Analiza efikasnosti algoritama II

- Alg. sortiranja sa umetanjem elemenata
- Alg. sortiranja sa spajanjem podnizova

Algoritam sortiranja sa umetanjem elemenata (1/4)

- Sort za ulazni niz od n brojeva $<a_1, a_2,...,a_n>$ na izlazu daje permutaciju $<a_1', a_2',...,a_n'>$:

  $a_1' \leq a_2' \leq ... \leq a_n'$ (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

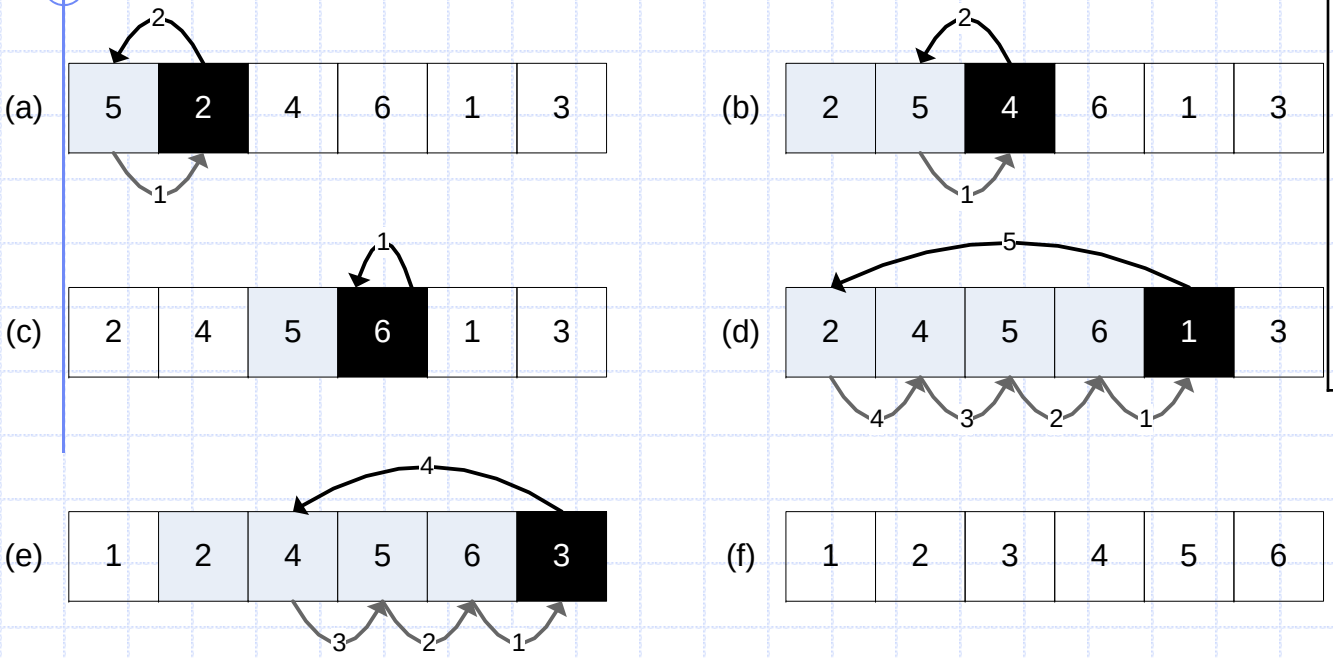
- Procedura Isertion-Sort
  - Ulazni niz brojeva u nizu $A[1..n]$
  - Broj elemenata n je zadat atributom niza $A.length$.
  - Sortira brojeve u istom tom nizu (eng. in place)
  - Na kraju procedure niz A sadrži sortiran niz brojeva

Algoritam sortiranja sa umetanjem elemenata (3/4)

```
Insertion-Sort(A)
1.for j = 2 to A.length
2.    key = A[j]
3.    // Umetni A[j] u sortirani 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

Algoritam sortiranja sa umetanjem elemenata (4/4)




