Analiza efikasnosti algoritma

Merenje vremena izvršavanje algoritma

- **Broj ulaznih podataka – n**
  - npr. sortiranje niza od n elemenata
- **Posmatra se vreme izvršavanja u zavisnosti od veličine ulaza**, tj. kako se vreme izvršavanja algoritma menja sa povećanjem broja ulaznih podataka
- **Manja veličina ulaza** → kraće vreme izvršavanja algoritma
- **Veća veličina ulaza** → duže vreme izvršavanja algoritma
- Kako se veličina ulaza povećava, vreme izvršavanja se takođe povećava
- Ako se ulaz u algoritam **duplira**, vreme izvršavanja može:
  - da se približno duplira (npr. algoritmi složenosti O(n))
  - da se poveća 4 puta (npr. $O(n^2)$)
  - da se poveća 100 ili više puta (npr. eksponencijalni algoritmi)

Merenje vremena izvršavanje algoritma

- **Eksperimentalna metoda**
  - Implementacija algoritma u programskom jeziku
  - Izvršavanje algoritma nad različitim ulazima
  - Beleženje vremena izvršavanja
  - Zavisna od softvera i hardvera
  - Posmatra se samo konačan broj ulaznih podataka
- **Analitička metoda**
  - Matematička analiza vremena izvršavanja u zavisnosti od veličine ulaza
  - Asimptotska analiza (O-notacija)
  - Nezavisna od softvera i hardvera
  - Razmatra sve moguće veličine ulaza

Klasa Stopwatch

- Osnovne informacije
  - Nalazi se u namespace-u System.Diagnostics
  - Kreira se konstruktorom: **Stopwatch** t = new **Stopwatch**();
- Glavne metode
  - **Start**() – pokreće merenje vremenskog intervala
  - **Stop**() – zaustavlja merenje
  - **Reset**() – resetuje tajmer
- Svojstva
  - **Elapsed** – vraća TimeSpan sa ukupno proteklim vremenom
  - **ElapsedMilliseconds** – vraća broj milisekundi(nije preporučljivo za veoma kratka vremena – koristiti **ElapsedTicks**)

Klasa Stopwatch

```
Stopwatch t = new Stopwatch();
t.Start();
TestMetoda(1000);
t.Stop();
Console.WriteLine(t.ElapsedMilliseconds);
```

Primer merenja vremena izvršavanja algoritma

```
static void UgnjezdenePetlje(int[,] x, int n)
{
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            if (i > j)
            {
                x[i, j] = 1;
            }
        }
    }
}
```

