Analiza efikasnosti algoritama

Sadržaj:

1. Osnovne pretpostavke
   - RAM model
2. Asimptotske notacije
   - Θ, O, Ω - notacije
3. Analiza Insertion-sort algoritma
   - Provera korektnosti
   - Asimptotske notacije
4. Analiza rekurentnih jednačina
   - Rekurencije i načini rešavanja
   - Master teorema
5. Analiza Merge-sort algoritma
   - Provera korektnosti
   - Asimptotske notacije

Osnovne pretpostavke analize

- RAM (random-access machine) model
  - Algoritmi se izvršavaju kao računarski programi
  - Instrukcije unutar programa se izvršavaju jedna za drugom
  - Ne postoje konkurentne operacije
  - Memorijska hijerarhija:
    - Ne uključuje virtuelnu memoriju ni *cache* nivoe
  - Sadrži standardne mašinske instrukcije:
    - Aritmetičke: add, sub, mul, div, reminder, floor, ceilling itd.
    - Za rad sa podacima: load, store, copy
    - Za kontrolu toka: branch, call ,ret
  - Sve instrukcije traju konstantno vreme izvršenja
  - Podaci (*int* i *float*) su predstavljeni sa lg^n bita
    - Niz elementa: n x lg^n

Asimptotske notacije

- Služe za opis vremena izvršenja algoritma - $T(n)$
  - gde je $n \in \mathrm{N}$ veličina ulaznih podataka (npr. br. el. niza)
  - n dovoljno veliko da učini najdominantniji član relevantnim

- Npr. $T(n)=an^2+bn+c$
  - gde su a, b i c konstante

- Zanemarivanjem detalja ove funkcije može se proceniti da je asimptotsko vreme izvršenja tog algoritma neka funkcija od n^2
  - jer taj član najbrže raste sa n

Asimptotske notacije

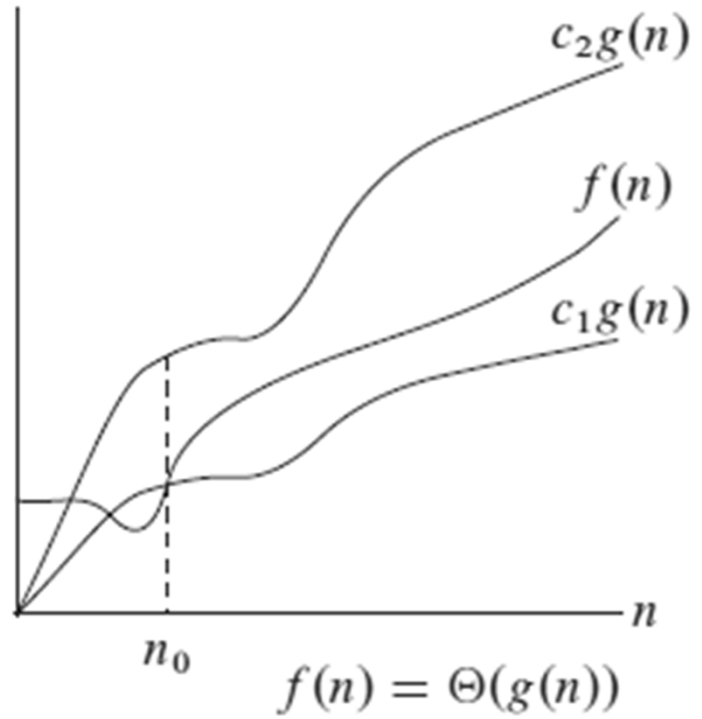
- Asimptotske notacije se mogu primeniti i na druge aspekte efikasnoti algoritma
  - Npr. zauzeće memorijskog prostora
- Ako se koristi za vreme, za koje vreme?
  - Vreme u najgorem slučaju (worst-case running time)
  - Vreme bez obzira na veličinu ulaznih podataka, itd.
- Postoji pet asimptotskih notacija: Θ-notacija, O-notacija, Ω-notacija, o-notacija i ω-notacija
- Mi se fokusiramo na 3 notacije: Θ-notacija, O-notacija, Ω-notacija

Θ-notacija

- Služi za određivanje vremena izvršenja algoritma u najgorem slučaju
  - Def.: Za zadatu funkciju $g(n)$, $\Theta(g(n))$ je *skup funkcija*, $\Theta(g(n))=\{f(n)\}$, takvih da postoje pozitivne konstante c_1, c_2 i n_0 za koje je: $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$, za svako $n \geq n_0$
- Kažemo: g(n) je ASIMPTOTSKI USKO OGRANIČENJE za f(n)
- g(n) mora biti ASIMPTOTSKI NENEGATIVNO
- Umesto $f(n) \in \Theta(g(n))$ pišemo $f(n)=\Theta(g(n))$

Θ-notacija

- Θ-notacija se zasniva na odbacivanju članova nižeg reda i zanemarivanju koeficijenta uz vodeći član
- Za bilo koji polinom $p(n)$ reda d, važi da je $p(n)=\Theta(n^d)$
  - Za $f(n)=an^2+bn+c$ je $f(n)=\Theta(n^2)$



O-notacija

- Služi za za definisanje ASIMPTOTSKI GORNJE GRANICE zadate funkcije
- Def.: za zadatu funkciju $g(n)$, $O(g(n))$ je *skup funkcija*, $O(g(n))=\{f(n)\}$, takvih da postoje pozitivne konstante c i n_0 za koje je:

