Programiranje 2

Složenost algoritma

Milena Vujošević Janičić
Jelena Graovac

www.matf.bg.ac.rs/~milena
www.matf.bg.ac.rs/~jgraovac

Programiranje 2
Beograd, 20. februar, 2020.

Pregled

Pregled

Složenost algoritama

- Pored svojstva ispravnosti programa, veoma je važno i pitanje koliko program zahteva vremena (ili izvršenih instrukcija) i memorije za svoje izvršavanje.
- Mnogi problemi mogu se rešavati na puno načina i nije svako rešenje jednako dobro.
- Najbolja rešenja podrazumevaju minimalnu potrošnju resursa kao što su memorija i vreme. Zato se razmatraju:
  - vremenska složenost algoritma;
  - prostorna (memorijska) složenost algoritma.

Složenost algoritama

- Problemi koje rešavamo imaju prirodnu veličinu (dimenziju) (npr količina podataka koje treba obraditi)
- Složenost algoritma je obično neka funkcija koja povezuje veličinu problema sa brojem koraka izvršavanja algoritma (vremenska složenost) ili brojem potrebnih memorijskih lokacija (prostorna složenost).

Složenost algoritama

- Vremenska i prostorna složenosti mogu se razmatrati
  - u terminima konkretnog vremena/prostora utrošenog za neku konkretnu ulaznu veličinu;
  - u terminima asimptotskog ponašanja vremena/prostora kada ulazna veličina raste.

Složenost algoritama

- Vremenska/prostorna složenost algoritma određuje i njegovu praktičnu upotrebljivost tj. najveću dimenziju ulaza za koju je moguće da će se algoritam izvršiti u nekom razumnom vremenu ili koristeći postojeće resurse.
- Najčešće se govori o vremenskoj složenosti algoritama, ali u većini situacija potpuno analogna razmatranja mogu se primeniti na prostornu složenost.

Pregled

Merenje utrošenog vremena

- U standardnoj biblioteci postoje razne funkcije za merenje vremena, npr *time*, *difftime*, *clock*.
- Navedene funkcije mogu se koristiti za merenje vremena koje troši neka funkcija ili operacija.
- Najbolje je izvršiti veći broj merenja, kako bi se dobili precizniji rezultati.

Procena vremena

- Vreme izvršavanja pojedinih delova koda (na nekom konkretnom računaru) može da se proceni ukoliko je raspoloživa procena vremena izvršavanja pojedinih operacija.
- Na primer, na računaru sa procesorom Intel Core i7-2670QM 2.2GHz koji radi pod operativnim sistemom Linux, operacija sabiranja dve vrednosti tipa `int` troši oko trećinu nanosekunde, a operacija množenja oko jednu nanosekundu.
- Ova procenu treba uzimati sa velikom rezervom jer ne uzima razne važne aspekte izvršavanja programa (koji se još procesi trenutno izvršavaju, koji je operativni sistem, kako je napravljena izvršna verzija programa i slično).

Profajleri

- Procena vremena izvršavanja pojedinih operacija i kontrolnih struktura na konkretnom računaru može se napraviti merenjem.
- Postoje mnogi alati za analizu i unapređenje performansi programa i najčešće se zovu *profajleri* (engl. profiler).
- Njihova osnovna uloga je da pruže podatke o tome koliko puta je koja funkcija pozvana tokom (nekog konkretnog) izvršavanja, koliko je utrošila vremena i slično.
- Ukoliko se razvijeni program ne izvršava željeno brzo, prvi kandidati za izmenu su delovi koji troše najviše vremena.
- Primer profajlera: alat `callgrind`, sastavni deo sistema `valgrind` koji objedinjuje mnoštvo alata za dinamičku analizu rada programa.

Pregled

Analiza najgoreg slučaja

- Umesto konkretnih merenja, vreme izvršavanja programa može biti opisano opštije, u vidu funkcije koja zavisi od ulaznih argumenata.
- Često se algoritmi ne izvršavaju isto za sve ulaze istih veličina.
  - *Analiza najgoreg slučaja* zasniva procenu složenosti algoritma na najgorem slučaju (ova procena može da bude varljiva, ali predstavlja dobar opšti način za poređenje efikasnosti algoritama).
  - U nekim situacijama moguće je izračunati prosečno vreme izvršavanja algoritma.
  - Analiziranje najboljeg slučaja, naravno, nema smisla.
- Najčešće se koristi analiza najgoreg slučaja.

Red složenosti algoritma

- Neka je funkcija $f(n)$ jednaka broju instrukcija koje zadati algoritam izvrši za ulaz veličine n.
- Naredna tabela prikazuje potrebno vreme izvršavanja algoritma ako se pretpostavi da jedna instrukcija traje jednu nanosekundu.

Tabela vremena izvršavanja: 1 instrukcija — 10^{-9} sekunde

$n \setminus f(n)$	$\log n$	n	$n \log n$	n^2	2^n	$n!$
10	0.003 μs	0.01 μs	0.033 μs	0.1 μs	1 μs	3.63 *ms*
20	0.004 μs	0.02 μs	0.086 μs	0.4 μs	1 *ms*	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	1 *ms*		
10,000	0.013 μs	10 μs	130 μs	100 *ms*		
100,000	0.017 μs	0.10 μs	1.67 *ms*	10 *s*		
1,000,000	0.020 μs	1 *ms*	19.93 *ms*	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*		

Tabela: Ilustracija vremena izvršavanja

Primeri

- Vodeći član u funkciji $f(n)$ određuje potrebno vreme izvršavanja.
- Na primer, ako je broj instrukcija $n^2 + 2n$, onda za ulaz dimenzije 1000000, član n^2 odnosi 16.7 minuta dok član $2n$ odnosi samo dodatne dve mikrosekunde.
- Vremenska (a i prostorna) složenost je, dakle, skoro potpuno određena „vodećim“ (ili „dominantnim“) članom u izrazu koji određuje broj potrebnih instrukcija.
- Na upotrebljivost algoritma ne utiču mnogo ni multiplikativni i aditivni konstantni faktori u broju potrebnih instrukcija, koliko asimptotsko ponašanje broja instrukcija u zavisnosti od veličine ulaza.

O notacija

O notacija

Ako postoje pozitivna konstanta c i prirodan broj N takvi da za funkcije f i g nad prirodnim brojevima važi

$$f(n) \leq c \cdot g(n) \text{ za sve vrednosti } n \text{ veće od } N$$

onda pišemo

$$f = O(g)$$

i čitamo „f je veliko ‚o‘ od g“.

Naglasimo da O ne označava neku konkretnu funkciju, već klasu funkcija i uobičajeni zapis $f = O(g)$ zapravo znači $f \in O(g)$.

Aditivne i multiplikativne konstante

- U izrazu $5n^2 + 1$, za konstantu 1 kažemo da je aditivna, a za konstantu 5 da je multiplikativna
- Aditivne i multiplikativne konstante ne utiču na klasu kojoj funkcija pripada.
- Zato se kaže da je neki algoritam složenosti $O(n^2)$, a ne npr. klasi $O(5n^2 + 1)$, jer aditivna konstanta 1 i multiplikativna 5 nisu od suštinske važnosti.
- Ako jedan algoritam zahteva $5n^2 + 1$ instrukcija, a drugi n^2, i ako se prvi algoritam izvršava na računaru koji je šest puta brži od drugog, on će biti brže izvršen za svaku veličinu ulaza.
- Ali, ako jedan algoritam zahteva n^2 instrukcija, a drugi n, ne postoji računar na kojem će prvi algoritam da se izvršava brže od drugog za svaku veličinu ulaza!