```
Insertion-Sort(A)
1.for j = 2 to A.length
2.    key = A[j]
3.    // Umetni A[j] u 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
```

- Koraci (a) do (f) odgovaraju iteracijama **for** petlje, linije 1-8
- crna kockica = ključ uzet iz *A*[*j*], koji se poredi u liniji 5 sa brojevima u osenčenim kockicama s leve strane
- Svetlije strelice - pomeranje u liniji 6, Crna strelica prebacuje ključ u liniji 8 (numeracija prema redosledu premeštanja)

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: $A[1..j\text{-}1] = A[1..(n+1\text{-}1)] = A[1..n]$, 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 (1/3)

Insertion-Sort(A)	Cena	Broj izvršenja
1.for $j = 2$ **to** $A.length$	c_1	n
2. $key = A[j]$	c_2	$n - 1$
3. // Umetni $A[j]$ u $A[1..j\text{-}1]$	$c_3 = 0$	$n - 1$
4. $i = j - 1$	c_4	$n - 1$
5. **while** $i > 0$ and $A[i] > key$	c_5	$\Sigma_{j=2..n}\, t_j$
6. $A[i + 1] = A[i]$	c_6	$\Sigma_{j=2..n}\, (t_j - 1)$
7. $i = i - 1$	c_7	$\Sigma_{j=2..n}\, (t_j - 1)$
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 (2/3)

- 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 $A[1..j\text{-}1]$, 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 (3/3)

- 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 $an^2 + bn - c$, 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 $\Theta(n^2)$

Algoritam sortiranja sa spajanjem podnizova (1/2)

- 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 (2/2)

- 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 (≤) 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

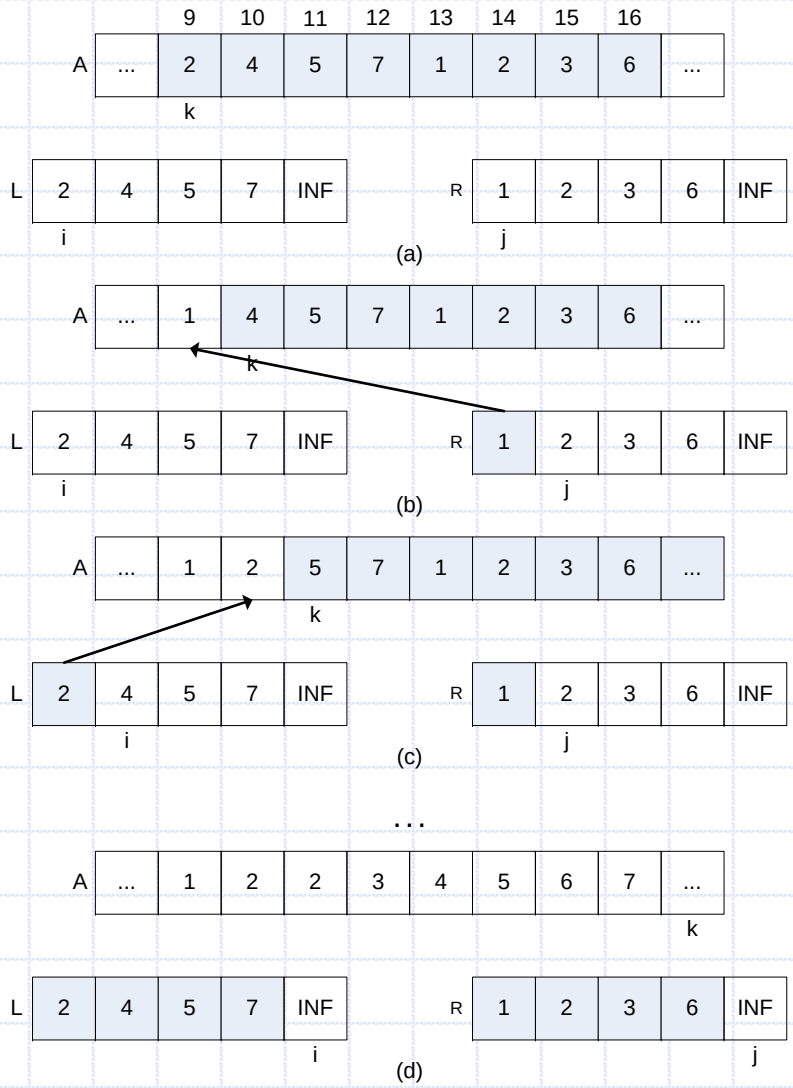
Pseudo kod procedure Merge

