

# Algoritmi

Pretrage i sortiranja

# Sortiranje i pretraga

- Da bi pretraga bila brza prethodno sortirati podatke!
  - Kada se pretraga koristi? Veoma često ...
  - Kada se sortiranje isplati? Ako je pretraga česta.
  - Kako se sortira? Algoritmom za sortiranje.
  - Kako se pretražuje? Algoritmom binarne pretrage.
- Algoritmi za sortiranje:
  - Sortiranje izborom – *Selection sort*
  - Sortiranje umetanjem – *Insertion sort*
  - Sortiranje objedinjavanjem – *Merge sort*
  - Sortiranje razdvajanjem – *Quicksort*
  - ...
- Složenost algoritama sortiranja je  $\Theta(n^2)$  ili  $\Theta(n \log_2 n)$  (kasnije detaljnije)

# Binarna pretraga

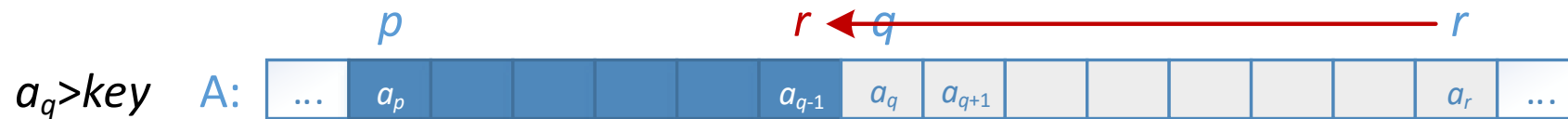
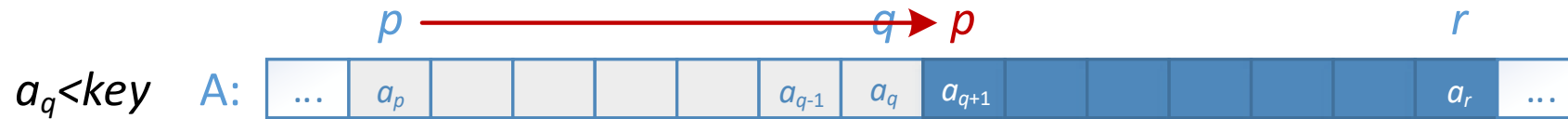
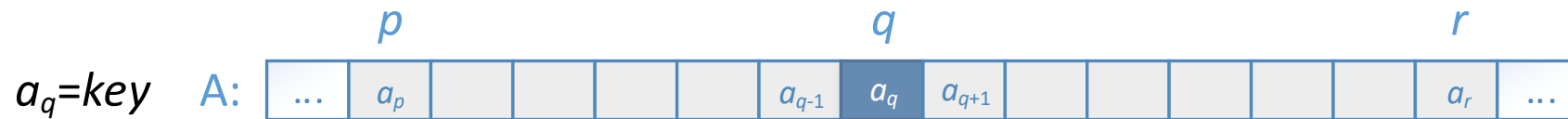
- Zahteva da je sortiran niz koji se pretražuje.
- Brzina algoritma je  $O(\log_2 n)$
- Ideja algoritma
  - Kako su podaci (ključevi) sortirani u npr. rastućem redosledu prvo očitamo vrednost na sredini niza i
    1. Ako je to tražena vrednost onda je pretraga gotova
    2. Ako je ta vrednost manja od tražene onda je tražena vrednost u desnoj polovini niza
    3. Ako je ta vrednost veća od tražene onda je tražena vrednost u levoj polovini niza
  - Time se prepolovi deo niza od interesa i postupak se sprovodi iterativno u podnizovima koji se smanjuju (prepolovljavaju) sve dok ne ostane samo jedan element u podnizu.

$n$	$\log_2(n)$
1.024	10
1.048.576	20
1.073.741.824	30

# Binarna pretraga (2)

- U svakoj iteraciji se posmatra deo podataka od indeksa  $p$  do indeksa  $r$
- Posmatra se sredina niza

$$q = \left\lfloor \frac{p+r}{2} \right\rfloor$$



# Binarna pretraga - algoritam

```
BINARY-SEARCH(A, key)
1  p = 1                // leva granica
2  r = A.Length         // desna granica
3  while p <= r
4      q =  $\lfloor (p + r) / 2 \rfloor$     // sredina
5      if A[q] == key
6          return q        // bingo!
7      elseif A[q] > key
8          r = q - 1
9      else
10         p = q + 1
11 return nije-nađen
```

# Binarna pretraga – rekurzivni algoritam

BINARY-SEARCH(*A*, *key*)

1 RECURSIVE-BINARY-SEARCH(*A*, 1, *A.Length*, *key*)

RECURSIVE-BINARY-SEARCH(*A*, *p*, *r*, *key*)

1 **if** *p* > *r*

2     **return** *nije-nađen*

3 **else**

4      $q = \lfloor (p + r) / 2 \rfloor$

5     **if** *A*[*q*] == *key*

6         **return** *q*

7     **elseif** *A*[*q*] > *key*

8         RECURSIVE-BINARY-SEARCH(*A*, *p*, *q*-1, *key*)

9     **else**

10         RECURSIVE-BINARY-SEARCH(*A*, *q*+1, *r*, *key*)

# Primer binarne pretrage

*key*  
↓

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	3	4	7	11	13	14	17	23	25	26	41	44	45	69	77
1	3	4	7	11	13	14	17	23	25	26	41	44	45	69	77
1	3	4	7	11	13	14	17	23	25	26	41	44	45	69	77
1	3	4	7	11	13	14	17	23	25	26	41	44	45	69	77

# Sortiranje izбором (*Selection sort*)

SELECTION-SORT(A)