Primeri

- $n^2 = O(n^2)$
- $n^2 + 10 = O(n^2)$
- $10 \cdot n^2 + 10 = O(n^2)$
- $10 \cdot n^2 + 8n + 10 = O(n^2)$
- $n^2 = O(n^3)$
- $2^n = O(2^n)$
- $2^n + 10 = O(2^n)$
- $10 \cdot 2^n + 10 = O(2^n)$
- $2^n + n^2 = O(2^n)$
- $3^n + 2^n = O(3^n)$
- $2^n + 2^n n = O(2^n n)$

Neka svojstva

Važe sledeća svojstva:

- ako važi $f_1 = O(g_1)$ i $f_2 = O(g_2)$, onda važi i

$$f_1 \cdot f_2 = O(g_1 \cdot g_2)$$

$$f_1 + f_2 = O(g_1 + g_2)$$

- ako važi $f = O(g)$ i a i b su pozitivne konstante, onda važi i

$$af + b = O(g)$$

Θ notacija

Θ notacija

Ako postoje pozitivne konstante c_1 i c_2 i prirodan broj N takvi da za funkcije f i g nad prirodnim brojevima važi

$$c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n) \text{ za sve vrednosti } n \text{ veće od } N$$

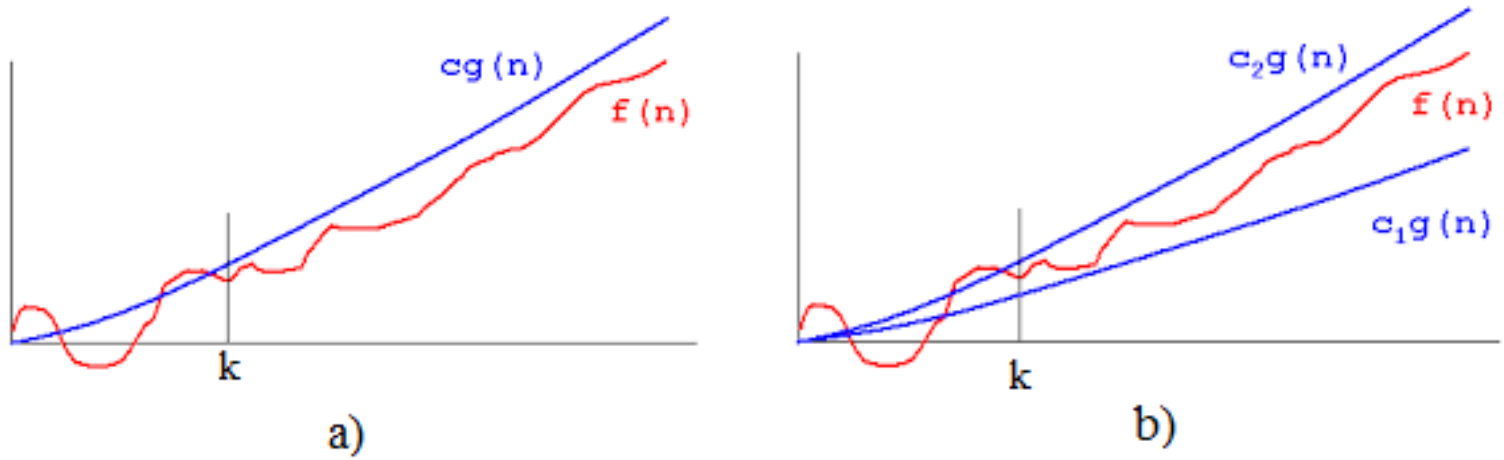
onda pišemo

$$f = \Theta(g)$$

i čitamo „f je veliko 'teta' od g“.

Ako važi $f = \Theta(g)$, onda važi i $f = O(g)$ i $g = O(f)$.

O i Θ notacija



Slika 1: O i Θ notacija: a) $f(n) = O(g(n))$, b) $f(n) = \Theta(g(n))$

Primeri

- $n^2 = \Theta(n^2)$
- $n^2 + 10 = \Theta(n^2)$
- $10 \cdot n^2 + 10 = \Theta(n^2)$
- $10 \cdot n^2 + 8n + 10 = \Theta(n^2)$
- $n^2 \neq \Theta(n^3)$
- $2^n = \Theta(2^n)$
- $2^n + 10 = \Theta(2^n)$
- $10 \cdot 2^n + 10 = \Theta(2^n)$
- $2^n + n^2 = \Theta(2^n)$
- $3^n + 2^n = \Theta(3^n)$
- $2^n + 2^n n = \Theta(2^n n)$

Klasa složenosti

Klasa složenosti O

Ako je $T(n)$ vreme izvršavanja algoritma A (čiji ulaz karakteriše prirodan broj n) i ako važi $T = O(g)$, onda kažemo da je algoritam A *složenosti* ili *reda* $O(g)$ ili da algoritam A *pripada klasi* $O(g)$.

Analogno se definiše klasa složenosti za Θ.