Primer merenja vremena izvršavanja algoritma

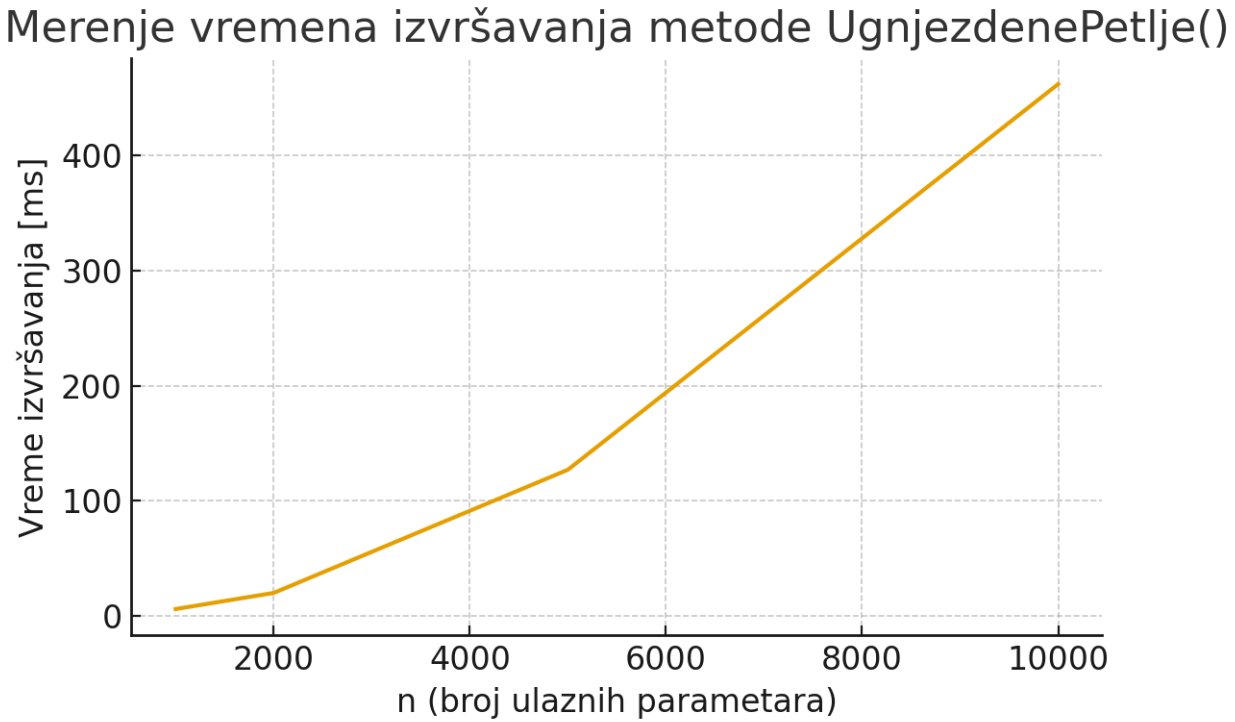
```
static void Main(string[] args)
{
    int n = 1000;                       // menjaš 2000,5000,10000
    int[,] x = new int[n, n];           // matrica n×n

    Stopwatch t = new Stopwatch();

    t.Start();
    UgnjezdenePetlje(x, n);
    t.Stop();

    Console.WriteLine($"Vreme izvršavanja za n = {n}: {t.ElapsedMilliseconds} ms");
}
```

Rezultati merenja vremena izvršavanja metode UgnjezdenePetlje ()

Broj ulaznih parametara n	1000	2000	5000	10000
Vreme izvršavanja [ms]	6	20	127	462

Napomena: Pri testiranju ne pokretati aplikaciju u debug modu nego sa CTRL + F5

Grafički prikaz vremena izvršavanja algoritma



Analitičko određivanje vremena izvršavanja algoritma

- Algoritam se analizira korak po korak i određuje se vreme njegovog izvršavanja
- Ne dobija se apsolutno precizno vreme izvršavanja
- Ovakva analiza daje dovoljno dobru procenu za razumevanje efikasnosti algoritma
- Vreme izvršavanja algoritma posmatra se kao funkcija veličine ulaznih podataka
- Za ulaz od n elemenata definiše se funkcija T(n) koja pokazuje koliko vremenskih jedinica traje izvršavanje algoritma

Jedinična instrukcija

- Jedinična instrukcija algoritma predstavlja osnovnu računsku operaciju čije je vreme izvršavanja konstantno
- Pretpostavlja se da se sve osnovne instrukcije algoritma izvršavaju tokom jedne vremenske jedinice
- Apsolutna vrednost vremenske jedinice nije bitna (ms, µs, ...) važna je samo konstantnost
- Tipične jedinične instrukcije algoritama su:
  - dodela vrednosti promenljivoj
  - poređenje vrednosti dve promenljive
  - aritmetičke operacije
  - logičke operacije
  - ulazno/izlazne operacije

Vreme izvršavanja ugnježdenih petlji

```
for (int i = 0; i < n; i++)          // izvršava se n puta
{
    for (int j = 0; j < n; j++)      // izvršava se n puta
    {
        x[i, j] = 1;                 // jedinična instrukcija
    }
}
```

$$T(n)=n \cdot n=n^2$$

Primer – indeks najmanjeg elementa niza

```
static int MinimalniClan(int[] x)
{
    int jmin = 0;          // 1 instrukcija
    int xmin = x[0];       // 1 instrukcija
    int i = 1;             // 1 instrukcija

    while (i < x.Length) // izvršava se n puta
    {
        if (x[i] < xmin) // izvršava se n-1 puta
        {
            xmin = x[i]; // u najgorem slučaju n-1 puta
            jmin = i;    // u najgorem slučaju n-1 puta
        }
        i++;             // n-1 puta
    }

    return jmin;         // 1 instrukcija
}
```

