Strukture podataka

Složenost algoritama

Parametri programa

- vreme izvršenja (kao funkcija ulaznih parametara)
- maksimalni memorijski prostor za smeštanje podataka
- ukupna veličina programskog koda
- korektno izračunavanje željenih rezultata
- kompleksnost (koliko lako se čita, razume, modifikuje)
- robusnost (koliko lako "izlazi na kraj" sa neočekivanim ili pogrešnim ulaznim parametrima)

Modeli računara

- **Detaljni**
  - Zahteva poznavanje velikog broja parametara (τ_{fetch}, τ_{store}, τ_{+}, τ_{-}, $\tau_{/}$, τ_{*}, $\tau_{<}$, τ_{call}, τ_{return}, $\tau_{[\bullet]}$)
  - Dobro predviđanje performansi
  - Veoma komplikovana analiza
- **Pojednostavljeni**
  - Olakšava analizu, ali
  - Nepreciznije predviđa performanse

Aksiomi

- Vremena za preuzimanje operanda iz memorije (τ_{fetch}) i smeštanje operanda (τ_{store}) u memoriju su konstante.
  - vreme izvršenja Y = X ; je $\boldsymbol{\tau_{fetch} + \tau_{store}}$
  - $\tau_{fetch\ (X)} + \tau_{store(Y)}$
- Vremena izvršavanja elementarnih aritmetičkih i logičkih operacija su konstante (τ_{+} , τ_{-} , $\tau_{/}$, τ_{*} , $\tau_{<}$)
  - vreme izvršenja Y = Y + 1 ; je $\boldsymbol{2\tau_{fetch} + \tau_{+} + \tau_{store}}$
  - $\tau_{fetch(Y)} + \tau_{fetch(1)} + \tau_{+} + \tau_{store(Y)}$
- Vremena potrebna za poziv metoda (τ_{call}) i vreme potrebno za povratak iz metoda (τ_{return}) su konstante

Aksiomi

- Vreme potrebno da se prosledi parametar metodu jednako je vremenu smeštanja u memoriju (τ_{store})
  - vreme izvršenja Y = f(x); je $\tau_{\mathbf{fetch}} + 2\tau_{\mathbf{store}} + \tau_{\mathbf{call}} + \mathbf{T}_{\mathbf{f(x)}} + \tau_{\mathbf{ret}}$

  gde je $\mathbf{T}_{\mathbf{f(x)}}$ vreme izvršenja metoda.
  - $\tau_{fetch(X)} + \tau_{call} + \tau_{store((X))} + T_{f(x)} + \tau_{ret} + \tau_{store(Y)}$
- Vremena potrebno za alokaciju memorije korišćenjem operatora *new* (τ_{new}) i dealokaciju korišćenjem operatora *delete* (τ_{del}) su konstante.
  - vreme izvršenja int* ptr = new int; je $\tau_{\mathbf{new}} + \tau_{\mathbf{store}}$
  - vreme izvršenja delete ptr; je $\tau_{\mathbf{fetch}} + \tau_{\mathbf{del}}$

Primer – *izračunavanje sume*

Izračunati vreme izvršavanja sume $\sum_{i=1}^{n} i$

```
1  unsigned int Sum (unsigned int n)
2  {
3      unsigned int result = 0;
4      for (unsigned int i = 1; i <= n; ++i)
5          result += i;
6      return result;
7  }
```

naredba	vreme	kod
3	$\tau_{\text{fetch}} + \tau_{\text{store}}$	result = 0
4a	$\tau_{\text{fetch}} + \tau_{\text{store}}$	i = 1
4b	$(2\tau_{\text{fetch}} + \tau_{<}) \times (n+1)$	i <= n
4c	$(2\tau_{\text{fetch}} + \tau_{+} + \tau_{\text{store}}) \times n$	++i
5	$(2\tau_{\text{fetch}} + \tau_{+} + \tau_{\text{store}}) \times n$	result += i
6	$\tau_{\text{fetch}} + \tau_{\text{return}}$	return result
ukupno	$(6\tau_{\text{fetch}} + 2\tau_{\text{store}} + \tau_{<} + 2\tau_{+}) \times n$ $+ (5\tau_{\text{fetch}} + 2\tau_{\text{store}} + \tau_{<} + \tau_{\text{return}})$	