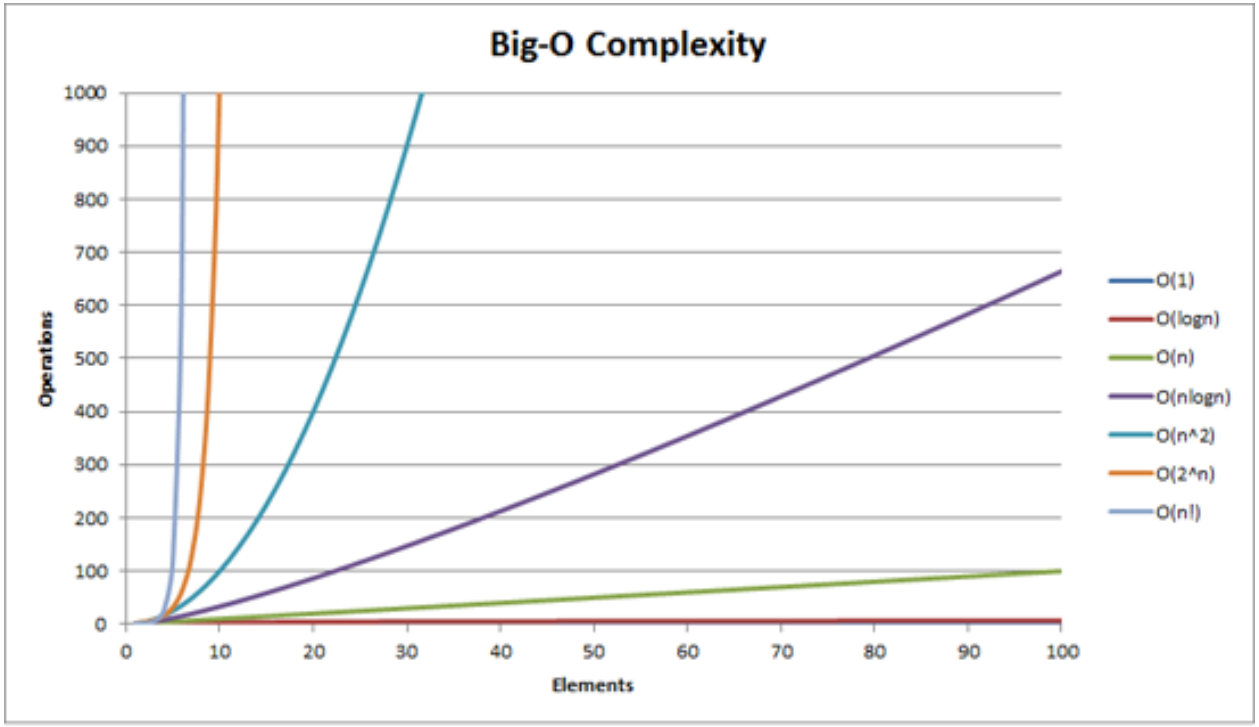
Odnosi između notacija

- Informacija o složenosti algoritma u terminima Θ (koja daje i gornju i donju granicu) je preciznija nego informacija u terminima O (koja daje samo gornju granicu).
- Na primer, algoritam koji zahteva $5n$ koraka jeste reda $\Theta(n)$, ali ne i reda $\Theta(n^2)$, s druge strane, za taj algoritam može da se smatra da je reda $O(n)$ ali i reda $O(n^2)$ (mada ne i reda, npr, $O(\log n)$).
- Često je jednostavnije složenost algoritma iskazati u terminima O nego u terminima Θ i zato se češće koristi O notacija.
- Kada se kaže da algoritam pripada klasi $O(g)$ obično se podrazumeva da je g najmanja takva klasa (ili makar — najmanja za koju se to može dokazati).

Nazivi klasa složenosti

- $O(1)$ — konstantna složenost
- $O(\log n)$ — logaritamska složenost
- $O(n)$ — linarna složenost
- $O(n^2)$ — kvadratna složenost
- $O(n^3)$ — kubna složenost
- $O(n^k)$ za neko k — polinomijalna složenost
- $O(2^n)$ — eksponencijalna složenost

Primeri nekih klasa složenosti



Primeri nekih klasa složenosti.

Prostorna složenost algoritma

- Zahtevani DODATNI memorijski prostor (pored prostora za smeštanje ulaznih podataka)
- Iste asimptotske notacije kao za vremensku složenost ($O(n)$, $\Theta(n)$)
- Iste klase algoritama kao za vremesnku složenost
- Na primer:
  - $O(1)$ — algoritam sortiranja izborom
  - $O(logn)$ — algoritam brzog sortiranja (lokacije za obradu rekurzije)
  - $O(n)$ — algoritam sortiranja spajanjem (lokacije za premeštanje elemenata)

Konstantna složenost

- U zavisnosti od prirode algoritma, složenost algoritma zavisi od vrednosti argumenata ili od broja argumenata ili od kombinacije raznih parametara.
- Ako se svi iskazi programa izvršavaju samo po jednom ili najviše po nekoliko puta (nezavisno od veličine i broja ulaznih podataka), kaže se da je vreme izvršavanja tog programa konstantno. To je očigledno najefikasnija vrsta algoritama.

Primer

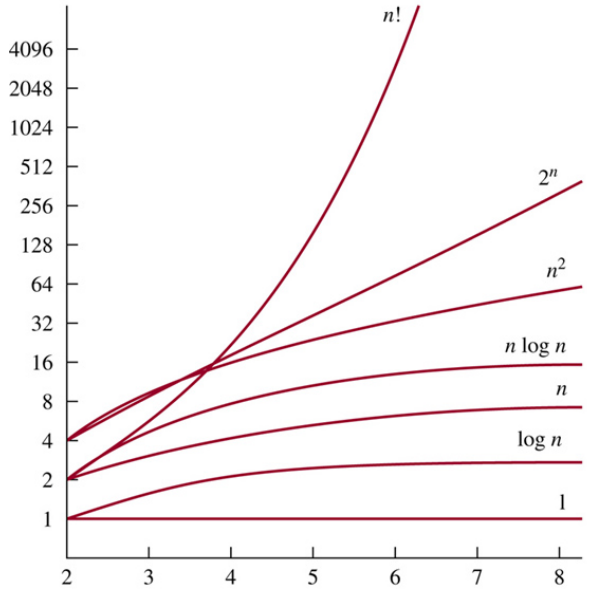
Oceniti vremensku složenost sledećeg koda. Odgovor obrazložiti.

```
int zbir(int x, int y) {
  return x+y;
}
```

Logaritamska složenost

- Logaritamska složenost — vreme izvršavanja programa lagano raste
- Ovakvu vremensku složenost ima, na primer, algoritam binarnog pretraživanja.

Logaritamska složenost



- Ako je osnova logaritma 10, za $n = 1000$, $log_{10} n$ je 3, za N = 1.000.000, $logn$ je samo dva puta veće, tj. 6.
- Ako je osnova logaritma 2, za $n = 1000$, $log_2 n$ je približno 10, što je veća konstanta od 3, ali nebitno veća.

Logaritamska složenost

- Osnova logaritma je kod ove klase algoritama zapravo nebitna jer se svi oni ponašaju isto do na konstantu: $log_a n = log_b n \cdot log_a b$.
- Bez obzira na osnovu, kadgod se n udvostruči, $\log n$ poraste za konstantu, a $\log n$ se udvostruči tek kad se n kvadrira.

Primer

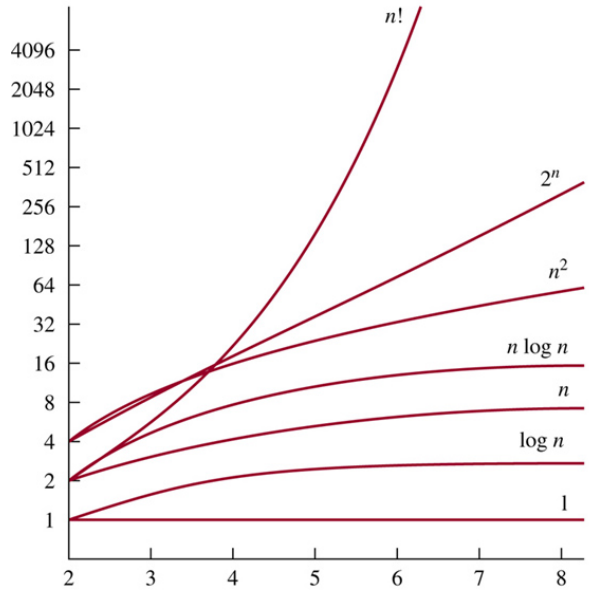
Oceniti vremensku složenost sledećeg koda. Odgovor obrazložiti.

```
int k=n;
while(k>0){
  printf("%d ", k);
  k=k/2;
}
```

Linearna složenost

- Kada se mala količina obrade izvrši nad svakim ulaznim podatkom, vreme izvršavanja programa je linearno.
- To je optimalno vreme izvršavanja za algoritam koji mora da obradi n ulaznih podataka ili da proizvede n izlaznih rezultata.

Linearna složenost



Kada je n milion, vreme izvršavanja je neka konstanta puta milion.
Kada se n udvostruči, udvostruči se i vreme izvršavanja.

Linearna složenost

- Složenost funkcije koja računa prosek k ulaznih brojeva ne zavisi od vrednosti tih brojeva, već samo od toga koliko ih ima i jednaka je $O(k)$.
- Linearna složenost može da se javi i u drugim situacijama
- Složenost funkcije za izračunavanje faktorijela ulazne vrednosti m zavisi od m i jednaka je (za razumnu implentaciju) $O(m)$.