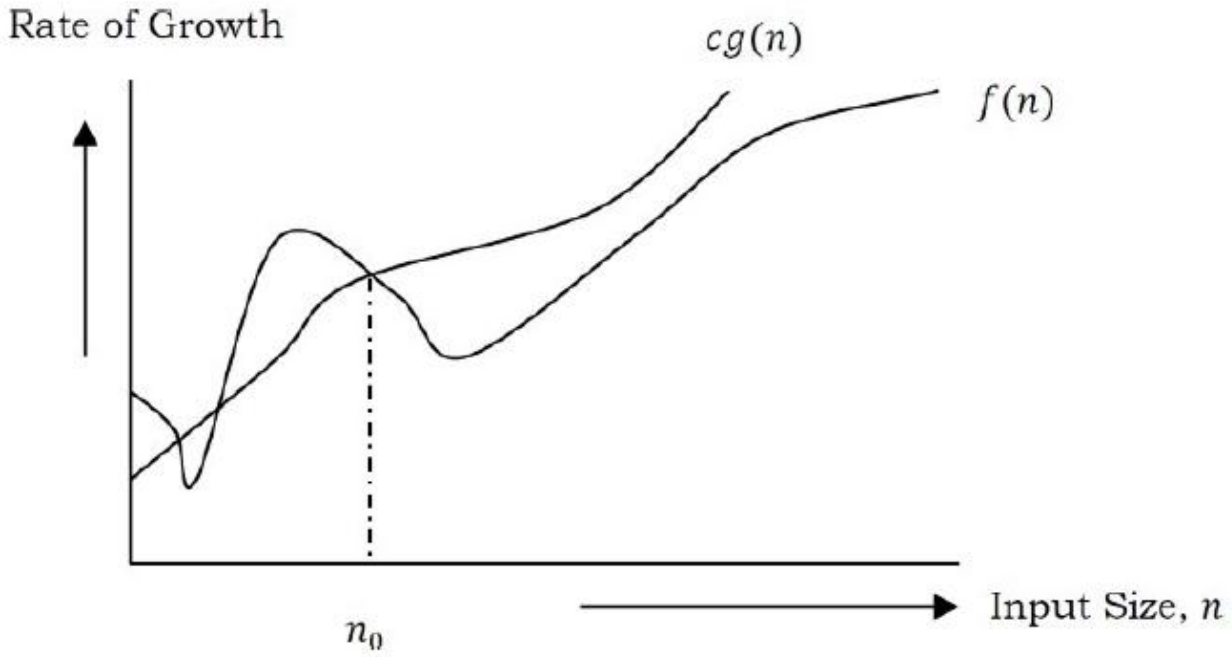
T(n)=3(inicijalizacije)
$+n$ *(provera while uslova)*
$+(n-1)$ *(if uslov)*
$+2(n-1)$ *(dve naredbe u if–u)*
$+(n-1)$ *(i++)*
$+1$ *(return)*

T(n)=3+n+(n−1)+2(n−1)+(n−1)+1
$T(n) = 5n + 1$

O-zapis

- O-zapis se koristi za precizno definisanje pojma da je neka funkcija manja od druge
- Za dve nenegativne funkcije f, g: $N \rightarrow R^{+}$ kažemo da je $f(n) = O(g(n))$ ako postoje pozitivne konstane c i n_0 tako da je $f(n) \leq c * g(n)$ za svako $n > n_0$
- Odnosno kažemo da funkcija $g(n)$ predstavlja asimptotsku gornju granicu za funkciju $f(n)$