Indeksiranje polja

- Vreme potrebno za izračunavanje adrese operanda u polju na osnovu indeksa je konstanta ($\tau_{[\bullet]}$). To vreme ne uključuje izračunavanje indeksnog izraza niti vreme potrebno za pristup elementu.
  - vreme izvršenja y = a [i] ; je $\mathbf{3\tau_{fetch} + \tau_{[\bullet]} + \tau_{store}}$
  - $\tau_{fetch(a)} + \tau_{fetch(i)} + \tau_{[\bullet]} + \tau_{fetch(a[i])} + \tau_{store(y)}$

Primer – *Hornerova šema*

Izračunati vreme izvršavanja sume $\sum_{i=0}^{n} a_i x^i$ primenom Hornerove šeme.

```
1    int Horner (int a [], unsigned int n, int x)
2    {
3        int result = a [n];
4        for (int i = n - 1; i >= 0; --i)
5                result = result * x + a [i];
6        return result;
7    }
```

naredba	vreme
3	$3\tau_{\text{fetch}} + \tau_{[\cdot]} + \tau_{\text{store}}$
4a	$2\tau_{\text{fetch}} + \tau_{-} + \tau_{\text{store}}$
4b	$(2\tau_{\text{fetch}} + \tau_{<}) \times (n+1)$
4c	$(2\tau_{\text{fetch}} + \tau_{-} + \tau_{\text{store}}) \times n$
5	$(5\tau_{\text{fetch}} + \tau_{[\cdot]} + \tau_{+} + \tau_{\times} + \tau_{\text{store}}) \times n$
6	$\tau_{\text{fetch}} + \tau_{\text{return}}$
TOTAL	$(9\tau_{\text{fetch}} + 2\tau_{\text{store}} + \tau_{<} + \tau_{[\cdot]} + \tau_{+} + \tau_{\times} + \tau_{-}) \times n$ $+ (8\tau_{\text{fetch}} + 2\tau_{\text{store}} + \tau_{[\cdot]} + \tau_{-} + \tau_{<} + \tau_{\text{return}})$

Primer – *rekurzija*

Primer rekurzivnog računanja faktorijela

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

```
1    unsigned int Factorial (unsigned int n)
2    {
3        if (n == 0)
4            return 1;
5        else
6            return n * Factorial (n - 1);
7    }
```

	vreme	
naredba	n=0	n>0
3	$2\tau_{fetch} + \tau_{<}$	$2\tau_{fetch} + \tau_{<}$
4	$\tau_{fetch} + \tau_{return}$	--
6	--	$3\tau_{fetch} + \tau_{-} + \tau_{store} + \tau_{\times}$
		$+ \tau_{call} + \tau_{return} + T(n-1)$

$$T(n) = \begin{cases} t_1 & n = 0 \\ T(n-1) + t_2 & n > 0 \end{cases}$$

$$t_1 = 3\tau_{fetch} + \tau_{<} + \tau_{return}$$
$$t_2 = 5\tau_{fetch} + \tau_{<} + \tau_{-} + \tau_{store} + \tau_{\times} + \tau_{call} + \tau_{return}$$

$$T_6 = \tau_{fetch(N)} + \tau_{fetch(1)} + \tau_{-} + \tau_{call} + \tau_{store(param)} + T(n\text{-}1) + \tau_{ret} + \tau_{fetch(N)} + \tau_{*}$$

Rešavanje rekurzivnih jednacina

$$\begin{aligned}
T(n) &= T(n-1) + t_2 \\
&= (T(n-2) + t_2) + t_2 \\
&= T(n-2) + 2t_2 \\
&= (T(n-3) + t_2) + 2t_2 \\
&= T(n-3) + 3t_2 \\
&\vdots
\end{aligned}$$

$$T(n) = T(n-k) + kt_2, \qquad 1 \leq k \leq n$$

$$\begin{aligned}
T(n) &= T(n-k) + kt_2 \\
&= T(0) + nt_2 \\
&= t_1 + nt_2 \qquad (2.7)
\end{aligned}$$

$$t_1 = 3\tau_{\text{fetch}} + \tau_{<} + \tau_{\text{return}}$$

$$t_2 = 5\tau_{\text{fetch}} + \tau_{<} + \tau_{-} + \tau_{\text{store}} + \tau_{\times} + \tau_{\text{call}} + \tau_{\text{return}}$$