Primer

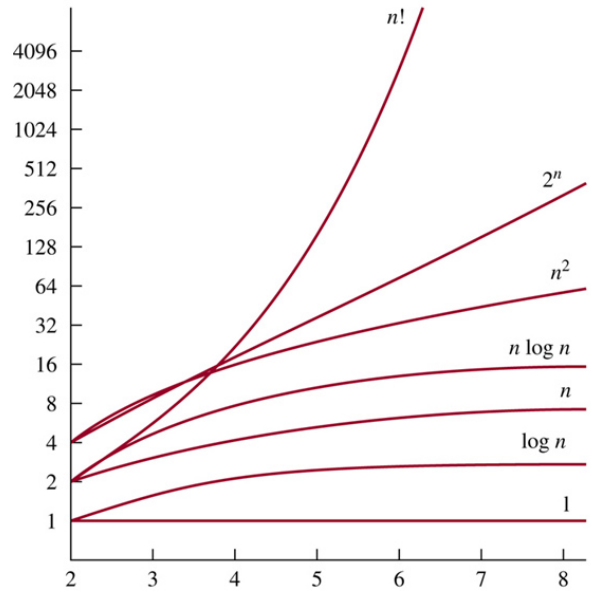
Oceniti vremensku i prostornu složenost sledećeg koda u odnosu na dimenziju n. Odgovor obrazložiti.

```
int suma(int niz[], int n) {
  int i, suma = 0;
  for(i=0; i<n; i++)
    suma+=niz[i];
  return suma;
}
```

$O(n \log n)$

- Vreme proporcionalno sa $n \log n$ utroše za svoje izračunavanje algoritmi koji problem rešavaju tako što ga razbiju u manje potprobleme, reše te potprobleme nezavisno jedan od drugog i zatim kombinuju rešenja.
- Ovakvu vremensku složenost ima, na primer, algoritam brzog sortiranja niza

$O(n \log n)$



Kada je n milion, $n \log n$ je oko dvadeset miliona. Kada se n udvostruči, vreme izvršavanja se poveća za više nego dvostruko (ali ne mnogo više).

Primer

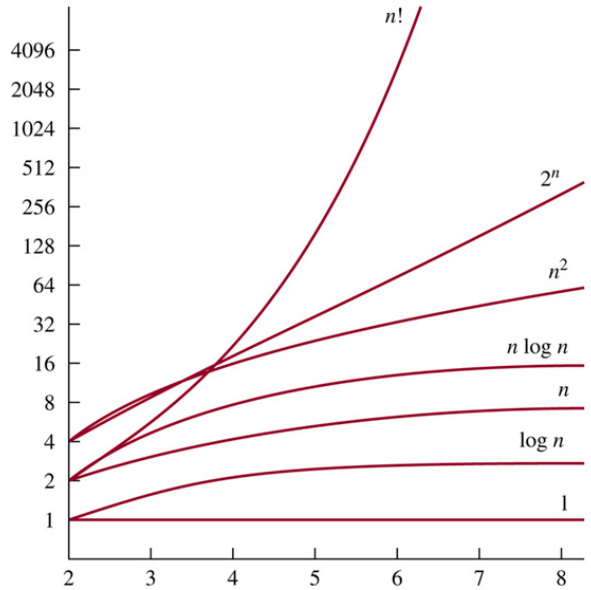
Oceniti vremensku složenost sledećeg koda. Odgovor obrazložiti.

```
for(i=0; i<n; i++) {
  int k=n;
  while(k>0){
    printf("%d ", k);
    k=k/2;
  }
}
```

Kvadratna složenost

- Kvadratno vreme izvršavanja nastaje za algoritme koji obrađuju sve parove ulaznih podataka (najčešće u dvostruko ugnježdenoj petlji).
- Algoritam sa ovakvim vremenom izvršavanja praktično je primenjivati samo za relativno male probleme (malo n).
- Ovakvu vremensku složenost imaju elementarni metodi sortiranja, na primer sortiranje izborom.
- Algoritam koji za n ulaznih tačaka proverava da li pripadaju unutrašnjosti n ulaznih trouglova, očekivano ima složenost $O(n^2)$.

Kvadratna složenost



Kada je n hiljadu, vreme izvršavanja je milion. Kada se n udvostruči, vreme izvršavanja se uveća četiri puta.

Primeri

Oceniti vremensku i prostornu složenost sledećeg koda u odnosu na dimenziju n. Odgovor obrazložiti.

```
for(i=0; i<n; i++)
  for(j=0; j<i; j++)
    a[i]+=j;
```

Kubna složenost

- Kubno vreme izvršavanja nastaje za algoritme koji obrađuju sve trojke ulaznih podataka (najčešće u trostruko ugnježdenoj petlji).
- Algoritam sa ovakvim vremenom izvršavanja praktično je primenjivati samo za male probleme (malo n).
- Kada je n sto, vreme izvršavanja je milion. Kada se n udvostruči, vreme izvršavanja se uveća osam puta.

Primeri

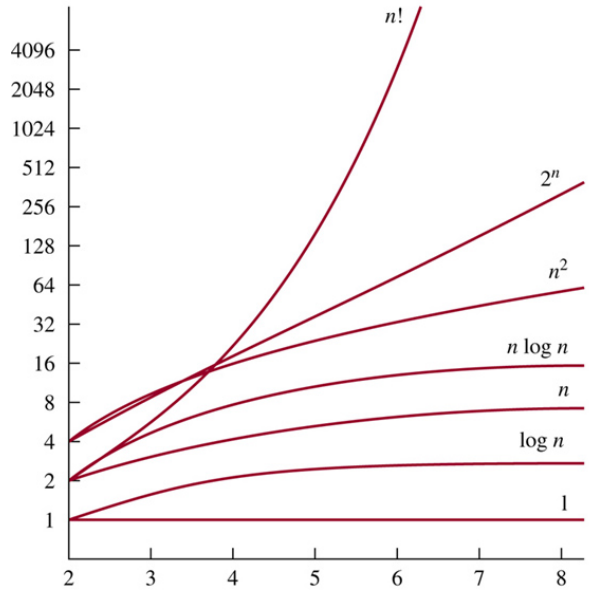
Oceniti vremensku i prostornu složenost sledećeg koda u odnosu na dimenziju n. Odgovor obrazložiti.

```
for(i=0; i<n; i++)
  for(j=0; j<n; j++)
    for(k=0; k<n; k++)
      a[i][j] = a[i][k]+a[k][j];
```

Eksponencijalna složenost $O(2^n)$

- Algoritmi sa eksponencijalnim vremenom izvršavanja nastaju prirodno pri primeni rešenja "grubom silom", tj. rešenja koja obično prvo padnu na pamet.
- Ipak, za neke probleme nisu nađeni (a sumnja se i da će ikada biti nađeni) algoritmi manje vremenske složenosti.
- Ovakvi algoritmi su vrlo retko praktični za upotrebu.

Eksponencijalna složenost $O(2^n)$



Kada je $n = 20$, vreme izvršavanja je milion. Kada se n udvostruči, vreme izvršavanja se kvadrira.

