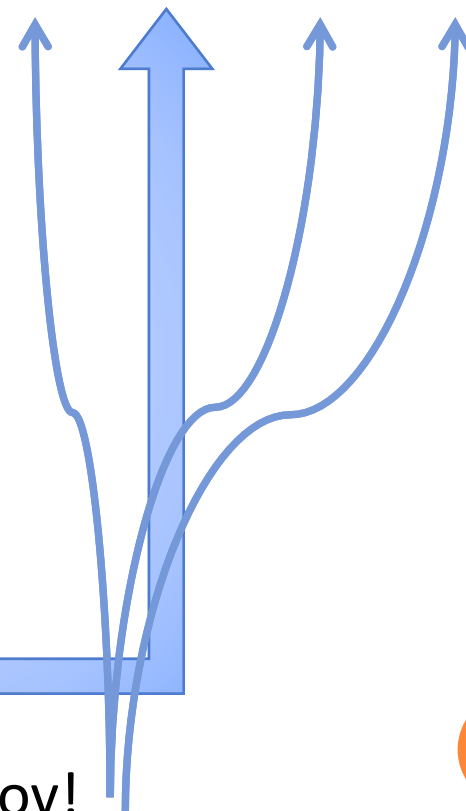
$$a = 0.01$$

$$c = 500$$

Preverimo ustreznost rešitve:

$$0.01 * 10^2 + 500 = 501$$

preverimo še za vse ostale velikosti podatkov!



OCENA DEJANSKEGA ČASA IZVAJANJA

Če ne najdemo funkcije, ki natančno aproksimira model:

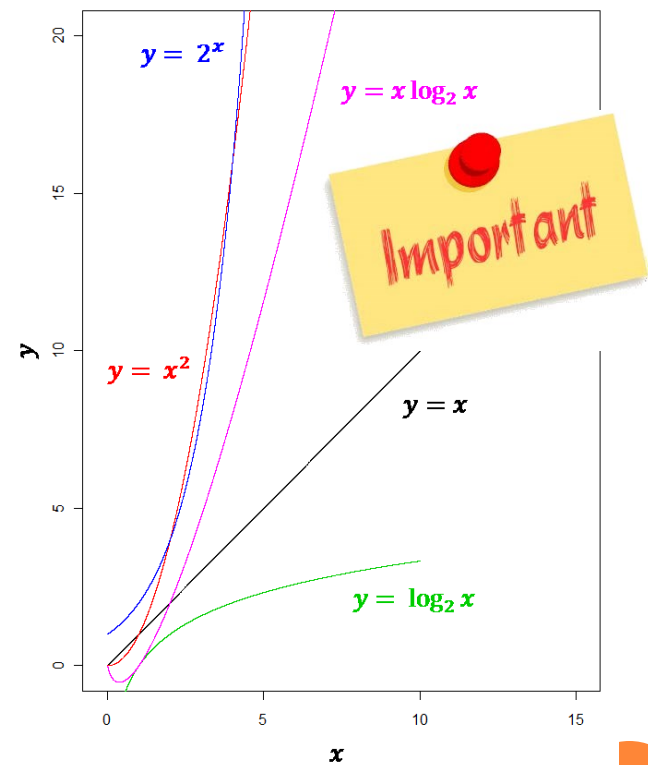


- izračunamo koeficiente za vse smiselne modele
- izberemo model, ki daje najbližje rezultate



POMEMBNE NEENAKOSTI

- $a > b > 0, c > 0 \rightarrow n^a > cn^b$
- $a > 0, b > 1, c > 0 \rightarrow n^a > c \log_b n$
- $a > 1, b > 0, c > 0 \rightarrow a^n > cn^b$
- $a > 1, c > 0 \rightarrow n! > ca^n$
- $c > 0 \rightarrow n^n > cn!$



RAST FUNKCIJ ZAHTEVNOSTI

$f(n)$	$f(n+1) - f(n) = O(g(n))$
$\log n$	$\log(n+1) - \log n = O\left(\frac{1}{n}\right)$
n	$1 = O(1)$
$n \log n$	$(n+1)\log(n+1) - n \log n = O(\log n)$
n^2	$2n+1 = O(n)$
n^3	$3n^2 + 3n + 1 = O(n^2)$
n^4	$4n^3 + 6n^2 + 4n + 1 = O(n^3)$
2^n	$2^n = O(2^n)$
$n!$	$n \times n! = O((n+1)!)$



RAST FUNKCIJ ZAHTEVNOSTI

$f(n)$	$f(n+1) - f(n) = O(g(n))$	povečanje velikosti rešljivega problema v danem času z $10 \times$ hitrejšim rač.
$\log n$	$\log(n+1) - \log n = O\left(\frac{1}{n}\right)$	n^{10}
n	$1 = O(1)$	$10n$
$n \log n$	$(n+1)\log(n+1) - n \log n = O(\log n)$	$< 10n$
n^2	$2n+1 = O(n)$	$3.16n$
n^3	$3n^2 + 3n + 1 = O(n^2)$	$2.15n$
n^4	$4n^3 + 6n^2 + 4n + 1 = O(n^3)$	$1.78n$
2^n	$2^n = O(2^n)$	$\leq n + 4$
$n!$	$n \times n! = O((n+1)!)$	$\leq n + 1$