Primer – *maksimalni element*

```
1   unsigned int FindMaximum (unsigned int a [], unsigned int n)
2   {
3       unsigned int result = a [0];
4       for (unsigned int i = 1; i < n; ++i)
5           if (a [i] > result)
6               result = a [i];
7       return result;
8   }
```

naredba	vreme
3	$3\tau_{\text{fetch}} + \tau_{[\cdot]} + \tau_{\text{store}}$
4a	$\tau_{\text{fetch}} + \tau_{\text{store}}$
4b	$(2\tau_{\text{fetch}} + \tau_{<}) \times n$
4c	$(2\tau_{\text{fetch}} + \tau_{+} + \tau_{\text{store}}) \times (n-1)$
5	$(4\tau_{\text{fetch}} + \tau_{[\cdot]} + \tau_{<}) \times (n-1)$
6	$(3\tau_{\text{fetch}} + \tau_{[\cdot]} + \tau_{\text{store}}) \times ?$
7	$\tau_{\text{fetch}} + \tau_{\text{store}}$

$$T(n, a_0, a_1, \ldots, a_{n-1}) = t_1 + t_2 n + \sum_{\substack{i=1 \\ a_i > (\max_{0 \le j < i} a_j)}}^{n-1} t_3$$

$$\begin{aligned}
t_1 &= 2\tau_{\text{store}} - \tau_{\text{fetch}} - \tau_{+} - \tau_{<} \\
t_2 &= 8\tau_{\text{fetch}} + 2\tau_{<} + \tau_{[\cdot]} + \tau_{+} + \tau_{\text{store}} \\
t_3 &= 3\tau_{\text{fetch}} + \tau_{[\cdot]} + \tau_{\text{store}}.
\end{aligned}$$

Pojednostavljeni model računara

- Detaljni model uzima u obzir različita vremena konkretne implementacije računara, i to kao celobrojni umnožak takta $\tau_{fetch} = k_{fetch}\ T$
- Pojednostavljeni model podrazumeva:
  - svi vremenski parametri zadaju se u jedinicama takta, tj. T=1,
  - konstanta proporcionalnosti za sve parametre smatra se istom i jednakom 1

Primer – *sumiranje geometrijske progresije - I način*

$$\sum_{i=0}^{n} x^i$$

```
1    double GeometricSeriesSum (double x, unsigned int n)
2    {
3        double sum = 0;
4        for (unsigned int i = 0; i <= n; ++i)
5        {
6            int prod = 1;
7            for (unsigned int j = 0; j < i; ++j)
8                prod *= x;
9            sum += prod;
10       }
11       return sum;
12   }
```

naredba	vreme
3	2
4a	2
4b	$3(n+2)$
4c	$4(n+1)$
6	$2(n+1)$
7a	$2(n+1)$
7b	$3\sum_{i=0}^{n}(i+1)$
7c	$4\sum_{i=0}^{n} i$
8	$4\sum_{i=0}^{n} i$
9	$4(n+1)$
11	2
Ukupno	$\frac{11}{2}n^2+\frac{47}{2}n+24$

$$\sum_{i=0}^{n} i = n(n+1)/2$$

Primer – *sumiranje geometrijske progresije primenom Hornerove šeme - II način*

Suma geometrijske progresije može se posmatrati i kao vrednost polinoma čiji su svi koeficijenti 1. S = (...((x + 1) * x + 1) * x + ...) * x + 1

```
1    double GeometricSeriesSum (double x, unsigned int n)
2    {
3        double sum = 0;
4        for (unsigned int i = 0; i <= n; ++i)
5            sum = sum * x + 1;
6        return sum;
7    }
```

naredba	vreme
3	2
4a	2
4b	3(*n*+2)
4c	4(*n*+1)
5	6(*n*+1)
6	2
Ukupno	13*n*+22

$$\sum_{i=0}^{n} a_i = \frac{a^{n+1} - 1}{a - 1}$$

Primer – *računanje stepena*

Odrediti složenost “standardnog” načina računanja stepena:

```
int result = 1;
for (int i = 0; i <= n; ++i)
        result *= x;
```