Izračunavanje složenosti funkcija

- Tačno određivanje (vremenske i prostorne) složenosti funkcija je najčešće veoma teško ili nemoguće, te se obično koriste razna pojednostavljivanja.
- U ovom kontekstu pojam „jedinična instrukcija“ se obično pojednostavljuje, te se može smatrati da, na primer, i poredjenje i sabiranje i množenje i druge pojedinačne naredbe troše po jednu vremensku jedinicu.

Izračunavanje složenosti funkcija

- Ukoliko se deo programa sastoji od nekoliko instrukcija bez grananja, onda se procenjuje da je njegovo vreme izvršavanja uvek isto, konstantno, te da pripada klasi $O(1)$.
- Ukoliko deo programa sadrži petlju koja se izvršava n puta, a vreme izvršavanja tela petlje je konstantno, onda ukupno vreme izvršavanja programa pripada klasi $O(n)$.
- Ukoliko deo programa sadrži jednu petlju koja se izvršava m puta i nakon nje jednu petlju koja se izvršava n puta, a vremena izvršavanja tela ovih petlji su konstantna, onda ukupno vreme izvršavanja petlje pripada klasi $O(m + n)$.

Izračunavanje složenosti funkcija

- Generalno, ukoliko program ima dva dela, složenosti $O(f)$ i $O(g)$, koji se izvršavaju jedan za drugim, ukupna složenost je $O(f + g)$.
- Ukoliko deo programa sadrži jedno grananje i ukoliko vreme izvršavanja jedne grane pripada klasi $O(m)$ a druge grane pripada klasi $O(n)$, onda ukupno vreme izvršavanja tog dela programa pripada klasi $max\{O(m), O(n)\}$ (što je jednako sa $O(m + n)$).
- Ukoliko deo programa sadrži dvostruku petlju – jednu koja se izvršava m puta i, unutar nje, drugu koja se izvršava n puta i ukoliko je vreme izvršavanja tela unutrašnje petlje konstantno, onda ukupno vreme izvršavanja petlje pripada klasi $O(m \cdot n)$.

Izračunavanje složenosti funkcija

- Analogno se računa složenost za druge vrste kombinovanja linearnog koda, grananja i petlji.
- Za izračanavanje složenosti rekurzivnih funkcija potreban je matematički aparat za rešavanje *rekurentnih jednačina*.

Izračunavanje složenosti funkcija

```
unsigned faktorijel(unsigned n) {
  if (n == 0)
    return 1;
  else
    return n*faktorijel(n-1);
}
```

- Neka $T(n)$ označava broj instrukcija koje zahteva poziv funkcije `faktorijel` za ulaznu vrednost `n`.
- Za $n = 0$, važi $T(n) = 2$ (jedno poređenje i jedna naredba `return`).
- Za $n > 0$, važi $T(n) = 4 + T(n-1)$ (jedno poređenje, jedno množenje, jedno oduzimanje, broj instrukcija koje zahteva funkcija `faktorijel` za `n-1` i jedna naredba `return`).
-

$$T(n) = 4 + T(n-1) = 4 + 4 + T(n-2) = \ldots = \underbrace{4 + 4 + \ldots + 4}_{n} + T(0) = 4n + 2 = O(n)$$

Navedena funkcija ima linearnu vremensku složenost.

Pristup "podeli i zavladaj"i master teorema o složenosti

- Naredna teorema govori o asimptotskom ponašanju rešenja nehomogene rekurentne jednačine oblika: $aT(n/b) + cn^k$

Teorema 4.1 (Master teorema o složenosti). *Rešenje rekurentne relacije*

$$T(n) = aT(n/b) + cn^k \ ,$$

gde su a i b celobrojne konstante takve da važi $a \geq 1$ i $b \geq 1$, i c i k su pozitivne realne konstante je

$$T(n) = \begin{cases} \Theta(n^{\log_b a}) \ , & ako\ je\ a > b^k \\ \Theta(n^k \log n) \ , & ako\ je\ a = b^k \\ \Theta(n^k) \ , & ako\ je\ a < b^k \end{cases}$$

- Algoritami tipa „podeli i zavladaj“ – rešavanje problema se svodi na rešavanje nekoliko problema iste vrste ali manje veličine ulaza.
- Konstanta a odgovara broju takvih potpoblema, b odgovara faktoru smanjenja veličine problema, a cn^k je vreme potrebno da se polazni problem podeli na potprobleme i da se rešenja potproblema objedine u rešenje polaznog problema.

Primer

Oceniti vremensku i prostornu složenost sledećeg koda u odnosu na dimenziju n. Odgovor obrazložiti.

```
int suma_prvih_5(int niz[], int n) {
  int i, suma = 0;
  for(i=0; i<n; i++) {
    if(i>4) break;
    suma+=niz[i];
  }
  return suma;
}
```

Primer

Oceniti vremensku i prostornu složenost sledećeg koda u odnosu na dimenziju n. Odgovor obrazložiti.

```
int suma_prvih_5(int niz[], int n) {
  int i, suma = 0;
  for(i=0; i<n; i++) {
    if(i>4) break;
    suma+=niz[i];
  }
  return suma;
}
```

- Vremenska složenost: $O(1)$
- Prostorna složenost: $O(1)$

Primer

Oceniti vremensku i prostornu složenost sledećeg koda u odnosu na dimenziju n. Odgovor obrazložiti.

```
int suma_parnih(int niz[], int n) {
  int i, suma = 0;
  for(i=0; i<n; i++) {
    if(niz[i]%2) continue;
    suma+=niz[i];
      }
      return suma;
}
```

Primer

Oceniti vremensku i prostornu složenost sledećeg koda u odnosu na dimenziju n. Odgovor obrazložiti.

```
int suma_parnih(int niz[], int n) {
  int i, suma = 0;
  for(i=0; i<n; i++) {
    if(niz[i]%2) continue;
    suma+=niz[i];
      }
      return suma;
}
```

- Vremenska složenost: $O(n)$
- Prostorna složenost: $O(1)$

Primeri

Oceniti vremensku i prostornu složenost sledećeg koda u odnosu na dimenziju n. Odgovor obrazložiti.

```
for(i=0; i<n; i++) {
  if(i<5) continue;
  for(j=0; j<n; j++)
    s++;
}
```

Primeri

Oceniti vremensku i prostornu složenost sledećeg koda u odnosu na dimenziju n. Odgovor obrazložiti.

```
for(i=0; i<n; i++) {
  if(i<5) continue;
  for(j=0; j<n; j++)
    s++;
}
```

- Vremenska složenost: $O(n^2)$
- Prostorna složenost: $O(1)$

Primeri

Oceniti vremensku i prostornu složenost sledećeg koda u odnosu na dimenziju n. Odgovor obrazložiti.

```
for(i=0; i<n; i++)
  for(j=0; j<i*i; j++) {
    k=1; m=n;
    while(m>k) {
      k=k*3;
      m=m/2;
    }
  }
```

Primeri

Oceniti vremensku i prostornu složenost sledećeg koda u odnosu na dimenziju n. Odgovor obrazložiti.

```
for(i=0; i<n; i++)
  for(j=0; j<i*i; j++) {
    k=1; m=n;
    while(m>k) {
      k=k*3;
      m=m/2;
    }
  }
```