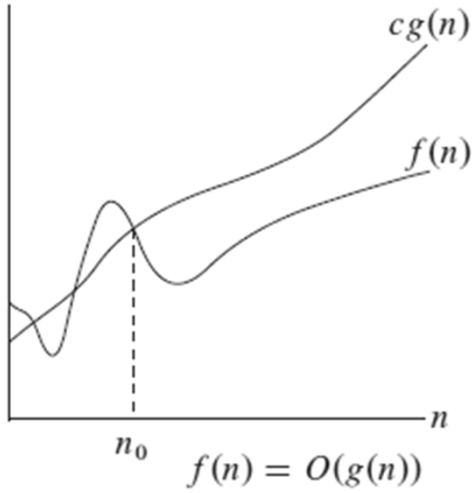
  $0 \leq f(n) \leq cg(n)$, za svako $n \geq n_0$

- $f(n)=an^2+bn+c$ je u $\Theta(n^2)$, takođe je i u $O(n^2)$
  - Po teoriji skupova $\Theta(g(n))\subseteq O(g(n))$
  - Θ-notacija je jača u odnosu na *O*-notaciju

O-notacija

- Može biti neobično da je npr. $n=O(n^2)$
- Vreme na osnovu ukupne strukture algoritma
  - Npr. za sledeći pseudo kod sa dve petlje očigledno je da je $O(n^2)$ gornja granica vremena izvršenja:

```
for(j=0; j<n; j++)
    for(k=0; (k<n) && condition; k++)
        // Neka O(1) obrada
```



  - Ova granica se može primeniti sa svaki ulaz
    - U zavisnosti od uslova usko graničenje je $\Theta(n^2)$ ili $\Theta(n)$

Ω-notacija

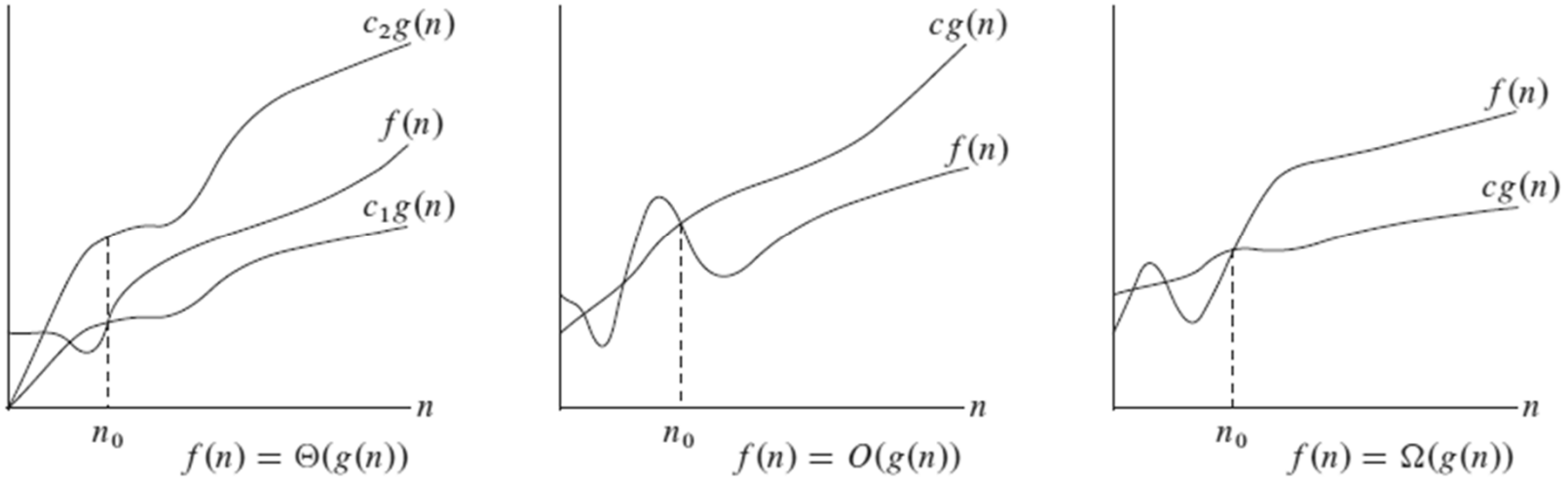
- Služi za za definisanje ASIMPTOTSKI DONJE GRANICE zadate funkcije
- Def.: Za zadatu funkciju $g(n)$, $\Omega(g(n))$ je *skup funkcija*, $\Omega(g(n))=\{f(n)\}$, takvih da postoje pozitivne konstante c i n_0 za koje je:

  $0 \leq cg(n) \leq f(n)$, za svako $n \geq n_0$

- $\Omega(g(n))$ daje vreme izvršenja algoritma u najboljem slučaju (best-case running time)

Teorema o asimptotskim notacijama

- Za bilo koje dve funkcije $f(n)$ i $g(n)$, važi da je $f(n)=\Theta(g(n))$ ako i samo ako je $f(n)=O(g(n))$ i $f(n)=\Omega(g(n))$



Asimptotske notacije u jednačinama i nejednačinama

- Već smo videli formulu $n=O(n^2)$
  - Slično bi mogli da napišemo $2n^2+3n+1=2n^2+\Theta(n)$
  - Kako se interpretiraju ovakve formule?