Razmotrimo rekurzivni način izračunavanja:

$$x^n = \begin{cases} 1 & n = 0 \\ (x^2)^{\lfloor n/2 \rfloor} & n > 0 \quad n - parno \\ x(x^2)^{\lfloor n/2 \rfloor} & n > 0 \quad n - neparno \end{cases}$$

Primer – *računanje stepena*

```
1     double Power (double x, unsigned int n)
2     {
3         if (n == 0)
4                         return 1;
5         else if (n % 2 == 0) // n je parno
6                         return Power (x * x, n / 2);
7         else // n je neparno
8                         return x * Power (x * x, n / 2);
9     }
```

$$x^n = \begin{cases} 5 & n = 0 \\ 18 + T(\lfloor n/2 \rfloor) & n > 0 \quad n - parno \\ 20 + T(\lfloor n/2 \rfloor) & n > 0 \quad n - neparno \end{cases}$$

$$T(n) = 19(\lfloor \log_2 n \rfloor + 1) + 5$$

	vreme		
naredba	$n=0$	$n>0$	$n>0$
		n is even	n is odd
3	3	3	3
4	2	--	--
5	--	5	5
6	--	$10 + T(\lfloor n/2 \rfloor)$	--
8	--	--	$12 + T(\lfloor n/2 \rfloor)$
Ukupno	5	$18 + T(\lfloor n/2 \rfloor)$	$20 + T(\lfloor n/2 \rfloor)$

Primer – *računanje stepena*

Pretpostavimo da je $\mathbf{n = 2^k}$, za $k > 0$, tada je $\lfloor n / 2 \rfloor = n / 2 = 2^{k-1}$.

$T(2^k) = 18 + T(2^{k-1}) = 18 + 18 + T(2^{k-2}) = \ldots = 18k + T(1) = 18k + 20 + T(0)$
$= 18k + 20 + 5 = 18k + 25$

Kako je $n = 2^k$, tada je $k = \log_2 n$, pa je $\mathbf{T(n) = 18 \log_2 n + 25}$

Ako je **n** neparno, tj. $\mathbf{n = 2^k - 1}$, tada je $\lfloor n / 2 \rfloor = \lfloor (2^k - 1) / 2 \rfloor = (2^k - 2) / 2$
$= 2^{k-1} - 1$.

$T(2^k - 1) = 20 + T(2^{k-1} - 1) = \ldots = 20k + T(2^0 - 1) = 20k + 5$

Kako je $n = 2^k - 1$, tada je $k = \log_2(n+1)$, pa je $\mathbf{T(n) = 20 \log_2(n+1) + 5}$

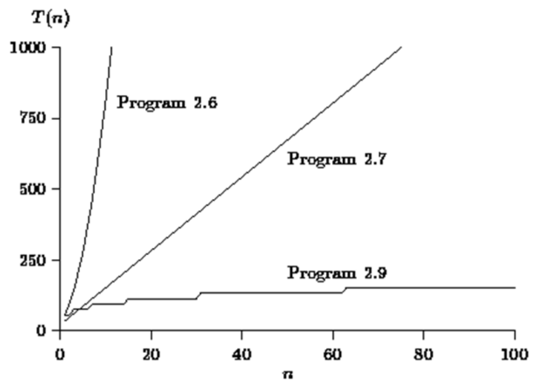
U proseku $\mathbf{T(n) = 19 (\lfloor \log_2 n \rfloor + 1) + 5}$

Primer – *sumiranje geometrijske progresije – III način*

$$\sum_{i=0}^{n} a_i = \frac{a^{n+1}-1}{a-1}$$

```
double Power (double, unsigned int);

double GeometricSeriesSum (double x, unsigned int n)
{
    return (Power (x, n + 1) - 1) / (x - 1);
}
```



program	T(n)
I način	$(\frac{11}{2}n^2 + \frac{47}{2}n + 24)$
II način	$13n+22$
III način	$19(\lfloor \log_2(n+1) \rfloor + 1) + 18$