1 **for**  $i = 1$  **to**  $A.Length - 1$

2      $indMin = i$

3     **for**  $j = i + 1$  **to**  $A.Length$

4         **if**  $A[j] < A[indMin]$

5              $indMin = j$

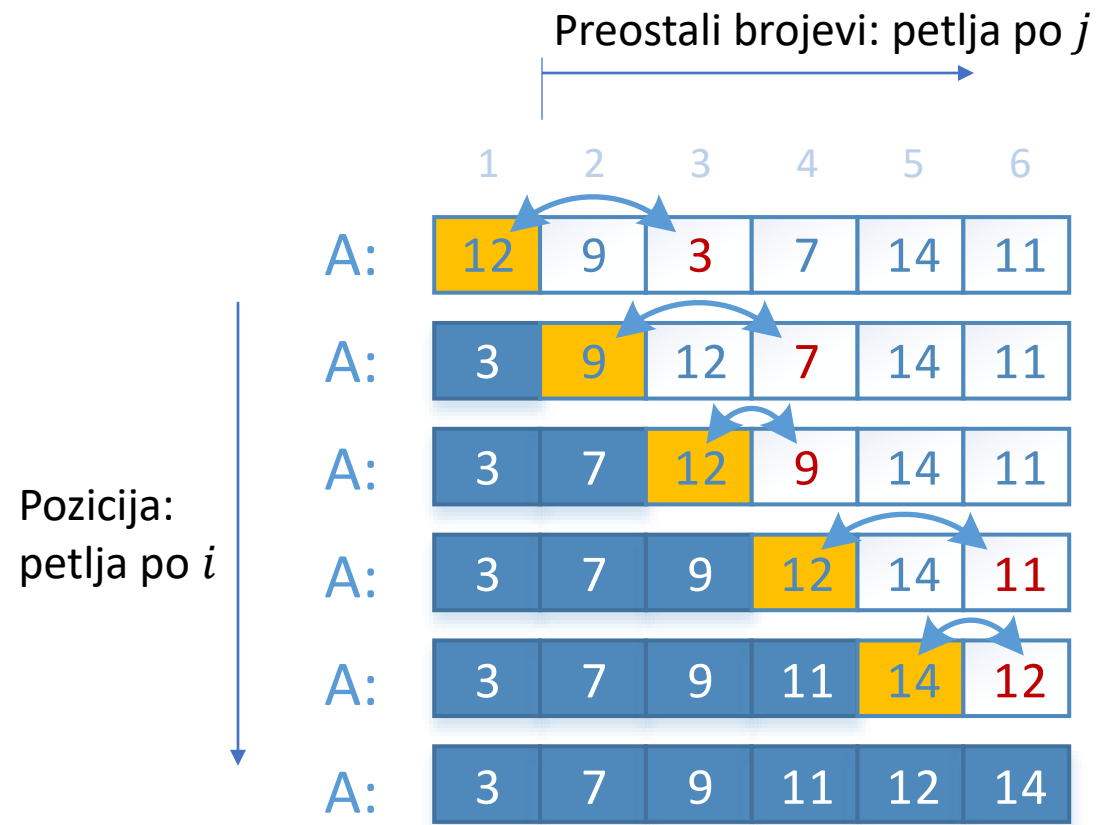
6      $A[i] \leftrightarrow A[indMin]$          // zameni



# Primer *Selection sort*

**SORTIRAJ - IZBOROM(A)**

```
1  for i = 1 to A.Length-1
2      indMin = i
3      for j = i+1 to A.Length
4          if A[j] < A[indMin]
5              indMin = j
6      A[i] ↔ A[indMin]
```



Za svaku poziciju (do pretposlednje) bira se najmanji broj od preostalih (iz belih „kućica“) da se postavi na poziciju naranđastog.

Primetiti da su levo od pozicije sortirani brojevi.

Za poslednji broj nema šta da se radi jer je on najveći.

# Vreme izvršavanja *Selection sort*

- Algoritam ima  $n - 1$  spoljašnju iteraciju (spoljašnja petlja)
  - U okviru svake iteracije postoji unutrašnja petlja od  $n - i$  prolaza
- Ukupan broj unutrašnjih iteracija je

$$(n-1) + (n-2) + (n-3) + \dots + 2 + 1 =$$

$$\frac{(n-1) \cdot n}{2} = \frac{1}{2}n^2 - \frac{1}{2}n = \Theta(n^2)$$

- Odakle vidimo da je složenost algoritma  $\Theta(n^2)$
- Dodatna osobina:  
Maksimalan broj zamena elemenata u nizu je  $n - 1$ , tj.  $\Theta(n)$

```
SORTIRAJ-IZBOROM(A)
1  for i = 1 to A.Length-1
2      indMin = i
3      for j = i+1 to A.Length
4          if A[j] < A[indMin]
5              indMin = j
6      A[i] ↔ A[indMin]
```

# Sortiranje umetanjem (*Insertion sort*)

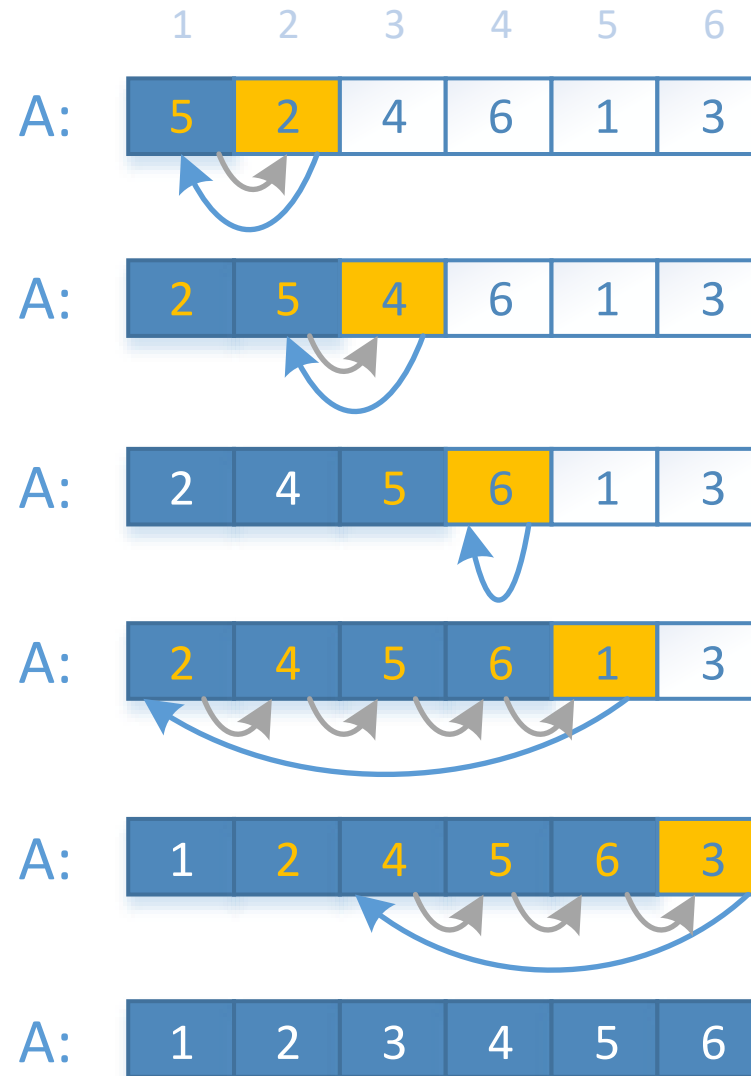
INSERTION-SORT(*A*)

```
1  for j = 2 to A.Length
2      key = A[j]
3      i = j-1
4      while i>0 and A[i]>key
5          A[i+1] = A[i]
6          i = i-1
7      A[i+1] = key
```

- Ideja algoritma je slična postupku ređanja karata u ruci.