  - U $n=O(n^2)$, znak = označava: $n \in O(n^2)$
  - U većim formulama, asim. notacija predstavlja neku nepoznatu funkciju
  - Npr. formula $2n^2+3n+1=2n^2+\Theta(n)$ znači da je $2n^2+3n+1=2n^2+f(n)$, gde je $f(n)$ neka funkcija u skupu $\Theta(n)$

Asimptotske notacije u jednačinama i nejednačinama

- Broj anonimnih funkcija u nekom izrazu jednak broju pojava asimtotskih notacija.
  - Na primer, u izrazu $\Sigma\ O(i)$ postoji samo jedna anonimna funkcija, koja zavisi od i
- Šta ako se asimptotska notacija pojavljuje sa leve strane jednačine?
  - Npr. $2n^2 + \Theta(n) = \Theta(n^2)$
  - Pravilo: Kako god izabrali f-iju sa leve strane, postoji izbor f-ije sa desne strane, tako da je = zadovoljena
  - Za: $2n^2 + \Theta(n) = \Theta(n^2)$: za $f(n)$ postoji $g(n) \in \Theta(n^2)$, takva da je $2n^2 + f(n) = g(n)$ za svako n

Asimptotske notacije u jednačinama i nejednačinama

- Moguće je ulančati više ovakvih relacija
  - Npr. $2n^2+3n+1 = 2n^2 + \Theta(n) = \Theta(n^2)$

- Primenom gornjeg pravila svaka jednačina u lancu se interpretira nezavisno
  - Najpre: $2n^2+3n+1 = 2n^2 + f(n)$
  - A zatim: $2n^2 + g(n) = h(n)$, $g(n) \in \Theta(n)$, $h(n) \in \Theta(n^2)$
  - Zaključak: $2n^2+3n+1 = \Theta(n^2)$

Poređenje funkcija

- Relaciona svojstva Re brojeva važe i za asimptotska poređenja
  - Tranzitivnost i refleksivnost za svih 5 asim. not.
  - Simetričnost za Θ i transponovana sim O-Ω i o-ω
- Zato važi:
  - $f(n)=\Theta(g(n))$ je kao $a=b$
  - $f(n)=O(g(n))$ je kao $a\leq b$
  - $f(n)=\Omega(g(n))$ je kao $a\geq b$
  - $f(n)=o(g(n))$ je kao $a<b$
  - $f(n)=\omega(g(n))$ je kao $a>b$

Poređenje funkcija

- Kaže se da je:
  - $f(n)$ asimptotski manje od $g(n)$ ako je $f(n)=o(g(n))$ i
  - $f(n)$ asimptotski veće od $g(n)$ ako je $f(n)=\omega(g(n))$
- Trojakost ne važi!
  - Trojakost: za dva realna broja a i b, samo jedna od sledeće tri relacije može biti tačna: $a<b$, $a=b$ ili $a>b$
  - Ali, dve funkcije $f(n)$ i $g(n)$ mogu biti takve da za njih ne važi ni $f(n)=O(g(n))$ niti $f(n)=\Omega(g(n))$

Algoritam sortiranja sa umetanjem elemenata

- Sort za ulazni niz od n brojeva $\langle a_1, a_2, \ldots, a_n \rangle$ na izlazu daje permutaciju $\langle a_1', a_2', \ldots, a_n' \rangle$:

  $a_1' \leq a_2' \leq \ldots \leq a_n$‘ ($a_i$ se nazivaju ključevima)

- Eng. “insertion sort” je efikasan algoritma za sortiranje malog broja elemenata
  - Imitira način na koji čovek sorira karte u levoj ruci

- Karta u desnoj ruci se poredi sa kartama u levoj
  - s desna u levo

Algoritam sortiranja sa umetanjem elemenata (2/4)

- Karte u levoj ruci su sve vreme sortirane!
- Pripada klasi INKREMENTALNIH algoritama

- Procedura Isertion-Sort
  - Ulazni niz brojeva u nizu $A[1..n]$
  - Broj elemenata n je zadat atributom niza $A.length$
  - Sortira brojeve u istom tom nizu (eng. in place)
  - Na kraju procedure niz A sadrži sortiran niz brojeva

Algoritam sortiranja sa umetanjem elemenata (3/4)

```
INSERTION-SORT(A)
1  for j = 2 to A.length
2      key = A[j]
3      // Insert A[j] into the sorted sequence A[1 .. j − 1].
4      i = j − 1
5      while i > 0 and A[i] > key
6          A[i + 1] = A[i]
7          i = i − 1
8      A[i + 1] = key
```

- Indeks *j* odgovara tekućoj karti u desnoj ruci
- Elementi u *A*[1..*j*-1] odgovaraju sortiranim kartama u levoj ruci
- Elementi u *A*[*j*+1..*n*] odgovaraju kartama u špilu na stolu
- Osobina da je *A*[1..*j*-1] uvek sortiran se zove INVARIJANTA PETLJE

Provera koreknosti algoritma

- Invarijanta petlje se koristi za dokaz da je algoritam korektan
- Potrebno pokazati tri osobine invarijante petlje:
  - **Inicijalizacija**: Istinita je pre prve iteracije petlje
  - **Održavanje**: Ako je istinita pre iteracije petlje, ostaje istinita posle iteracije
  - **Završetak**: Kada se petlja završi, invarijanta daje korisnu osobinu za dokazivanje korektnosti
- Velika sličnost sa matematičkom indukcijom
  - Ovde se postupak zaustavlja kada se petlja završi

Provera korektnosti procedure Insertion-Sort

- **Inicijalizacija:**
  - Pre prve iteracije $j = 2$, podniz $A[1..j\text{-}1]$ je $A[1]$, koji je sortiran
- **Održavanje:**
  - Povećanje indeksa j za sledeću iteraciju ne utiče na već sortiran podniz $A[1..j\text{-}1]$
- **Završetak:**
  - Uslov da je $j > A.length = n$, izaziva završetak **for** petlje
    - Tada j mora imati vrednost $j = n + 1$
    - Zamenom u invarijantu: $A[1..j\text{-}1] = A[1..(n+1-1)] = A[1..n]$, a to je ceo niz!
    - Pošto je ceo niz sortiran, algoritam je korektan