- Vremenska složenost: $O(n^3 logn)$
- Prostorna složenost: $O(1)$

Primeri

Oceniti vremensku i prostornu složenost sledećeg koda u odnosu na dimenziju n. Odgovor obrazložiti.

```
int f(int n) {
  int x;
  if(n==0) return 1;
  x=n%10;
  if(x>1) return x*f(n/x);
  else return 2*f(n/2);
}
```

Primeri

Oceniti vremensku i prostornu složenost sledećeg koda u odnosu na dimenziju n. Odgovor obrazložiti.

```
int f(int n) {
  int x;
  if(n==0) return 1;
  x=n%10;
  if(x>1) return x*f(n/x);
  else return 2*f(n/2);
}
```

- Vremenska složenost: $O(log(n))$
- Prostorna složenost: $O(log(n))$

Primeri

Oceniti vremensku i prostornu složenost sledećeg koda u odnosu na dimenziju k. Odgovor obrazložiti.

```
float stepen_sporo(float x, unsigned k) {
  if (k == 0)
     return 1.0f;
  else
    return x * stepen_sporo(x, k - 1);
}
```

Primeri

Oceniti vremensku i prostornu složenost sledećeg koda u odnosu na dimenziju k. Odgovor obrazložiti.

```
float stepen_sporo(float x, unsigned k) {
  if (k == 0)
     return 1.0f;
  else
    return x * stepen_sporo(x, k - 1);
}
```

- Vremenska složenost: $O(k)$
- Prostorna složenost: $O(k)$

Primeri

Oceniti vremensku i prostornu složenost sledećeg koda u odnosu na dimenziju k. Odgovor obrazložiti.

```
float stepen_brzo(float x, unsigned k) {
  if (k == 0)
    return 1.0f;
  else if (k % 2 == 0)
    return stepen_brzo(x * x, k / 2);
  else
    return x * stepen_brzo(x, k - 1);
}
```

Primeri

Oceniti vremensku i prostornu složenost sledećeg koda u odnosu na dimenziju k. Odgovor obrazložiti.

```
float stepen_brzo(float x, unsigned k) {
  if (k == 0)
    return 1.0f;
  else if (k % 2 == 0)
    return stepen_brzo(x * x, k / 2);
  else
    return x * stepen_brzo(x, k - 1);
}
```

- Vremenska složenost: $O(logk)$
- Prostorna složenost: $O(logk)$

Primeri

Oceniti vremensku i prostornu složenost sledećeg koda u odnosu na dimenziju n. Odgovor obrazložiti.

```
int f(int n) {
  int x;
  if(n==0) return 1;
  x=n%10;
  y=(n/10)%10;
  return x*f(n-1)+y*f(n-1);
}
```

Primeri

Oceniti vremensku i prostornu složenost sledećeg koda u odnosu na dimenziju n. Odgovor obrazložiti.

```
int f(int n) {
  int x;
  if(n==0) return 1;
  x=n%10;
  y=(n/10)%10;
  return x*f(n-1)+y*f(n-1);
}
```

- Vremenska složenost: $O(2^n)$
- Prostorna složenost: $O(n)$

Primeri

Oceniti vremensku i prostornu složenost sledećeg koda u odnosu na dimenziju n. Odgovor obrazložiti.

```
int f(int n) {
  if(n<=0) return 1;
  return f(n-2)*f(n-2)*f(n-2)*f(n-2);
}
```

Primeri

Oceniti vremensku i prostornu složenost sledećeg koda u odnosu na dimenziju n. Odgovor obrazložiti.

```
int f(int n) {
  if(n<=0) return 1;
  return f(n-2)*f(n-2)*f(n-2)*f(n-2);
}
```

- Vremenska složenost: $O(4^{\frac{n}{2}}) = O(2^n)$
- Prostorna složenost: $O(n)$

Primeri

Oceniti vremensku i prostornu složenost sledećeg koda u odnosu na dimenziju n. Odgovor obrazložiti.

```
int f(int n) {
  if(n<=0) return 1;
  return f(n-2)*f(n-2);
}
```

Primeri

Oceniti vremensku i prostornu složenost sledećeg koda u odnosu na dimenziju n. Odgovor obrazložiti.

```
int f(int n) {
  if(n<=0) return 1;
  return f(n-2)*f(n-2);
}
```

- Vremenska složenost: $O(2^{\frac{n}{2}})$
- Prostorna složenost: $O(n)$

Primeri za vežbu

Oceniti vremensku i prostornu složenost sledećeg koda u odnosu na dimenziju n. Odgovor obrazložiti.

```
int f(int m, int n) {
  if(m>=n)
    return 1;
  return 2*f(m+1,n-1);
}
```

Primeri za vežbu

Oceniti vremensku i prostornu složenost sledećeg koda u odnosu na dimenziju n. Odgovor obrazložiti.

```
int f(int n) {
  if(n==0) return 1;
  if(n%2==0) return f(n/2);
  else return f(n/3);
}
```

Primeri za vežbu

Oceniti vremensku i prostornu složenost sledećeg koda u odnosu na dimenziju n. Odgovor obrazložiti.

```
for(i=0; i<n; i++)
  for(j=0; j<15; j++)
    s/=2;
```

Primeri za vežbu

Oceniti vremensku i prostornu složenost sledećeg koda u odnosu na dimenziju n. Odgovor obrazložiti.

```
for(i=0; i<20; i++)
  for(j=0; j<20; j++) {
    if(n%2==0)
      break;
    s+=n;
}
```

Primeri za vežbu

Oceniti vremensku i prostornu složenost sledećeg koda u odnosu na dimenziju n. Odgovor obrazložiti.

```
for(i=0; i<n; i++) {
  if(i>2)
    continue;
  for(j=0; j<n; j++)
    s++;
}
```

Primeri za vežbu

Formulisati algoritame i proceniti njihovu vremensku i prostornu složenost za sledeće probleme:

- Proizvod dva broja
- Proizvod niza brojeva
- Maksimum elemenata u nizu
- Kvadriranje elemenata u nizu
- Izračunavanje apsolutne vrednosti n-tog elementa u nizu
- Računanje n-tog Fibonačijevog broja

Primeri za vežbu

Formulisati algoritame i proceniti njihovu vremensku i prostornu složenost za sledeće probleme.

- Skalarni proizvod dva vektora
- Pronalaženje maksimalnog elementa matrice
- Pronalaženje minimalnog elementa za svaku vrstu matrice
- Zbir elemenata na dijagonali matrice
- Množenje matrica

Primeri za vežbu

Formulisati algoritame i proceniti njihovu vremensku i prostornu složenost za sledeće probleme:

- Stepenovanje prirodnog broja
- Određivanje da li element pripada nizu
- Sortiranje niza
- Određivanje binomnih koeficijenata

Kompromisi

- U opštem slučaju, postoji nagodba između prostora i vremena koje jedan problem zahteva za svoje rešavanje: on se ne može rešiti i za kratko vreme i uz malo korišćenje prostora.
- Obično se pravi neki kompromis u zavisnosti od izabranog algoritma: neki algoritmi za rešavanje problema koristiće malo vremena i puno prostora, drugi obratno.

Greške pri izboru algoritma

- Najčešća greška pri izboru algoritma je zanemarivanje karakteristika efikasnosti.
- Na primer, nekada je moguće naći neznatno komplikovaniji algoritam koji radi u vremenu $N \cdot logN$ od jednostavnog algoritma koji radi u vremenu N^2 - i slično u odnosu na prostor.
- Takođe greška je i pridati preveliki značaj efikasnosti.
- Na primer, bolji algoritam može da bude znatno kompleksniji i da ta kompleksnost otežava održavanje koda, dok je dobit u efikasnosti zanemarljiva (na primer, algoritam koji postiže bolje rezultate samo za neke vrlo velike N).