# Primer *Insertion sort*



SOFTIRAJ-UMETANJEM( $A$ )

```
1  for  $j = 2$  to  $A.Length$ 
2       $key = A[j]$ 
3       $i = j - 1$ 
4      while  $i > 0$  &  $A[i] > key$ 
5           $A[i+1] = A[i]$ 
6           $i = i - 1$ 
7       $A[i+1] = key$ 
```

# Vreme izvršavanja *Insertion sort*

INSERTION-SORT(A)

TRAJANJE BROJ PROLAZA

```
1  for  $j = 2$  to  $A.Length$ 
2       $key = A[j]$ 
3       $i = j - 1$ 
4      while  $i > 0$  and  $A[i] > key$ 
5           $A[i+1] = A[i]$ 
6           $i = i - 1$ 
7       $A[i+1] = key$ 
```

# Vreme izvršavanja *Insertion sort*

INSERTION-SORT(A)		TRAJANJE	BROJ	PROLAZA
1	<b>for</b> $j = 2$ <b>to</b> $A.Length$	$c_1$		$n$
2	$key = A[j]$	$c_2$		$n-1$
3	$i = j-1$	$c_3$		$n-1$
4	<b>while</b> $i > 0$ and $A[i] > key$	$c_4$		$\sum t_j, j=2..n$
5	$A[i+1] = A[i]$	$c_5$		$\sum t_j - 1, j=2..n$
6	$i = i-1$	$c_6$		$\sum t_j - 1, j=2..n$
7	$A[i+1] = key$	$c_7$		$n-1$

$$T(n) = c_1 n + c_2 (n-1) + c_3 (n-1) + c_4 \sum_{j=2}^n t_j + c_5 \sum_{j=2}^n (t_j - 1) + c_6 \sum_{j=2}^n (t_j - 1) + c_7 (n-1)$$

# Vreme izvršavanja *Insertion sort* (2)

$$T(n) = c_1n + c_2(n-1) + c_3(n-1) + c_4 \sum_{j=2}^n t_j + c_5 \sum_{j=2}^n (t_j - 1) + c_6 \sum_{j=2}^n (t_j - 1) + c_7(n-1)$$

- Najbolji slučaj (A je na početku sortirano),  $t_j = 1$

$$T(n) = c_1n + c_2(n-1) + c_3(n-1) + c_4(n-1) + c_7(n-1)$$

$$T(n) = (c_1 + c_2 + c_3 + c_4 + c_7)n - (c_2 + c_3 + c_4 + c_7)$$

$$T(n) = \Omega(n)$$

# Vreme izvršavanja *Insertion sort* (3)

$$T(n) = c_1n + c_2(n-1) + c_3(n-1) + c_4 \sum_{j=2}^n t_j + c_5 \sum_{j=2}^n (t_j - 1) + c_6 \sum_{j=2}^n (t_j - 1) + c_7(n-1)$$

- Najgori slučaj (A je sortiran ali u obrnutom redosledu)

$$t_j = j$$

$$T(n) = c_1n + (c_2 + c_3 + c_7)(n-1) + c_4 \left( \frac{n(n+1)}{2} - 1 \right) + (c_5 + c_6) \left( \frac{n(n-1)}{2} \right)$$

$$T(n) = \frac{c_4 + c_5 + c_6}{2} n^2 + \left( c_1 + c_2 + c_3 + \frac{c_4 - c_5 - c_6}{2} + c_7 \right) n - (c_2 + c_3 + c_4 + c_7)$$

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

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

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



# Vreme izvršavanja *Insertion sort* (4)

- Najgori slučaj  $O(n^2)$
- Najbolji slučaj  $\Omega(n)$  – ovo ne možemo očekivati
- Prosek: svaki elemenat koji se umeće neka je manji od polovine do tada umetnutih, tj.  $(i - 1)/2$ . Potrebna je  $\frac{1}{2}$  ukupnog broja poređenja ( $\frac{1}{2}$  je konstantan faktor koji se zanemaruje u  $\frac{1}{2} n^2$ ), ali je  $\Theta(n^2)$
- Međutim, ukoliko je početni niz „skoro“ sortiran, tj. pretpostavimo da je svaki elemenat udaljen od svog konačnog mesta za oko  $k$  mesta, tada je potrebno  $kn$  pomeranja elemenata, tj. složenost je  $\Theta(n)$

# Još neke osobine algoritma

Izvršavanje u mestu

- *Insertion sort*, kao i *Selection sort*, vrši pomeranje vrednosti u zadatom (originalnom) nizu.
- Pri tome ne zahteva dodatne nizove
- Stoga ima osobinu izvršavanja „u mestu“

Stabilan algoritam

- Posmatraju se pozicije jednakih ključeva pre i nakon sortiranja i ako je njihov međusobni redosled (ne pozicija) nepromenjen sortiranjem onda je algoritam stabilan.
- *Insertion sort*, kao i *Selection sort*, su stabilni algoritmi.

# *Merge sort – sortiranje objedinjavanjem*

- Ovaj algoritam se razlikuje od *Selection* i *Insertion sort*-a:
  - Njegovo vreme izvršavanja je  $\Theta(n \log_2 n)$ , što je znatno brže kada se gledaju najgori slučajevi druga dva algoritma  $O(n^2)$
  - Konstantan faktor u asimptotskoj notaciji je veći nego kod drugih algoritama – sporiji je za malo  $n$
  - Ne radi „u mestu“. Ne može da pomera elemente u nizu A, nego radi sa kopijama niza.
- *Merge sort* primenjuje algoritamsku paradigmu „podeli i osvoji“

# Podeli i osvoji (*Divide-and-Conquer*)

- Ideja: Zadatak se deli na podzadatke koji su slični originalnom zadatku. Podzadaci se rešavaju rekurzijom i njihova rešenja se kombinuju da bi se rešio originalan zadatak.
- Koraci:
  1. **Podeli** – zadatak se deli na manje zadatke koji su slični originalu
  2. **Osvoji** – manji zadaci se rešavaju rekurzivno. Kada je sasvim mali, zadatak je trivijalan (zove se *base case*)
  3. **Kombinuj** – rešenja podeljenih zadataka se objedinjuju da bi se dobilo rešenje originalnog zadatka