Analiza algoritma

- Vreme izvršenja zavisi od veličine ulaza
  - sortiranje 10^3 brojeva traje duže od sortiranja 3 broja
- Vremena za dva niza brojeva iste veličine
  - zavisi koliko dobro su oni već sortirani
- Vreme izvršenja = broj operacija, tj. koraka
  - KORAK što je moguće više mašinski nezavistan
  - Npr. za svaku liniju pseudo koda potrebna neka konstantna količina vremena
  - Neka je c_i konstantno vreme potrebno za i-tu liniju

Analiza procedure Insertion-Sort

INSERTION-SORT(A)	*cost*	*times*
1 **for** $j = 2$ **to** $A.length$	c_1	n
2 $key = A[j]$	c_2	$n-1$
3 // Insert $A[j]$ into the sorted sequence $A[1..j-1]$.	0	$n-1$
4 $i = j-1$	c_4	$n-1$
5 **while** $i > 0$ and $A[i] > key$	c_5	$\sum_{j=2}^{n} t_j$
6 $A[i+1] = A[i]$	c_6	$\sum_{j=2}^{n}(t_j-1)$
7 $i = i-1$	c_7	$\sum_{j=2}^{n}(t_j-1)$
8 $A[i+1] = key$	c_8	$n-1$

- t_j je broj ispitivanja uslova **while** petlje u liniji 5
- Ispitivanje uslova petlje se izvršava jednom više nego telo petlje
- Vreme izvršenja $T(n)$ se dobija sabiranjem proizvoda:

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5 \sum_{j=2}^{n} t_j + c_6 \sum_{j=2}^{n}(t_j - 1) + c_7 \sum_{j=2}^{n}(t_j - 1) + c_8(n-1)$$

Analiza procedure Insertion-Sort

- Vreme izvršenja zavisi od toga kakav ulaz te veličine je zadat
- Najbolji slučaj se dobija kada je ulaz već sortiran
- Tada je $A[i] \leq key$, za svako j, u liniji 5, pa je $t_j = 1$, i najbolje (najkraće) vreme izvršenja je:

$$T(n) = c_1 n + c_2(n - 1) + c_4(n - 1) + c_5(n - 1) + c_8(n - 1) =$$
$$= (c_1 + c_2 + c_4 + c_5 + c_8)n - (c_2 + c_4 + c_5 + c_8) = an + b$$

- Ako je ulazni niz sortiran u obratnom redosledu, dobija se najgore (najduže) vreme izvršenja
- Svaki A[j] se mora porediti sa celim podnizom A[1..j-1], pa je $t_j = j$
- Uzimajući u obzir:

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

Analiza procedure Insertion-Sort

- sledi da je najgore vreme izvršenja:

$$T(n) = c_1 n + c_2(n - 1) + c_4(n - 1) + c_5\left(\frac{n(n+1)}{2} - 1\right) + c_6\left(\frac{n(n-1)}{2}\right) + \\ + c_7\left(\frac{n(n-1)}{2}\right) + c_8(n - 1) = \\ = an^2 + bn - c$$

- Najgore vreme izvršenja je dakle kvadratna funkcija veličine ulaza an2 + bn – c, gde konstante a, b i c zavise od cena iskaza

Analiza najgoreg slučaja i prosečnog slučaja

- Najčešće je od interesa samo vreme izvršenja algoritma u najgorem slučaju. 3 razloga:
  - To je gornja granica izvršenja za bilo koji ulaz – nikada se neće izvršavati duže
  - Najgori slučaj se dešava veoma često, npr. kod DB - najgori slučaj je da tražena informacija nije u DB
  - Prosečan slučaj je često loš skoro kao najgori slučaj
- Npr. za slučajno generisan niz brojeva
  - polovina elemenata u $A[1..j\text{-}1]$ je manja od $A[j]$, a polovina elemenata je veća, pa je $t_j = j/2$
  - vreme izvršenja takođe kvadratna funkcija n

Asimptotsko ponašanje algoritma

- Niz uprošćenja
  - Prvo su zanemarene cene iskaza, putem konstanti c_i
  - Zatim se prešlo na konstante a, b, c
- Dalje uprošćenje – asimptotske notacije
  - Procenjivanje stope rasta, ili VELIČINE RASTA, vremena izvršenja algoritma
  - Ranije je izveden zaključak da asimptotski uska granica za funkciju $f(n) = an^2 + bn - c$ iznosi $\Theta(n^2)$
  - Dakle, asimptotski uska granica za vreme izvršenja algoritma sortiranja sa umetanjem elemenat, u najgorem slučaju, je jednaka $\Theta(n^2)$

Rekurencije

- Rekurencije idu ruku pod ruku sa *podeli-i-zavladaj* algoritmima
- Rekurencije su jednačine ili nejednakosti koje definišu vrednost funkcije u odnosu na manje vrednosti ulaza
- Opšta rekurentna jednačina za $T(n)$ algoritma zasnovanog na pristupu *podeli-i-zavladaj*:

$$T(n) = \begin{cases} \Theta(1), & n \leq c \\ aT\left(\frac{n}{b}\right) + D(n) + C(n), & inače \end{cases}$$