O-zapis



$f(n) \leq c * g(n)$ za svako $n > n_0$

f(n) predstavlja vreme izvršavanja algoritma u zavisnosti od veličine ulaza n
O-zapis daje asimptotsku gornju granicu rasta funkcije f(n)
Ako je f(n) = O(g(n)), onda funkcija g(n) ograničava rast funkcije f(n) (od n_o nadalje)

f(n) je veliko O od g(n)

Tipične funkcije složenosti poređane po rastućim brzinama rada

Funkcija	Neformalno ime
1	konstantna funkcija
log n	logaritamska funkcija
n	linearna funkcija
n log n	linearno-logaritamska funkcija
n^2	kvadratna funkcija
n^3	kubna funkcija
2^n	eksponencijalna funkcija

Pronalaženje velikog O

- Ako je $f(n) = c$,

  onda je $f(n) = O(1)$

- Ako je $f(n) = a_k n^k + a_{k-1} n^{k-1} + \ldots + a_o$,

  onda je $f(n) = O(n^k)$
  - zadržava se najbrže rastući član polinoma

- Koeficijenti se ignorišu: ako je $f(n) = k \cdot g(n)$,

  onda je $f(n) = O(g(n))$

- Baza logaritma je nebitna: ako je $f(n) = 8 \cdot \log_2(n)$,

  onda je $f(n) = O(\log n)$

- Ako je $f(n) = O(2^n)$, algoritam je eksponencijalne složenosti:
  - nijedan polinom ne može ograničiti rast 2^n
  - takvi algoritmi su često nepraktični za realne sisteme

Primeri određivanja velikog O

- $f(n) = 45$ → $O(1)$
- $f(n) = 6n^3 + 27\log_2 n + 2n$ → $O(n^3)$
- $f(n) = 8\log_2 n + 7n + 6$ → $O(n)$
- $f(n) = n \cdot \log_{10} n + 5n + 81n^2$ → $O(n^2)$
- $f(n) = \log_2 n + n \cdot \log_{10} n$ → $O(n \log n)$
- $f(n) = 3n + 5n^2 + 7n^3 + 2^n$ → $O(2^n)$

Asimptotsko vreme izvršavanja algoritama

```
for (int i = 0; i < n; i++)
{
    for (int j = 0; j < n; j++)
    {
        a[i, j] = 0;
    }
}

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

$$T(n) = n^2 + n$$

za veliko n dominira kvadratni član

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

Asimptotsko vreme izvršavanja algoritama

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

$$T(n) = n^2$$

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

Pitanje 1

Eksperimentalna metoda vremena izvršavanja algoritma:

a. Ne zavisi od hardvera i softvera na kome se radi testiranje
b. Zavisi od hardvera i softvera na kome se radi testiranje
c. Zavisi isključivo od broja ulaznih parametara

Odgovor: b

Pitanje 2

Jedinična instrukcija algoritma ima:

a. Promenljivo vreme izvršavanja
b. Konstantno vreme izvršavanja
c. Vreme izvršavanja koje zavisi od tipa algoritma

Odgovor: b

Pitanje 3

Ko asimptotske analize algoritma izračunava se tačno vreme izvršavanja algoritma:

a. Da
b. Ne

Odgovor: b

Pitanje 4

Funkcija $f(n) = 2n + 3n \cdot \log_2(n)$ je veliko O za funkciju

a. $2n$
b. $n \cdot \log(n)$
c. $3n \cdot \log_2(n)$

Odgovor: b

Pitanje 5

Funkcija $f(n) = 3n^2 + 5n + 18$ je veliko O za funkciju:

a. 2^n
b. n
c. n^2

Odgovor: c

Pitanje 6

Vremenska složenost algoritma koji se izvršava za isto vreme bez obzira na broj ulaznih parametara je:

a. O(1)
b. O(n)
c. O(logn)

Odgovor: a

Pitanje 7

Koja je vremenska složenost sledećeg koda:

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

a. O(n2)
b. O(n)
c. O(1)

Odgovor: b