# Primena *Divide-and-Conquer* na *Merge sort*

- 1. Podeli** – deli niz od  $n$  elemenata na dva jednaka podniza sa  $n/2$  elemenata svaki
  - Prvi podniz  $A[p..q]$ ,
  - Drugi podniz  $A[q + 1..r]$ ,  $p \leq q < r$
- 2. Osvoji** – svaki od delova se nezavisno sortira upotrebom *Merge sort*-a.
  - Pri tome se primenjuje rekurzija tako da se jedna polovina niza dalje deli i tako sve dok se ne dobije podniz sa jednim elementom.
- 3. Kombinuj** – dva sortirana podniza se objedinjuju u jedan sortirani niz.
  - Podnizovi  $A[p..q]$  i  $A[q + 1..r]$  se kombinuju (*merge*) u  $A[p..q]$

# Merge sort – Algoritam (1)

MERGE-SORT(*A*)

1 MERGE-SORT-STEP(*A*, 1, *A.Length*)

MERGE-SORT-STEP(*A*, *p*, *r*)

1 **if** *p* < *r*

2      $q = \lfloor (p + r) / 2 \rfloor$

3     MERGE-SORT-STEP(*A*, *p*, *q*)

4     MERGE-SORT-STEP(*A*, *q*+1, *r*)

5     MERGE(*A*, *p*, *q*, *r*)

...