Rekurencije

- Rekurzije mogu imati različite oblike:
  - $T(n) = T(2n/3) + T(n/3) + \Theta(n)$
    - Rekurzivni algoritam deli posao na podprobleme različite veličine
    - Podela na dva zadatka: 2/3 i 1/3 od ukupnog ulaza *n*
  - $T(n) = T(n-1) + \Theta(1)$
    - Algoritam pravi samo jedan podproblem koji sadrži samo jedan element manje u odnosu na početni ulaz
    - Svaki rekurzivni poziv traje konstantno vreme plus vreme izvršenja funkcije za ulaz *n-1*

Načini rešavanja rekurencije

- Uopšteno postoje tri metode za rešavanje rekurencija, tj. za određivanje Θ i O rešenja:
  - Metod zamene (*subtitution method*)
    - Pretpostavi se rešenje i matematičkom indukcijom se ono dokaže ili opovrgne
  - Metod rekurzivnog stabla (*recursion-tree method*)
    - Rekurencija se konvertuje u stablo čiji čvorovi na svakom nivou stabla predstavljaju utrošak potreban za rešavanje podproblema
  - Master metoda (master method)
    - Određuje granice za rekurencije oblika $T(n) = aT(n/b) + f(n)$

Master metoda

- Recept za rešavanje rekurentne jednačine oblika

  $T(n) = aT(n/b) + f(n)$

  $a \geq 1$ i $b > 1$, $f(n)$ neka asimptotski pozitivna funkcija
- Master metoda razlikuje tri slučaja
  - Lako rešavanje mnogih rekurentnih jednačina
- Jednačina opisuje vreme izvršenja algoritma
  - koji deli problem veličine n na a podproblema
  - svaki veličine n/b se rešava rekurzivno
  - vreme rešavanja podproblema je $T(n/b)$
  - $f(n)$ pokriva cenu deljenja problema na podprobleme i kombinovanja rešenja tih podproblema

Master teorema

- Neka je data rekurentna jednačina oblika:
  $T(n) = aT(n/b) + f(n)$
  $a \geq 1$ i $b > 1$, $f(n)$ neka asimptotski pozitivna funkcija

- Tada $T(n)$ ima sledeće asimptotske granice:
  - Ako je $f(n) = O(n^{\log_b a - \varepsilon})$ za neku konstantu $\varepsilon > 0$, onda je $T(n) = \Theta(n^{\log_b a})$
  - Ako je $f(n) = \Theta(n^{\log_b a})$, onda je $T(n) = \Theta(n^{\log_b a} \cdot \lg n)$, lg je $\log_2$
  - Ako je $f(n) = \Omega(n^{\log_b a + \varepsilon})$ za neku konstantu $\varepsilon > 0$, i ako je $af(n/b) \leq cf(n)$ za neku konstantu $c < 1$ i za sva dovoljno velika n, onda je $T(n) = \Theta(f(n))$

Tumačenje master teoreme

- U sva tri slučaja $f(n)$ se poredi sa $n^{\log_b a}$
  - Veća od ove dve funkcije određuje rešenje
- Prilikom upoređivanja voditi računa o sledećem:
  - U slučaju 1, $f(n)$ mora biti polinomijalno manja, za faktor n^{ε}
  - U slučaju 3, $f(n)$ mora biti polinomijalno veća i mora zadovoljiti tzv. uslov regularnosti $af(n/b) \leq cf(n)$
- Tri slučaja ne pokrivaju sve mogućnosti!
  - Postoje procepi između slučaja 1 i 2, i slučaja 2 i 3

Korišćenje master metode

- Prepozna se koji slučaj iz master teoreme važi, a onda se jednostavno napiše odgovor
- Primer 1: $T(n) = 9T(n/3) + n$
  - Rešenje: $a=9$, $b=3$, $f(n)=n$, $\log_b a=2$, $n^{\log_b a}=\Theta(n^2)$. Pošto je $n=O(n^{2-\varepsilon})$ za $\varepsilon=1$, u pitanju je slučaj 1, i rešenje je $T(n) = \Theta(n^2)$
- Primer 2: $T(n) = T(2n/3) + 1$
  - Rešenje: $a=1$, $b=3/2$, $f(n)=1$, $\log_b a=0$, $n^{\log_b a}=1$, Pošto je $1=\Theta(n^0)=\Theta(1)$, u pitanju je drugi slučaj i rešenje je $T(n) = \Theta(\lg n)$

Korišćenje master metode

- Primer 3: $T(n) = 3T(n/4) + n \lg n$
  - Rešenje: a=3, b=4, $f(n)= n \lg n$, $\log_b a$=0.793, $n^{\log_b a}=\Theta(n^{0.793})$. Pošto je $n \lg n=\Omega(n^{0.793+\varepsilon})$ za $\varepsilon\approx 0.2$, u pitanju je slučaj 3, ako je zadovoljen uslov regularnosti. Za dovoljno veliko n je: $af(n/b)=3(n/4)\lg(n/4)\leq(3/4)n \lg n = cf(n)$ za c=3/4. Pošto je uslov regularnost ispunjen,
    rešenje je $T(n) = \Theta(n \lg n)$.