Ilustracija složenosti

$n \setminus f(n)$	$\log n$	n	$n \log n$	n^2	2^n	$n!$
10	0.003 μs	0.01 μs	0.033 μs	0.1 μs	1 μs	3.63ms
20	0.004 μs	0.02 μs	0.086 μs	0.4 μs	1ms	77.1 god
30	0.005 μs	0.03 μs	0.147 μs	0.9 μs	1 s	8.4×10^{15} god
40	0.005 μs	0.04 μs	0.213 μs	1.6 μs	18.3 min	
50	0.006 μs	0.05 μs	0.282 μs	2.5 μs	13 dan	
100	0.007 μs	0.1 μs	0.644 μs	10 μs	4×10^{13} god	
1,000	0.010 μs	1.0 μs	9.966 μs	1ms		
10,000	0.013 μs	10 μs	130 μs	100ms		
100,000	0.017 μs	0.10 μs	1.67ms	10 s		
1,000,000	0.020 μs	1ms	19.93ms	16.7 min		
10,000,000	0.023 μs	0.01 s	0.23 s	1.16 dan		
100,000,000	0.027 μs	0.10 s	2.66 s	115.7 dan		
1,000,000,000	0.030 μs	1 s	29.9 s	31.7 god		

Neka je funkcija *f(n)* jednaka broju instrukcija koje zadati algoritam izvrši za ulaz veličine *n*. Tabela prikazuje potrebno vreme izvršavanja algoritma ako se pretpostavi da jedna instrukcija traje jednu nanosekundu (*ns*), tj. 0.001 mikrosekundu (*us*).

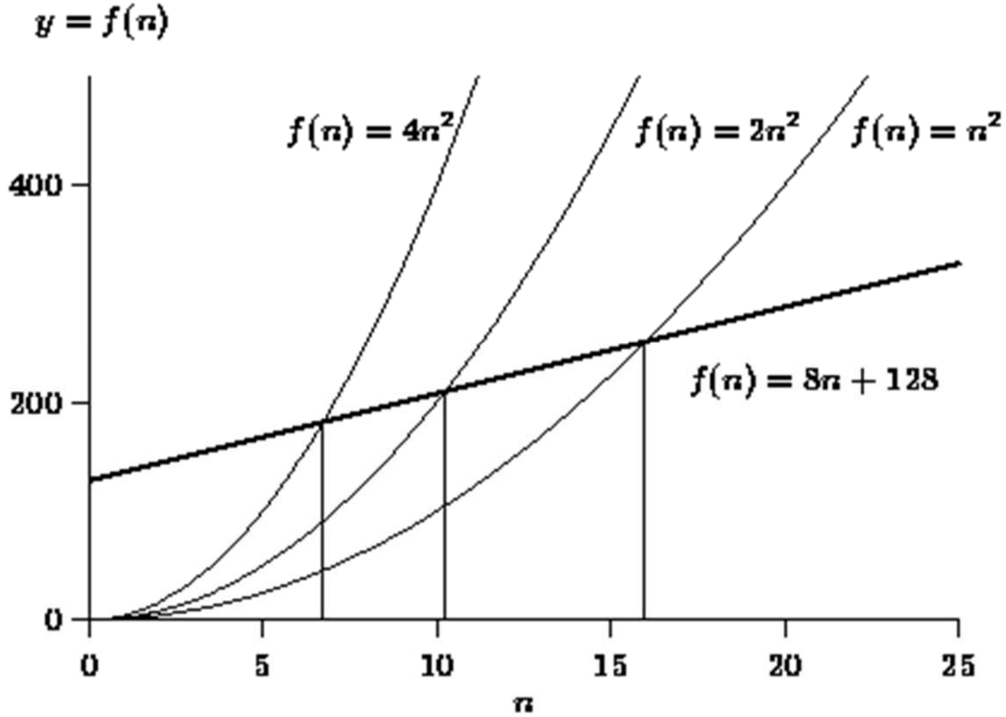
Asimptotska notacija

Obzirom da se obično ne može proceniti veličina problema, prilikom poređenja dve funkcije često se koristi asimptotsko ponašanje funkcija za slučaj vrlo velikih problema.

- Gornja granica – veliko O
- Donja granica – malo O
- Teta
- Omega

Veliko O

Za funkciju *f(n)* nenegativnu za sve cele brojeve n ≥ 0, kažemo da je "veliko O" od *g(n)*, u oznaci $f(n) = O(g(n))$, ako postoji celi broj n_0 i konstanta c > 0 tako da za sve cele brojeve važi $n \geq n_0$, $f(n) \leq cg(n)$.



$f(n) = 8n + 128 = O(n^2)$

Teoreme

T 1.1 Za polinom $f(n) = \Sigma a_i n^i = a_m n^m + a_{m-1} n^{m-1} + \ldots a_1 n + a_0$, gde je $a_m > 0$, važi da je $f(n) = O(n^m)$.