Pregled

Popravljanje vremenske složenosti

- Ukoliko performanse programa nisu zadovoljavajuće, treba razmotriti zamenu ključnih algoritama algoritama koji dominantno utiču na složenost.
- Ukoliko to ne uspeva tj. ukoliko se smatra da je asimptotsko ponašanje najbolje moguće, preostaje da se efikasnost programa popravi na polju broja pojedinačnih izvršenih naredbi (koje ne utiču na asimptotsko ponašanje, ali utiču na ukupno utrošeno vreme).

Popravljanje vremenske složenosti

- Koristiti optimizacije kompilatora
  - 0 — za podrazumevani režim kompilacije, samo sa bazičnim tehnikama optimizacije.
  - 1 — kompilator pokušava da smanji i veličinu izvršivog programa i vreme izvršavanja, ali ne primenjuje tehnike optimizacije koje mogu da bitno uvećaju vreme kompiliranja.
  - 2 — kompilator primenjuje tehnike optimizacije koje ne zahtevaju dodatni memorijski prostor u zamenu za veću brzinu (u fazi izvršavanja programa).
  - 3 — kompilator primenjuje i tehnike optimizacije koje zahtevaju dodatni memorijski prostor u zamenu za veću brzinu (u fazi izvršavanja programa). s kompilator primenjuje tehnike optimizovanja koje smanjuju izvršivi program, a ne njegovo vreme izvršavanja.

Popravljanje vremenske složenosti

- Ne optimizovati nebitne delove programa
- Izdvojiti izračunavanja koja se ponavljaju

```
x = x0*cos(0.01) - y0*sin(0.01);
y = x0*sin(0.01) + y0*cos(0.01);
```

Popravljanje vremenske složenosti

- Ne optimizovati nebitne delove programa
- Izdvojiti izračunavanja koja se ponavljaju

```
x = x0*cos(0.01) - y0*sin(0.01);
y = x0*sin(0.01) + y0*cos(0.01);
```

```
cs = cos(0.01);
sn = sin(0.01);
x = x0*cs - y0*sn;
y = x0*sn + y0*cs;
```

Popravljanje vremenske složenosti

- Izdvajanje koda izvan petlje

```
for (i=0; i < strlen(s); i++)
  if (s[i] == c)
    ...
```

Popravljanje vremenske složenosti

- Izdvajanje koda izvan petlje

```
for (i=0; i < strlen(s); i++)
  if (s[i] == c)
    ...

len = strlen(s);
for (i = 0; i < len; i++)
  if (s[i] == c)
    ...
```

Popravljanje vremenske složenosti

- Zameniti skupe operacije jeftinim
  Na primer, umesto
  `sqrt(x_1*x_1+y_1*y_1) > sqrt(x_2*x_2+y_2*y_2)`
  bolje je koristiti
  `x_1*x_1+y_1*y_1 > x_2*x_2+y_2*y_2`
  jer se njime izbegava pozivanje veoma skupe funkcije sqrt
- Ne ostavljati za fazu izvršavanja izračunavanja koja se mogu obaviti ranije
- Napisati kritične delove kôda na asembleru

Popravljanje prostorne složenosti

- Koristiti najmanje moguće tipove
- Ne čuvati ono što može da se lako izračuna

Pregled

Obnavljanje

- Kada se kaže da važi $f(n) \in O(g(n))$?

Obnavljanje

- Kada se kaže da važi $f(n) \in O(g(n))$?
- Da li funkcija $7n^2 + 3$ pripada sledećim klasama složenosti: $O(3n^2 + 1), O(n), O(n^2), O(n^3 logn), O(2logn)$?

Obnavljanje

- Kada se kaže da važi $f(n) \in O(g(n))$?
- Da li funkcija $7n^2 + 3$ pripada sledećim klasama složenosti: $O(3n^2 + 1), O(n), O(n^2), O(n^3 logn), O(2logn)$?
- Da li funkcija $3nlogn + 5n$ pripada sledećim klasama složenosti: $O(n), O(nlogn), O(n^2 logn), O(nlog^2(n)), O(logn), O(n + logn), O(15n)$?

Obnavljanje

- Kada se kaže da važi $f(n) \in O(g(n))$?
- Da li funkcija $7n^2 + 3$ pripada sledećim klasama složenosti: $O(3n^2 + 1), O(n), O(n^2), O(n^3 logn), O(2logn)$?
- Da li funkcija $3nlogn + 5n$ pripada sledećim klasama složenosti: $O(n), O(nlogn), O(n^2 logn), O(nlog^2(n)), O(logn), O(n + logn), O(15n)$?
- Da li funkcija $7n^2 + 3nlogn$ pripada sledećim klasama složenosti: $O(3n^2 + 1), O(n), O(n^2), O(n^3 logn), O(2logn)$?

Obnavljanje

- Kada se kaže da važi $f(n) \in O(g(n))$?
- Da li funkcija $7n^2 + 3$ pripada sledećim klasama složenosti: $O(3n^2 + 1), O(n), O(n^2), O(n^3 logn), O(2logn)$?
- Da li funkcija $3nlogn + 5n$ pripada sledećim klasama složenosti: $O(n), O(nlogn), O(n^2 logn), O(nlog^2(n)), O(logn), O(n + logn), O(15n)$?
- Da li funkcija $7n^2 + 3nlogn$ pripada sledećim klasama složenosti: $O(3n^2 + 1), O(n), O(n^2), O(n^3 logn), O(2logn)$?
- Da li funkcija $3nlogn + 5n + 100$ pripada sledećim klasama složenosti:
  $O(n), O(nlogn), O(n^2 logn), O(nlog^2 n), O(logn)O(n + logn), O(15n), O(108)$?

Obnavljanje

- Da li funkcija $n^6 + 2n + 10^{10}$ pripada sledećim klasama složenosti: $O(n), O(2^n), O(n^6), O(n^{10}), O(10^{10}), O(n^6 + 2^n), O(2^n + 10^{10}), O(2^n 10^{10})$?