Algoritam sortiranja sa spajanjem podnizova

- Koristi pristup PODELI I ZAVLADAJ
  - Ti algoritmi su rekurzivni
- Na svakom nivou rekurzije – sledeća 3 koraka:
  - **Podeli** problem na nekoliko podproblema, koji su manje instance istog problema
  - **Zavladaj** podproblemima rešavajući ih rekurzivno. Ako su podproblemi dovoljno mali, reši ih direktno
  - **Kombinuj** rešenja podproblema u ukupno rešenje originalnog problema

Algoritam sortiranja sa spajanjem podnizova

- Ovaj konkretan algoritam radi na sledeći način:
  - **Podeli**: Podeli niz od *n* elemenata u dva podniza od po *n*/2 elemenata
  - **Zavladaj**: Sortiraj dva podniza rekurzivno korišćenjem istog aloritma
  - **Kombinuj**: Spoj dva sortirana podniza da bi proizveo sortiran niz
- Rekurzija se spušta do dna, do niza dužine 1, a niz sa jednim elementom već je sortiran

Spajanje dva sortirana podniza

- Spajanje obavlja procedura Merge(A, p, q, r)
  - A je niz, a p, q i r su indeksi niza: $p \leq q < r$
  - Pretpostavka: $A[p..q]$ i $A[q+1..r]$ već sortirani
  - Merge ih spaja u jedan sortiran niz $A[p..r]$
- Potrebno $\Theta(n)$ vremena, $n = r - p+1$, to je broj elemenata koje treba spojiti
  - Dve gomile karata na stolu, okrenute licem na gore
  - Već sortirane, najmanja karta je na vrhu gomile
  - Spojiti u jednu sortiranu gomilu na stolu, okrenutu licem na dole

Osnovi korak procedure Merge

- Osnovni korak se sastoji od:
  - izbor manje karte sa vrhova dve polazne gomile,
  - Uklanjanje te karte sa vrha gomile (karta ispod nje postaje vidljiva)
  - Smeštanje karte dole na izlaznu gomilu
- Osnovni korak se ponavlja sve dok se jedna ulazna gomila ne isprazni; onda
  - 2-gu gomilu stavimo, licem na dole, na izlaznu gomilu
- Korak uzima $\Theta(1)$, pošto se samo porede 2 karte
  - n koraka ukupno uzima $\Theta(n)$ vremena

Specijalna karta

- Da li je neka od polaznih gomila ispražnjena?
  - Zamena: Da li se došlo do specijalne karte?
- U pseudo kodu ta specijalna vrednost je ∞
  - ona ne može biti manja ili jednaka ($\leq$) karta
  - osim ako se došlo do dna obe gomile
    - kad se to desi, sve karte pre specijalne karte su već prebačene na izlaznu gomilu
- Ukupno ima $r-p+1$ običnih karata
  - Procedura ponavlja osnovni korak toliko puta

Pseudo kod procedure Merge

```
MERGE(A, p, q, r)
 1  n1 = q − p + 1
 2  n2 = r − q
 3  let L[1..n1 + 1] and R[1..n2 + 1]
 4  for i = 1 to n1
 5       L[i] = A[p + i − 1]
 6  for j = 1 to n2
 7       R[j] = A[q + j]
 8  L[n1 + 1] = ∞
 9  R[n2 + 1] = ∞
10  i = 1
11  j = 1
12  for k = p to r
13       if L[i] ≤ R[j]
14            A[k] = L[i]
15            i = i + 1
16       else A[k] = R[j]
17            j = j + 1
```

- 1: računa dužinu n_1 podniza $A[p..q]$
- 2: računa dužinu n_2 podniza $A[q+1..r]$
- 3: pravi nizove L i R (levi i desni), dužine n_1+1 i n_2+1
- 4-5: kopira niz $A[p..q]$ u niz $L[1..n_1]$
- 6-7: kopira niz $A[q+1..r]$ u niz $R[1..n_2]$
- 8-9: smeštaju specijalnu vrednost ∞ na kraj nizova L i R
- 10-17: ponavljaju osnovni korak $r - p+1$ puta

Invarijanta petlje

- Na početku svake iteracije **for** petlje, lin. 12-17, $A[p..k\text{-}1]$ sadrži $k\text{-}p$ najmanjih elemenata iz $L[1..n_1+1]$ i $R[1..n_2+1]$, u sortiranom redosledu
  - Pored toga, $L[i]$ i $R[j]$ sadrže najmanje elemente njihovih nizova koji još nisu kopirani nazad u niz A

- Da se podsetimo: Provera korektnosti?
  - Odgovor: inicijalizacija, održavanje, završetak