```
MERGE(A, p, q, r)
1   $n_1 = q - p + 1$ 
2   $n_2 = r - q$ 
3  for  $i = 1$  to  $n_1$ 
4       $L[i] = A[p + i - 1]$ 
5  for  $j = 1$  to  $n_2$ 
6       $R[j] = A[q + j]$ 
7   $L[n_1 + 1] = \infty$ 
8   $R[n_2 + 1] = \infty$ 
9   $i = 1$ 
10  $j = 1$ 
11 for  $k = p$  to  $r$ 
12     if  $L[i] \leq R[j]$ 
13          $A[k] = L[i]$ 
14          $i = i + 1$ 
15     else
16          $A[k] = R[j]$ 
17          $j = j + 1$ 
```

```
// # elem. u levom podnizu
// # elem. u desnom podnizu
// kopiraj levi

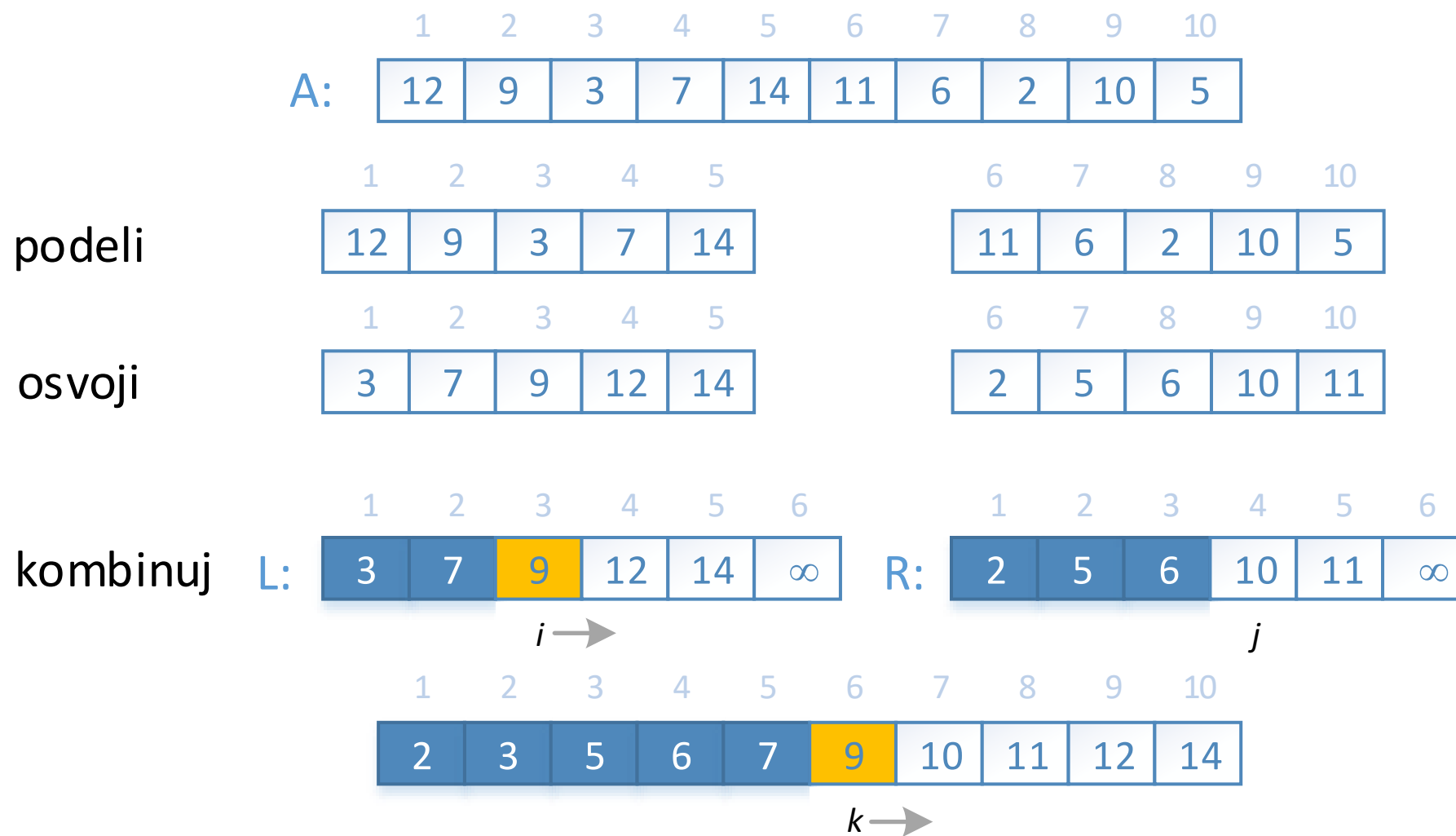
// kopiraj desni

// dodaj  $\infty$  da bude  $> R[n_2]$ 
// dodaj  $\infty$  da bude  $> L[n_1]$ 
// indeks u levom
// indeks u desnom podnizu
// “spoji” levi i desni

// kopiraj levi jer je manji
// pomeri se u levom podnizu

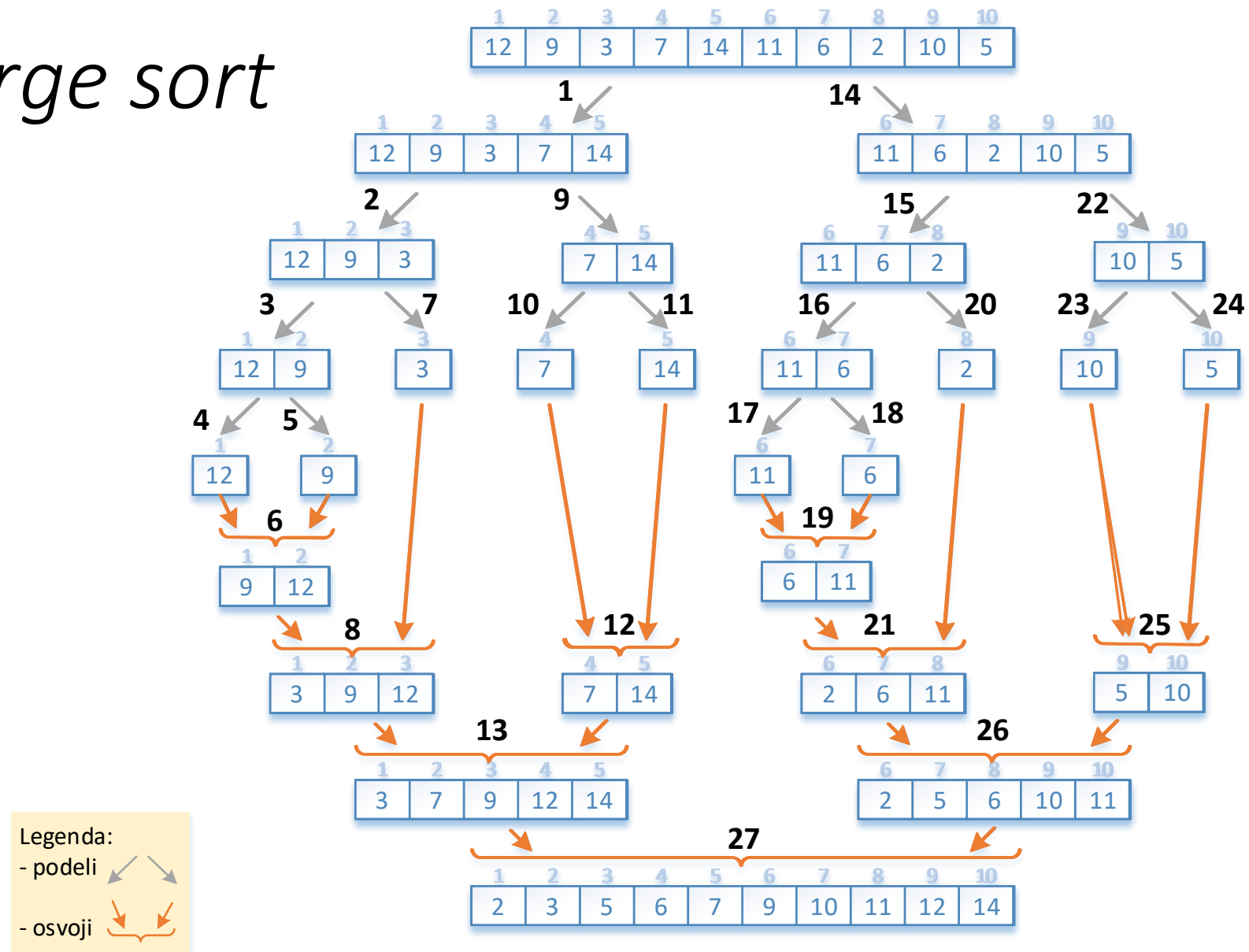
// kopiraj desni jer je manji
// pomeri se u desnom podnizu
```