T 1.2 Za svaki ceo broj $k \geq 1$ važi $\log^k n = O(n)$.

Tačnija granica

Posmatrajmo funkciju f(n) = O(g(n)). Ako svaka funkcija h(n), za koju važi da je f(n) = O(h(n)), takođe važi da je g(n) = O (h(n)), tada kažemo da je g(n) *bliska asimptotska granica* funkcije f(n).

Primer:

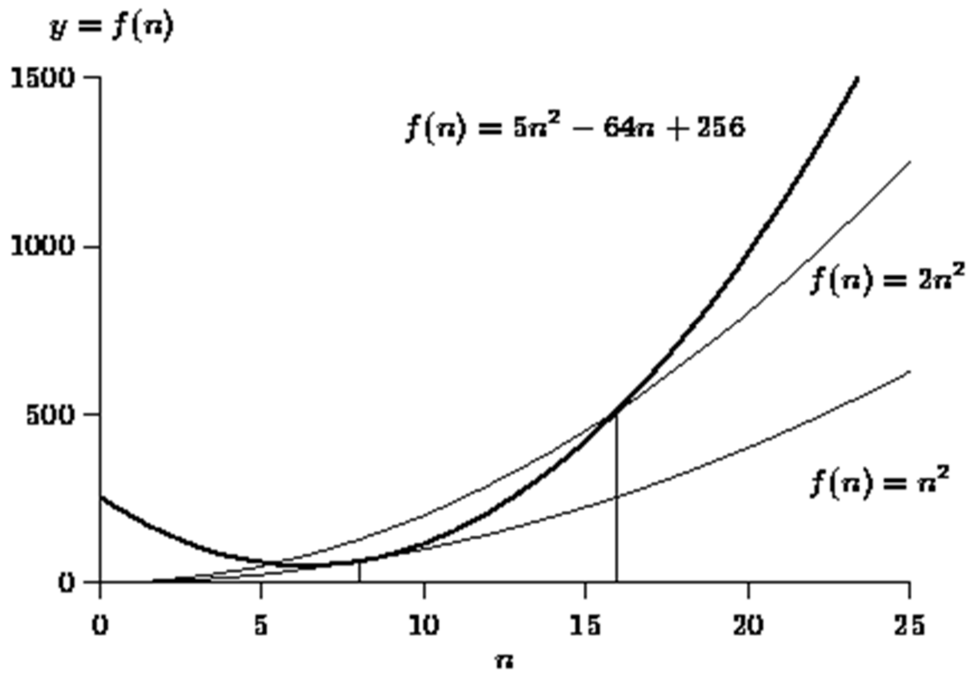
Za funkciju $f(n)=8n+128$ važi da je $f(n)=O(n^2)$, ali obzirom i da je polinom stepena 1, važi i $f(n)=O(n)$.

Očigledno je da je $O(n)$ bliža asimptotska granica!

Asimptotska donja granica - Ω

Za funkciju *f(n)* nenegativnu za sve cele brojeve $n \geq 0$, kažemo da je "omega" od *g(n)*, u oznaci $f(n) = \Omega(g(n))$, ako postoji celi broj n_0 i konstanta $c>0$ tako da za sve cele brojeve važi $n \geq n_0$, $f(n) \geq cg(n)$.

Primer: Za funkciju $f(n)=5n^2 - 64n + 256$ važi da je $f(n)= \Omega(n^2)$



Ostale notacije

- Teta: Za funkciju $f(n)$ nenegativnu za sve cele brojeve $n \geq 0$, kažemo da je "teta" od $g(n)$, u oznaci $f(n) = \Theta(g(n))$, ako i samo ako je $f(n) = O(g(n)) \wedge f(n) = \Omega(g(n))$.
  - Npr. $f(n) = a_m n^m + a_{m-1} n^{m-1} + \ldots a_1 n + a_0 = \Theta(n^m)$
- Malo o: Za funkciju $f(n)$ nenegativnu za sve cele brojeve $n \geq 0$, kažemo da je "malo o" od $g(n)$, u oznaci $f(n) = o(g(n))$, ako i samo ako je $f(n) = O(g(n)) \wedge \neg(f(n) = \Omega(g(n)))$.
  - Npr. za $f(n) = n + 1$ važi da je $f(n) = O(n^2)$, ali i da je $f(n) \neq \Theta(n^2)$, obzirom da je $cn^2 \geq n+1$, za $n>1 \Rightarrow f(n) = o(n^2)$,