Provera korektnosti algoritma

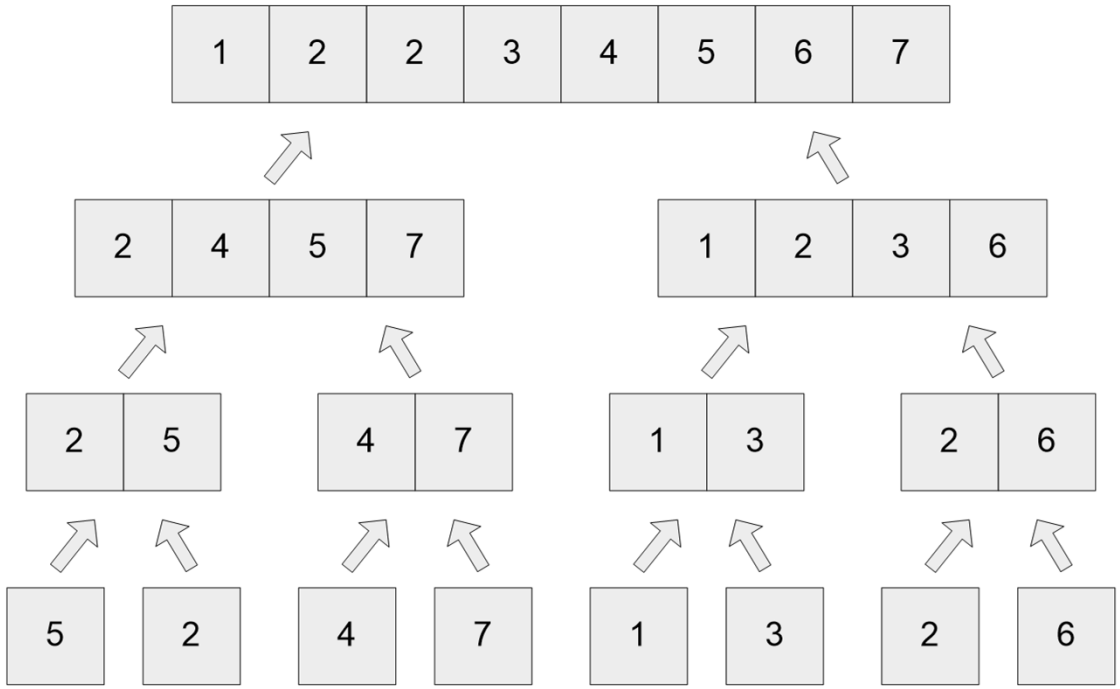
- Provera tri svojstva:
  - **Inicijalizacija**:
    - Pre I iteracije petlje, $k = p$, pa je niz $A[p..k\text{-}1]$ prazan. Prazan niz sadrži $k - p = 0$ min elem iz L i R; kako je $i = j = 1$, $L[1]$ i $R[1]$ su min elementi
  - **Održavanje**:
    - I deo: pp da je $L[i] \leq R[j]$, $L[i]$ je min elem. $A[p..k\text{-}1]$ sadrži $k - p$ min elem, a nakom kopira-nja $L[i]$ u $A[k]$, $A[p..k]$ će sadržati $k - p+1$ min elem. II deo: Ako je $L[i] > R[j]$, onda lin. 16-17 održavaju inv.
  - **Završetak**:
    - Na kraju je $k = r + 1$. $A[p..k\text{-}1]$, postaje $A[p..r]$, i sadrži $k - p = r - p + 1$ min elem. L i R zajedno sadrže $n_1+n_2+2 = r - p + 3$ elem, i svi oni su kopirani nazad u niz A, osim 2 spec. elem. ∞

Procedura Merge-Sort

```
MERGE-SORT(A, p, r)
1  if p < r
2      q = ⌊(p + r)/2⌋
3      MERGE-SORT(A, p, q)
4      MERGE-SORT(A, q + 1, r)
5      MERGE(A, p, q, r)
```

- Procedura Merge-Sort(*A*, *p*, *r*) sortira podniz *A*[*p*..*r*].
- Ako je *p*≥*r*, taj podniz ima najviše jedan element, pa je on već sortiran.
- U suprotnom slučaju, korak **podeli** jednostavno računa indeks *q*, koji deli *A*[*p*..*r*] na dva podniza: *A*[*p*..*q*] i *A*[*q*+1..*r*]
  - Prvi podniz sadrži ⌊n/2⌋ a drugi ⌈n/2⌉ elemenata
- Inicijalni poziv: Merge-Sort(*A*, 1, *A.length*)

Rad procedure Merge-Sort



- Ulazni niz (na dnu): <5,2,4,7,1,3,2,6>
- algoritam započinje spajanjem nizova sa po 1 elem u sortirane nizove dužine 2, itd.
- sve do spajanja dva niza dužine $n/2$ u konačnan niz dužine n

Analiza rekurzivnih algoritama

- Kada algoritam sadrži rekurzivne pozive, vreme izvršenja se opisuje rekurentnom jednačinom, ili kratko rekurencijom
- Npr. podeli i zavladaj
  - Direktno rešenje za $n \leq c$ uzima $\Theta(1)$ vreme
  - Podela problema na a podproblema, veličina $1/b$
  - Podproblem uzima $\Theta(n/b)$ vremena
  - a podproblema uzima $a\ \Theta(n/b)$ vremena
  - Podela problema na podprob. uzima $D(n)$ vremena
  - Kombinovanje rešenja uzima $C(n)$ vremena

Analiza rekurzivnih algoritama

- Opšta rekurentna jednačina za $T(n)$ algoritma zasnovanog na pristupu podeli i zavladaj:

$$T(n) = \begin{cases} \Theta(1), & n \leq c \\ aT\left(\frac{n}{b}\right) + D(n) + C(n), & inače \end{cases}$$

Analiza procedure Merge-Sort

- Vremena po koracima
  - **Podeli**: računa sredinu podniza, pa je $D(n) = \Theta(1)$
  - **Zavladaj**: dva podproblema, svaki veličine $n/2$, što daje doprinos ukupnom vremenu izvršenja od $2T(n/2)$
  - **Kombinuj**: Merge nad nizom od n elemenata uzima $\Theta(n)$ vremena, pa je $C(n) = \Theta(n)$
- Pošto je $C(n)+D(n)=\Theta(n)$, rekurentna jednačina za T(n) za proceduru Merge-Sort glasi:

$$T(n) = \begin{cases} \Theta(1), & n = 1 \\ 2T\left(\frac{n}{2}\right) + \Theta(n), & n > 1 \end{cases}$$