# Primer *Merge sort*





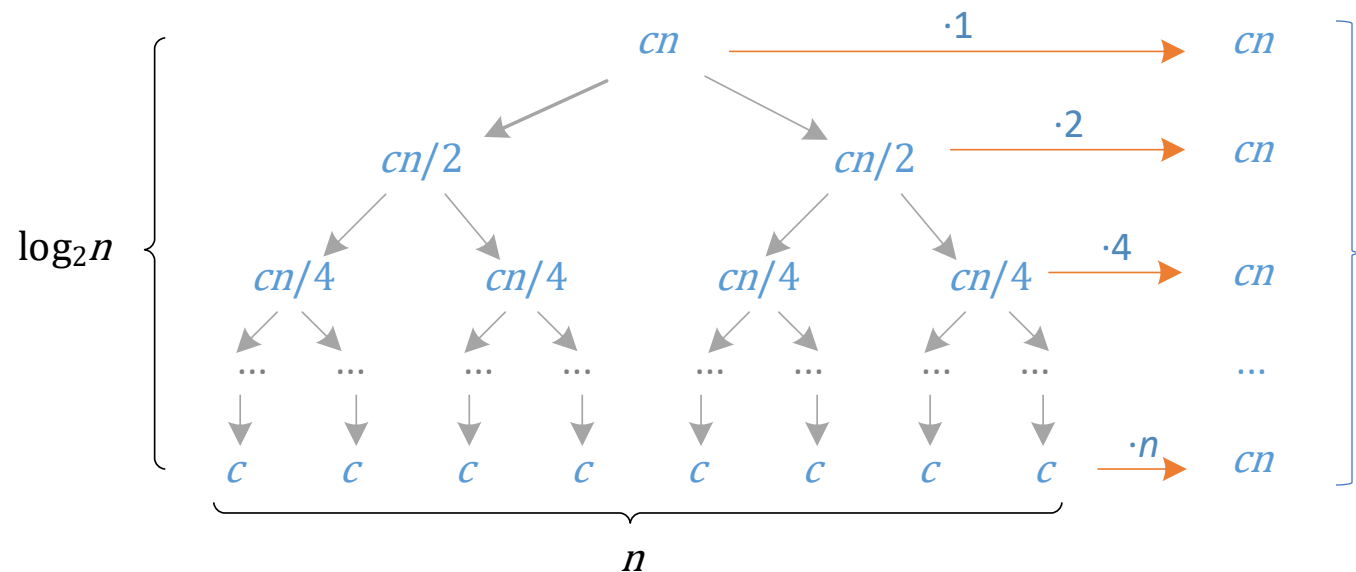
# Primer *Merge sort*



# Vreme izvršavanja *Merge sort*

- Podeli  $\Theta(1)$
- Osvoji  $2T(n/2)$
- Kombinuj  $\Theta(n)$

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

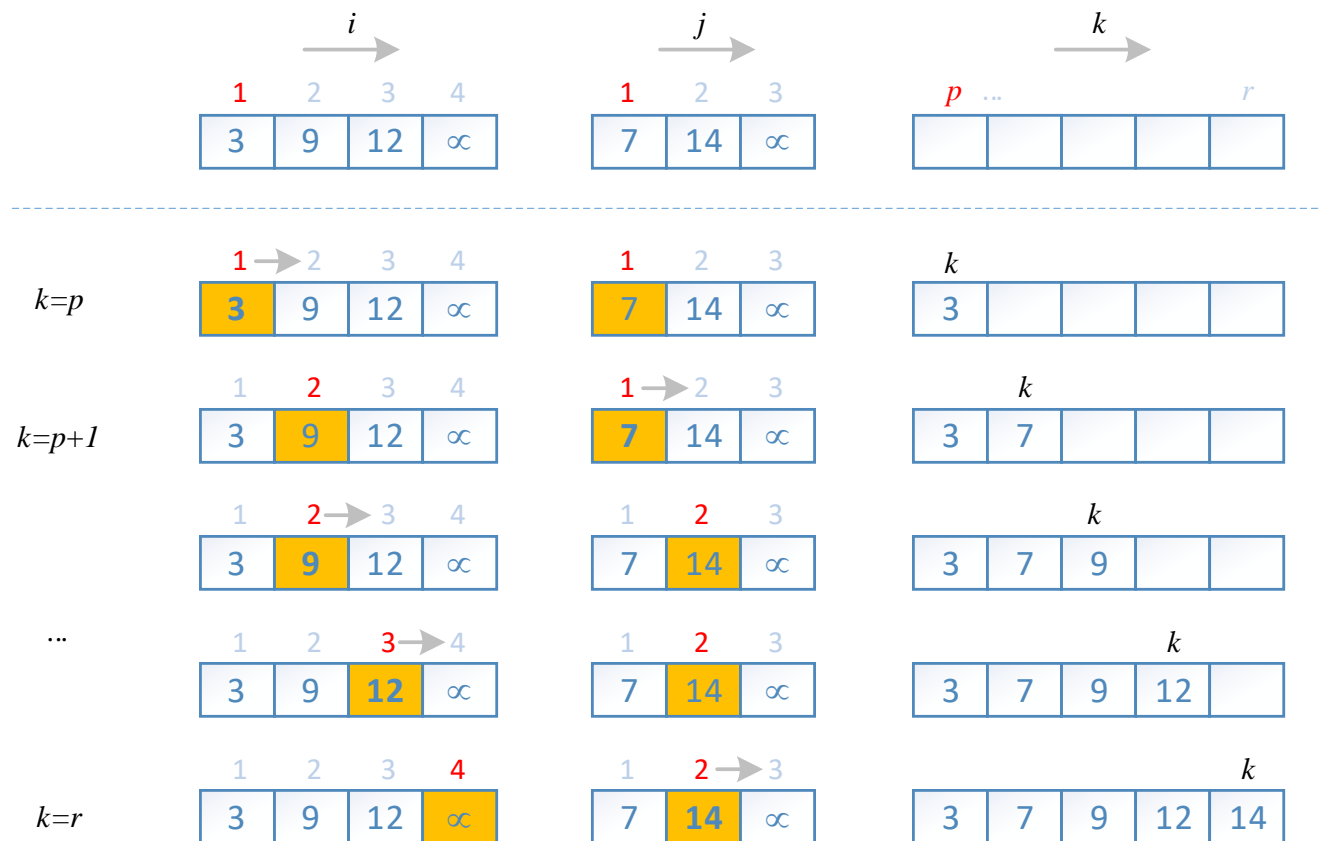


$$\text{Suma} = cn \log_2 n$$

$$T(n) = \Theta(n \log_2 n)$$

# Korak Merge u *Merge sort*

- Linearna složenost



OBJEDINI( $A, p, q, r$ )

```

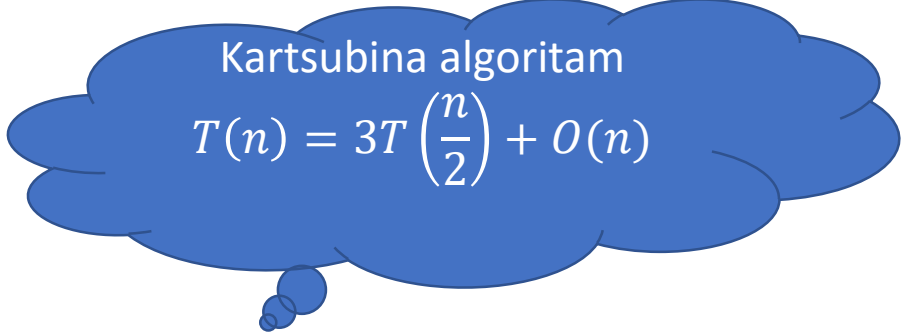
1   $n_1 = q - p + 1$ 
2   $n_2 = r - q$ 
3  for  $i = 1$  to  $n_1$ 
4       $L[i] = A[p + i - 1]$ 
5  for  $j = 1$  to  $n_2$ 
6       $R[j] = A[q + j]$ 
7   $L[n_1 + 1] = \infty$ 
8   $R[n_2 + 1] = \infty$ 
9   $i = 1$ 
10  $j = 1$ 
11 for  $k = p$  to  $r$ 
12     if  $L[i] \leq R[j]$ 
13          $A[k] = L[i]$ 
14          $i = i + 1$ 
15     else
16          $A[k] = R[j]$ 
17          $j = j + 1$ 

```

# Master teorema (kod rekurzivnih algoritama)

- Ako je

$$T(n) = aT\left(\frac{n}{b}\right) + O(n^d)$$



Karatsuba algoritam  
 $T(n) = 3T\left(\frac{n}{2}\right) + O(n)$

- $n$  – veličina problema
- $a$  – broj podproblema u rekurziiji
- $\frac{n}{b}$  – veličina svakog podproblema (pret. ako su iste veličine)
- $f(n) = O(n^d)$  – kompleksnost operacija van rekurziije

- Tada je

$$T(n) = \begin{cases} O(n^d) & d > \log_b a \\ O(n^d \log n) & d = \log_b a \\ O(n^{\log_b a}) & d < \log_b a \end{cases}$$

# *Quicksort* – sortiranje razdvajanjem

- Quicksort (takođe) primenjuje algoritamsku paradigmu „podeli i osvoji“
- Radi u mestu
- Vreme izvršavanja je u najgorem slučaju  $O(n^2)$ , ali je u prosečnom slučaju  $\Theta(n \log_2 n)$
- Konstantan faktor u asimptotskoj notaciji je manji nego kod *Merge sort* algoritama
- Praktično se često koristi!