```
Merge(A, p, q, r)
1.n1 = q – p + 1
2.n2= r – q
3.dodeli L[1.. n1+1] i 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

Ilustracija rada procedure Merge

poziv Merge(*A*, 9, 12, 16), niz *A*[9..16] sadrži brojeve <2,4,5,7,1,2,3,6>



- Nakon kopiranja i umetanja specijalne vrednosti, niz *L* sadrži <2, 4, 5, 7, ∞>, niz R sadrži <1, 2, 3, 6, ∞>
- Neosenčene pozicije u *A* su njihove konačne vrednosti
- Osenčene pozicije u *L* i *R* su kopirane u *A*
- Indeks *k* pozicija u izl. nizu *A*
- Indeks *i* pozicija u ul. nizu *L*
- Indeks *j* pozicija u ul. nizu *R*
- Slika (b): *L*[1]=2 i *R*[1]=1, *R*[1] sadrži manju vrednost (1), koja se kopira u *A*[9]
- O korak se ponavlja još 6 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

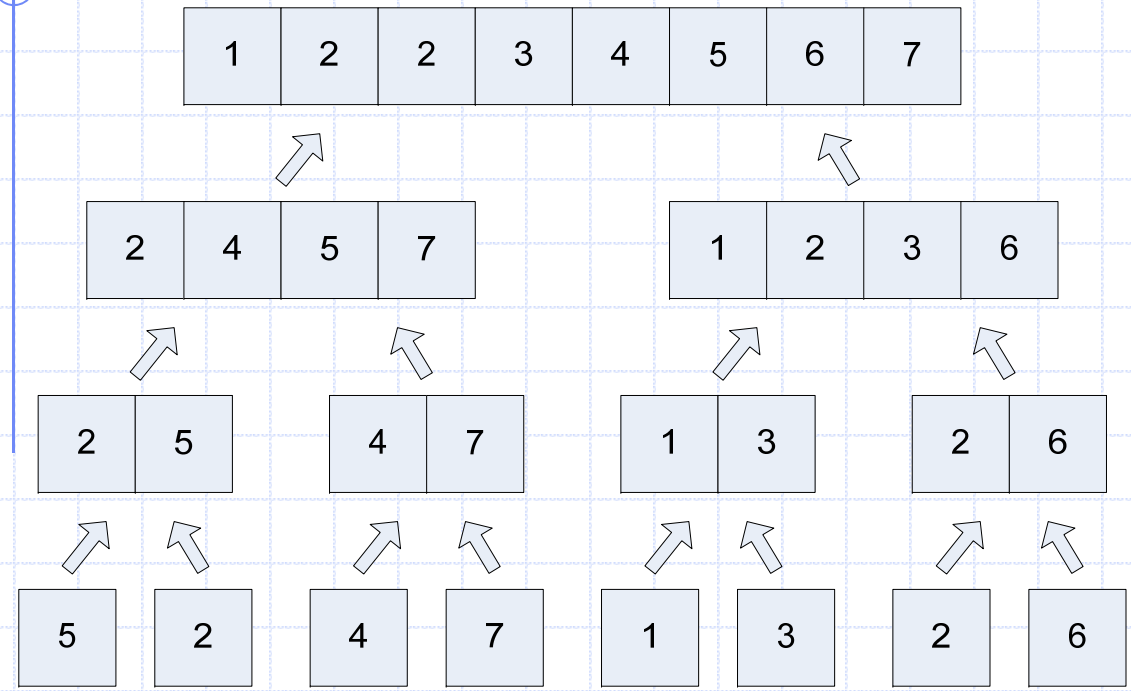
- Provera tri svojstva:
  - **Inicijalizacija**: Pre I iteracije petlje, $k = p$, pa je niz $A[p..k\text{-}1]$ prazan. Prazan niz sadrži $k - p = 0$ min elem iz L i R; kako je $i = j = 1$, $L[1]$ i $R[1]$ su min elementi
  - **Održavanje**: I deo: pp da je $L[i] \leq R[j]$, $L[i]$ je min elem. $A[p..k\text{-}1]$ sadrži $k - p$ min elem, a nakom kopiranja $L[i]$ 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 (1/2)

- 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 (2/2)

- 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 (1/3)