Rešenje

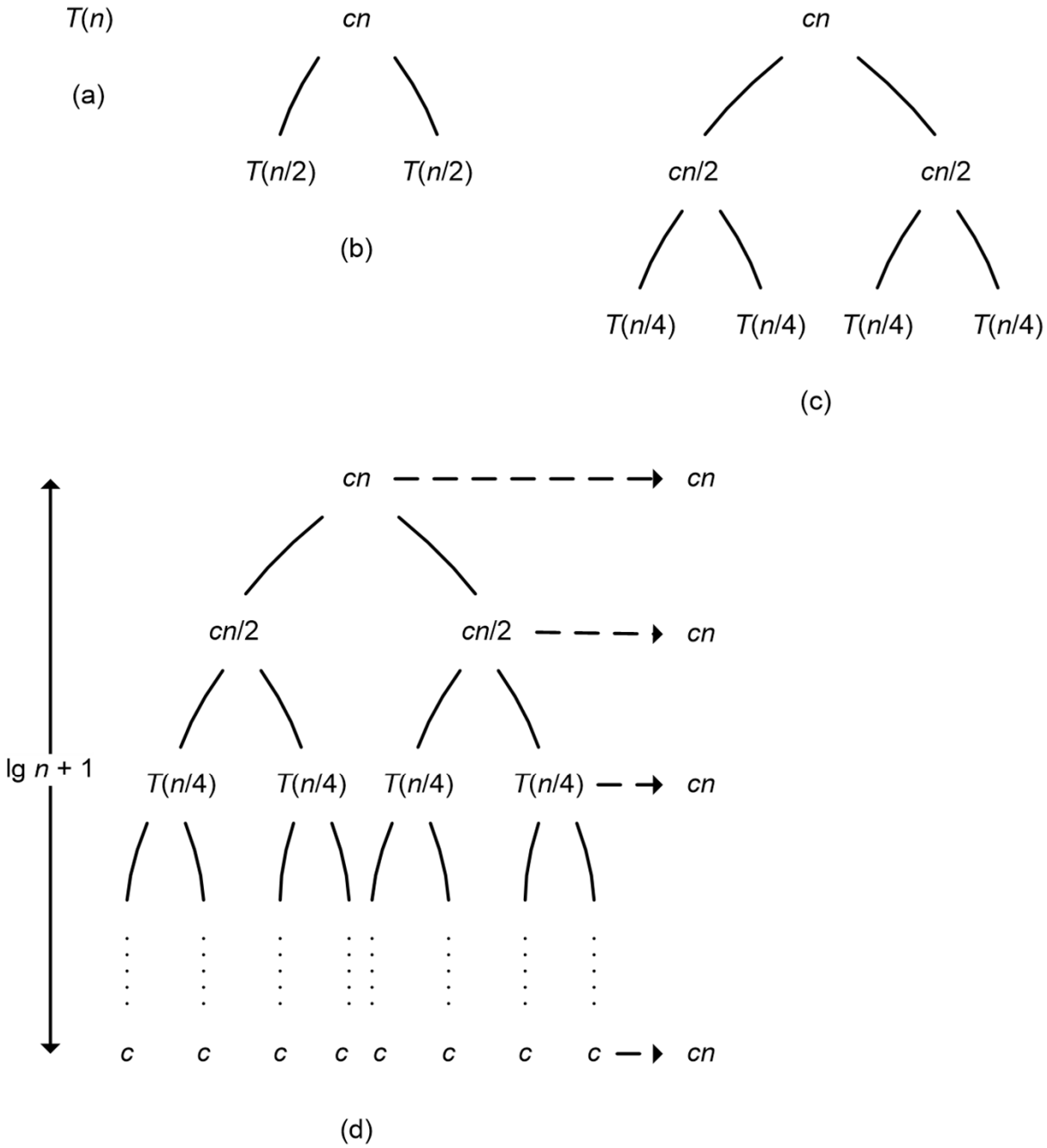
- Primenom master metode, slučaj 2, dobija se rešenje $T(n) = \Theta(n \lg n)$
- Intuitivno razumevanje rešenja$T(n) = \Theta(n \lg n)$ bez master teoreme
  - Napišimo gornju jednačinu ovako:

$$T(n) = \begin{cases} c, & n = 1 \\ 2T\left(\frac{n}{2}\right) + cn, & n > 1 \end{cases}$$

  - c je vreme za problem veličine 1, kao i vreme po elementu niza za korake podeli i kombinuj
  - Sledi grafičko rešenje poslednje jednačine
    - Pretpostavka: n je neki stepen za osnovu 2

Rešenje

- Rekurzivno stablo za $T(n) = 2T(n/2)+cn$
- (a): $T(n)$, koji se na (b) proširuje u stablo prema rekurenciji
- cn je cena na najvišem nivou, a dva podstabla su rekurencije$T(n/2)$
- (c): rezultat proširenja $T(n/2)$
- (d): celo rekurzivno stablo



Rešenje

- Dalje se sabiraju cene za svaki nivo stabla
  - najviši nivo: cena cn
  - sledeći novo: cena $c(n/2) + c(n/2) = cn$
  - sledeći novo: $c(n/4)+c(n/4)+c(n/4)+c(n/4) = cn$, itd.
  - najniži nivo: n čvorova x cena $c = cn$. Uvek cn.
- Ukupno nivoa u stablu: $\lg n + 1$, n broj listova
- Ukupna cena rekurencije:
  - Br nivoa x Cena nivoa = $(\lg n + 1)\, cn = cn \lg n + cn$
  - Zanemarujući niži član cn, kao i konstantu c, dobijamo:

    $T(n) = \Theta(n \lg n)$

Metod zamene

- Metodom zamene naći *O* rešenje za:
  - $T(n) = 2T(*n/2) + n$
    - *foor