# Primena *Divide-and-Conquer* na *Quicksort*

- 1. Podeli** – deli niz  $A[p..r]$  na dva podniza  $A[p..q-1]$  i  $A[q+1..r]$  tako da su u prvom elementi manji od  $A[q]$ , a u drugom veći od  $A[q]$ ,  $p \leq q < r$ 
  - $A[q]$  se naziva **pivot** i njegova vrednost se uzima npr. sa kraja niza  $A$
- 2. Osvoji** – svaki od delova se nezavisno sortira rekurzivnim pozivima *Quicksort*-a.
- 3. Kombinuj** – ne treba ništa raditi jer je niz  $A$  sortiran

# Quicksort – Algoritam (1)

QUICKSORT(*A*)

1 QUICKSORT-STEP(*A*, 1, *A.Length*)

QUICKSORT-STEP(*A*, *p*, *r*)

1 **if** *p* < *r*

2     *q* = PARTITION(*A*, *p*, *r*)

3     QUICKSORT-STEP(*A*, *p*, *q*-1)

4     QUICKSORT-STEP(*A*, *q*+1, *r*)

...

# Quicksort – Algoritam (2)

PARTITION(A,  $p$ ,  $r$ )

1  $x = A[r]$  // pivot

2  $i = p - 1$

3 **for**  $j = p$  **to**  $r-1$

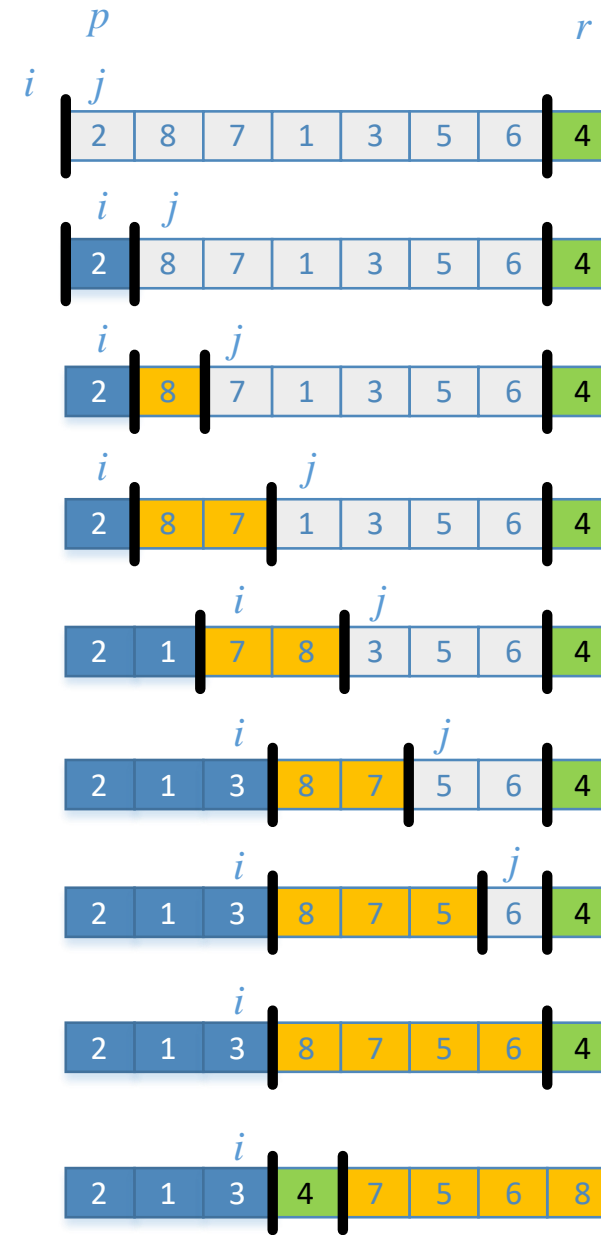
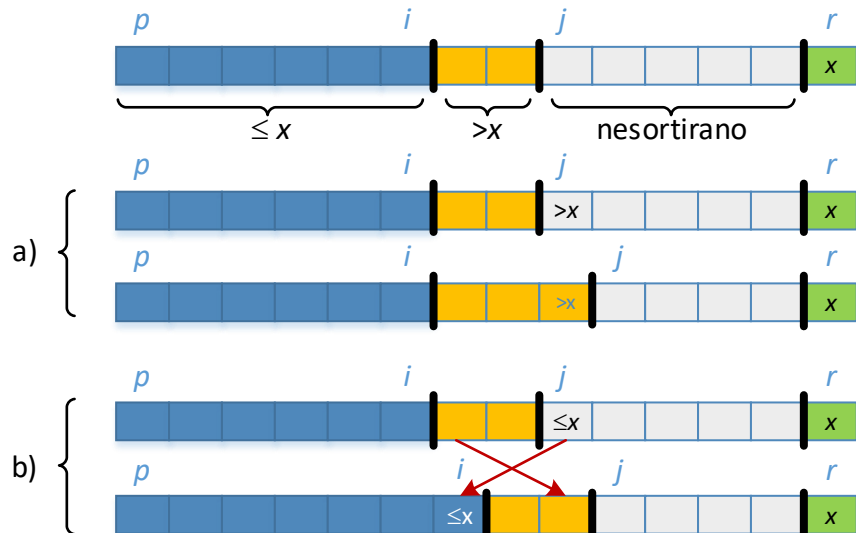
4     **if**  $A[j] \leq x$