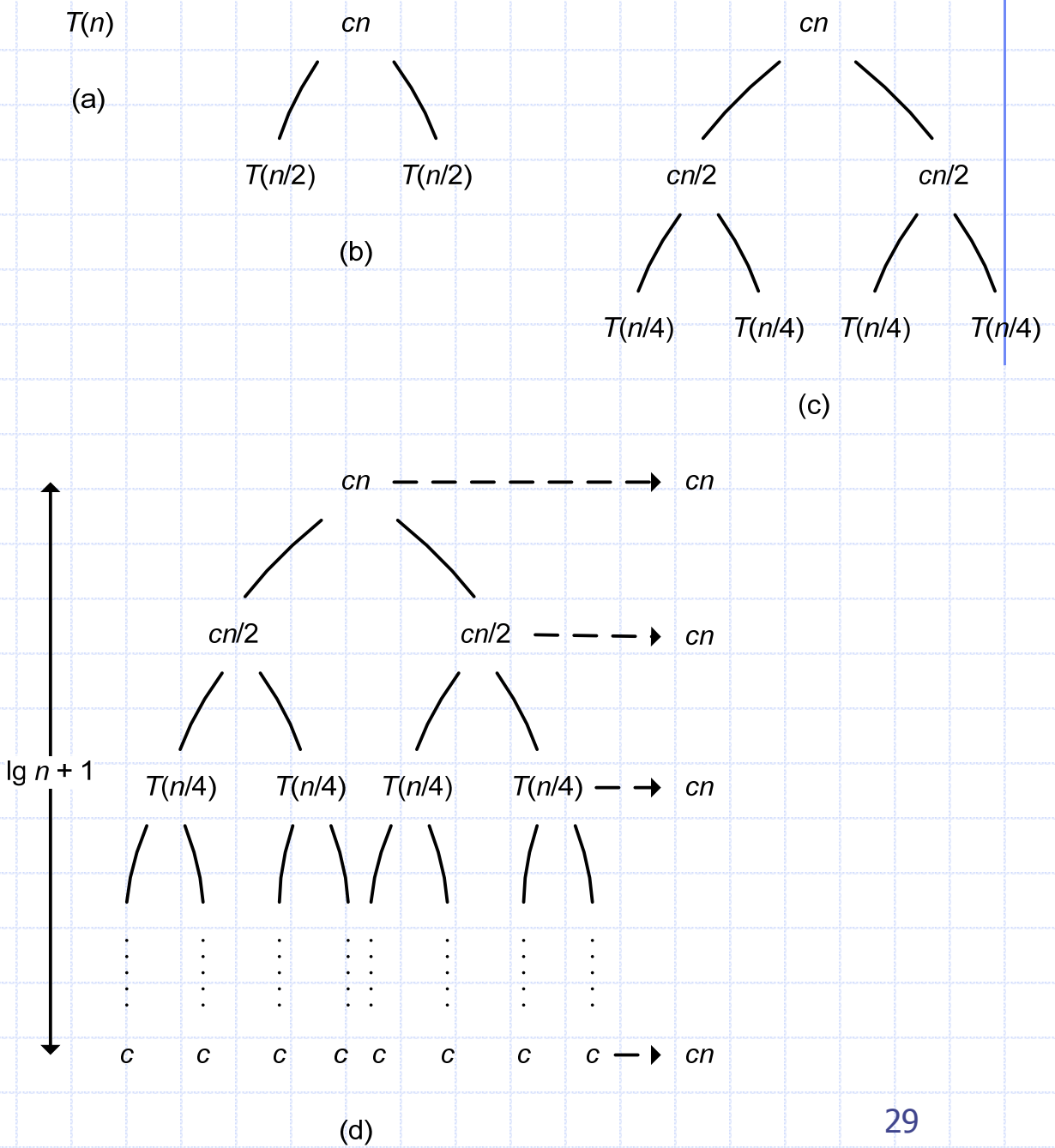
- Primenom master metode, slučaj 2, dobija se rešenje $T(n) = \Theta(n \lg n)$
- Intuitivno razumevanje rešenja $T(n) = \Theta(n \lg n)$ bez master teoreme
  - Napišimo gornju jednačinu ovako:

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

  - 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 (2/3)

- 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 (3/3)

- 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:

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