Asimptotska analiza algoritama

```
1     int Horner (int a [], unsigned int n, int x)
2     {
3         int result = a [n];
4         for (int i = n - 1; i >= 0; --i)
5                     result = result * x + a [i];
6         return result;
7     }
```

naredba	detaljni model	jednostavni model	O
3	$3\tau_{\text{fetch}} + \tau_{[\cdot]} + \tau_{\text{store}}$	5	$O(1)$
4a	$2\tau_{\text{fetch}} + \tau_{-} + \tau_{\text{store}}$	4	$O(1)$
4b	$(2\tau_{\text{fetch}} + \tau_{<}) \times (n+1)$	$3n$+3	$O(n)$
4c	$(2\tau_{\text{fetch}} + \tau_{-} + \tau_{\text{store}}) \times n$	$4n$	$O(n)$
5	$(5\tau_{\text{fetch}} + \tau_{[\cdot]} + \tau_{+} + \tau_{\times} + \tau_{\text{store}}) \times n$	$9n$	$O(n)$
6	$\tau_{\text{fetch}} + \tau_{\text{return}}$	2	$O(1)$
Ukupno	$(9\tau_{\text{fetch}} + 2\tau_{\text{store}} + \tau_{<} + \tau_{[\cdot]} + \tau_{+} + \tau_{\times} + \tau_{-}) \times n + (8\tau_{\text{fetch}} + 2\tau_{\text{store}} + \tau_{[\cdot]} + \tau_{-} + \tau_{<} + \tau_{\text{return}})$	$16n$+14	$O(n)$

Pravila za “veliko O” analizu izvršenja

- *P1 (Sekvencijalno izvršenje)* Najgori slučaj izvršenja sekvenci naredbi **S_1; S_2;... S_m;** je **O(max(T_1(n), T_2(n),... T_m(n)))**, gde je vreme izvršenja i-te naredbe u sekvenci O(T_i(n)) .

- *P2 (Iteracija)* Najgori slučaj izvršavanja petlje **for (S_1; S_2; S_3) S_4;** je **O(max(T_1(n), T_2(n) × (I(n)+1), T_3(n) × I(n), T_4(n) × I(n))**, gde je vreme izvršenja i-te naredbe O(T_i(n)), a I(n) broj iteracija izvršenih u najgorem slučaju.

- *P3 (Uslov)* Najgori slučaj izvršenja if-then-else strukture **if(S_1) S_2; else S_3;** je **O(max(T_1(n), T_2(n), T_3(n)))**, gde je vreme izvršenja i-te naredbe O(T_i(n)) .

Primer – *računanje sume*

```
1    void PrefixSums (int a [], unsigned int n)
2    {
3        for (int j = n - 1; j >= 0; --j)
4        {
5          int sum = 0;
6          for (unsigned int i = 0; i <= j; ++i)
7                  sum += a[i];
8          a [j] = sum;
9        }
10   }
```

naredba	vreme
3a	$O(1)$
3b	$O(1) \times O(n)$ iterations
3c	$O(1) \times O(n)$ iterations
5	$O(1) \times O(n)$ iterations
6a	$O(1) \times O(n)$ iterations
6b	$O(1) \times O(n^2)$ iterations
6c	$O(1) \times O(n^2)$ iterations
7	$O(1) \times O(n^2)$ iterations
8	$O(1) \times O(n)$ iterations
Ukupno	$O(n^2)$

$$\sum_{j=0}^{n-1} j+1 = \sum_{j=1}^{n} j = \frac{n(n+1)}{2} = \Theta(n^2)$$

Zadatak za vežbu

- Izračunati složenost (detaljnim, pojednostavljenim i asimptotskim modelom) računanja Fibonačijevih brojeva korišćenjem sledeće funkcije:

```
unsigned int Fibonacci (unsigned int n)
{
    int previous = -1;
    int result = 1;
    for (unsigned int i = 0; i <= n; ++i)
    {
      int const sum = result + previous;
      previous = result;
      result = sum;
    }
    return result;
}
```