Obnavljanje

- Da li funkcija $n^6 + 2n + 10^{10}$ pripada sledećim klasama složenosti: $O(n), O(2^n), O(n^6), O(n^{10}), O(10^{10}), O(n^6 + 2^n), O(2^n + 10^{10}), O(2^n 10^{10})$?
- Ako a(n) pripada klasi $O(nlogn)$, a b(n) pripada klasi $O(n^2)$, onda $a(n) + b(n)$ pripada klasi (izabrati sve ispravne odgovore): (a) $O(nlogn)$;

Obnavljanje

- Da li funkcija $n^6 + 2n + 10^{10}$ pripada sledećim klasama složenosti: $O(n), O(2^n), O(n^6), O(n^{10}), O(10^{10}), O(n^6 + 2^n), O(2^n + 10^{10}), O(2^n 10^{10})$?
- Ako a(n) pripada klasi $O(nlogn)$, a b(n) pripada klasi $O(n^2)$, onda $a(n) + b(n)$ pripada klasi (izabrati sve ispravne odgovore):
(a) $O(nlogn)$;
(b) $O(n^2)$;

Obnavljanje

- Da li funkcija $n^6 + 2n + 10^{10}$ pripada sledećim klasama složenosti: $O(n), O(2^n), O(n^6), O(n^{10}), O(10^{10}), O(n^6 + 2^n), O(2^n + 10^{10}), O(2^n 10^{10})$?
- Ako a(n) pripada klasi $O(nlogn)$, a b(n) pripada klasi $O(n^2)$, onda $a(n) + b(n)$ pripada klasi (izabrati sve ispravne odgovore):
  (a) $O(nlogn)$;
  (b) $O(n^2)$;
  (c) $O(nlogn + n^2)$;

Obnavljanje

- Da li funkcija $n^6 + 2n + 10^{10}$ pripada sledećim klasama složenosti: $O(n), O(2^n), O(n^6), O(n^{10}), O(10^{10}), O(n^6 + 2^n), O(2^n + 10^{10}), O(2^n 10^{10})$?
- Ako a(n) pripada klasi $O(nlogn)$, a b(n) pripada klasi $O(n^2)$, onda $a(n) + b(n)$ pripada klasi (izabrati sve ispravne odgovore):
  (a) $O(nlogn)$;
  (b) $O(n^2)$;
  (c) $O(nlogn + n^2)$;
  (d) $O(n^2 logn)$.

Obnavljanje

- Da li funkcija $n^6 + 2n + 10^{10}$ pripada sledećim klasama složenosti: $O(n), O(2^n), O(n^6), O(n^{10}), O(10^{10}), O(n^6 + 2^n), O(2^n + 10^{10}), O(2^n 10^{10})$?
- Ako a(n) pripada klasi $O(nlogn)$, a b(n) pripada klasi $O(n^2)$, onda $a(n) + b(n)$ pripada klasi (izabrati sve ispravne odgovore):
  (a) $O(nlogn)$;
  (b) $O(n^2)$;
  (c) $O(nlogn + n^2)$;
  (d) $O(n^2 logn)$.
- Ako a(n) pripada klasi $O(n^2)$, a b(n) pripada klasi $O(n^3)$, onda a(n) + b(n) pripada klasi (izabrati sve ispravne odgovore): (a) $O(n^2)$;

Obnavljanje

- Da li funkcija $n^6 + 2n + 10^{10}$ pripada sledećim klasama složenosti: $O(n), O(2^n), O(n^6), O(n^{10}), O(10^{10}), O(n^6 + 2^n), O(2^n + 10^{10}), O(2^n 10^{10})$?
- Ako a(n) pripada klasi $O(nlogn)$, a b(n) pripada klasi $O(n^2)$, onda $a(n) + b(n)$ pripada klasi (izabrati sve ispravne odgovore):
  (a) $O(nlogn)$;
  (b) $O(n^2)$;
  (c) $O(nlogn + n^2)$;
  (d) $O(n^2 logn)$.
- Ako a(n) pripada klasi $O(n^2)$, a b(n) pripada klasi $O(n^3)$, onda a(n) + b(n) pripada klasi (izabrati sve ispravne odgovore): (a) $O(n^2)$; (b) $O(n^3)$;

Obnavljanje

- Da li funkcija $n^6 + 2n + 10^{10}$ pripada sledećim klasama složenosti: $O(n), O(2^n), O(n^6), O(n^{10}), O(10^{10}), O(n^6 + 2^n), O(2^n + 10^{10}), O(2^n 10^{10})$?
- Ako a(n) pripada klasi $O(nlogn)$, a b(n) pripada klasi $O(n^2)$, onda $a(n) + b(n)$ pripada klasi (izabrati sve ispravne odgovore):
  (a) $O(nlogn)$;
  (b) $O(n^2)$;
  (c) $O(nlogn + n^2)$;
  (d) $O(n^2 logn)$.
- Ako a(n) pripada klasi $O(n^2)$, a b(n) pripada klasi $O(n^3)$, onda a(n) + b(n) pripada klasi (izabrati sve ispravne odgovore): (a) $O(n^2)$; (b) $O(n^3)$; (c) $O(n^5)$;

Obnavljanje

- Da li funkcija $n^6 + 2n + 10^{10}$ pripada sledećim klasama složenosti: $O(n), O(2^n), O(n^6), O(n^{10}), O(10^{10}), O(n^6 + 2^n), O(2^n + 10^{10}), O(2^n 10^{10})$?
- Ako a(n) pripada klasi $O(nlogn)$, a b(n) pripada klasi $O(n^2)$, onda $a(n) + b(n)$ pripada klasi (izabrati sve ispravne odgovore):
  (a) $O(nlogn)$;
  (b) $O(n^2)$;
  (c) $O(nlogn + n^2)$;
  (d) $O(n^2 logn)$.
- Ako a(n) pripada klasi $O(n^2)$, a b(n) pripada klasi $O(n^3)$, onda a(n) + b(n) pripada klasi (izabrati sve ispravne odgovore): (a) $O(n^2)$; (b) $O(n^3)$; (c) $O(n^5)$; (d) $O(n^6)$.

Obnavljanje

- Kada kažemo da je složenost algoritma A jednaka O(f(n))?

Obnavljanje

- Kada kažemo da je složenost algoritma A jednaka O(f(n))?
- Ako je složenost algoritma A za ulaznu vrednost n jednaka $O(n^3)$, a složenost algoritma B za ulaznu vrednost n jednaka $O(n^4)$, kolika je složenost algoritma C koji se izvršava tako što se izvršava prvo algoritam A, pa algoritam B: (a) $O(n^3)$;

Obnavljanje

- Kada kažemo da je složenost algoritma A jednaka O(f(n))?
- Ako je složenost algoritma A za ulaznu vrednost n jednaka $O(n^3)$, a složenost algoritma B za ulaznu vrednost n jednaka $O(n^4)$, kolika je složenost algoritma C koji se izvršava tako što se izvršava prvo algoritam A, pa algoritam B: (a) $O(n^3)$; (b) $O(n^4)$;

Obnavljanje

- Kada kažemo da je složenost algoritma A jednaka O(f(n))?
- Ako je složenost algoritma A za ulaznu vrednost n jednaka $O(n^3)$, a složenost algoritma B za ulaznu vrednost n jednaka $O(n^4)$, kolika je složenost algoritma C koji se izvršava tako što se izvršava prvo algoritam A, pa algoritam B: (a) $O(n^3)$; (b) $O(n^4)$; (c) $O(n^7)$;

Obnavljanje

- Kada kažemo da je složenost algoritma A jednaka O(f(n))?
- Ako je složenost algoritma A za ulaznu vrednost n jednaka $O(n^3)$, a složenost algoritma B za ulaznu vrednost n jednaka $O(n^4)$, kolika je složenost algoritma C koji se izvršava tako što se izvršava prvo algoritam A, pa algoritam B: (a) $O(n^3)$; (b) $O(n^4)$; (c) $O(n^7)$; (d) $O(n^{12})$.

Pregled

Literatura

- Slajdovi su pripremljeni na osnovu trećeg poglavlja knjige
  Predrag Janičić, Filip Marić: Programiranje 2
  i na osnovu četvrtog poglavlja skripte
  Gordana Pavlović-Lažetić: Programiranje 2
- Za pripremu ispita, slajdovi nisu dovoljni, neophodno je učiti iz knjige i skripte!