5          $i = i + 1$

6          $A[i] \leftrightarrow A[j]$

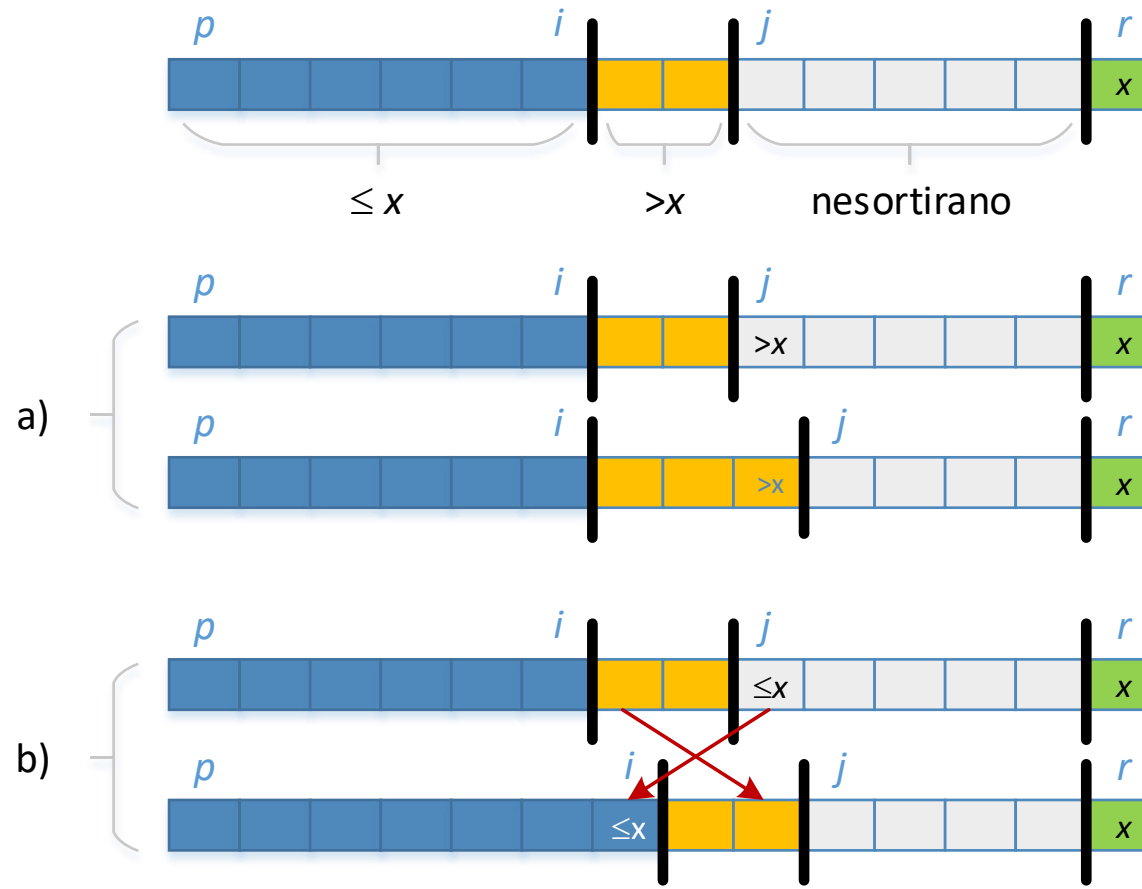
3  $A[i+1] \leftrightarrow A[r]$

4 **return**  $i+1$  // pozicija pivota





# Quicksort – Algoritam (3)



# Quicksort - Primer

$p$										$r$
1	2	3	4	5	6	7	8	9	10	
12	9	3	7	14	11	6	2	10	5	

$p$		$q$									$r$
1	2	3	4	5	6	7	8	9	10		
3	2	5	7	14	11	6	9	10	12		

$p, q$		$r$
1	2	
2	3	

$p$						$q$	$r$	
4	5	6	7	8	9	10		
7	11	6	9	10	12	14		

$p$		$q$		$r$
4	5	6	7	8
7	6	9	10	11

$p$		$q, r$	
4	5	6	
7	6	9	

$p, q$		$r$
4	5	
6	7	

# Vreme izvršavanja *Quicksort*-a

- Vreme izvršavanja zavisi od odnosa vrednosti u nizu
  - Ako je podela (*Partitioning*) balansirana vreme je  $\Theta(n \log_2 n)$
  - Kod nebalansirane podele vreme je  $\Theta(n^2)$
- Nebalansirana podela je najgori slučaj: od  $n$  elemenata proizvodi podgrupe sa  $n-1$  i 0 elemenata

- Npr. svi elementi su manji od pivota
- Tada je

$$T(n) = T(n-1) + T(0) + \Theta(n)$$

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

- Najbolji slučaj: od  $n$  elemenata se proizvode 2 podgrupe sa  $n/2$  elemenata

- Tada je

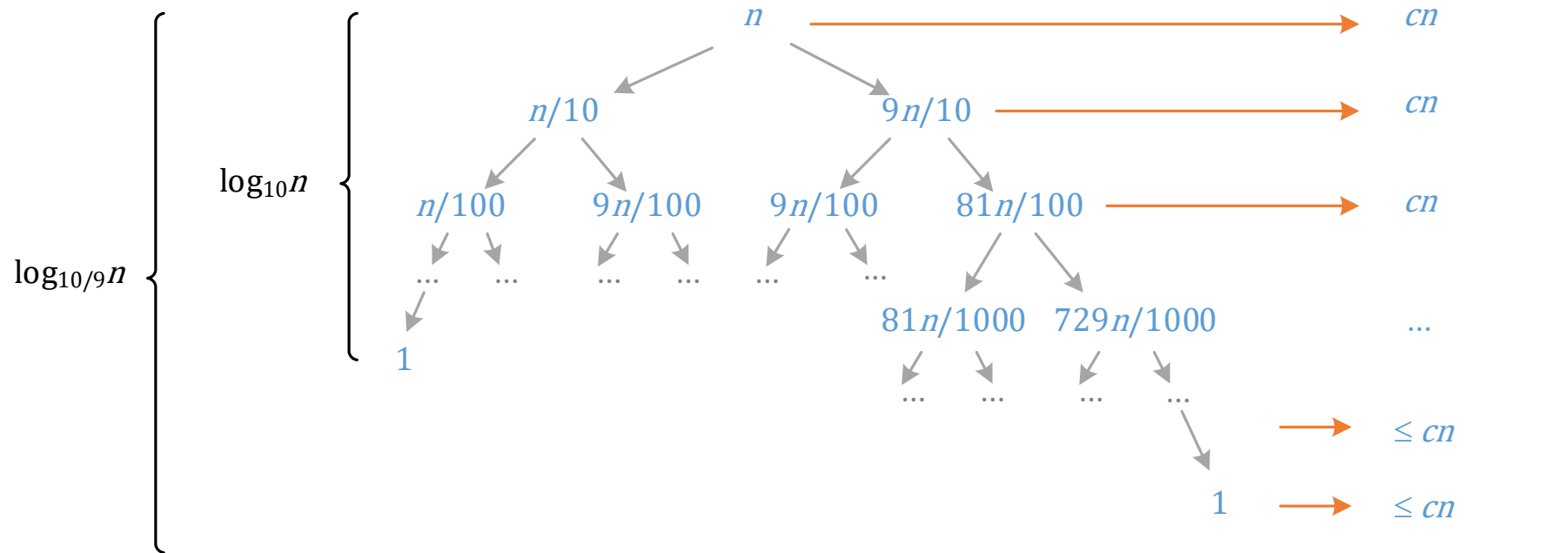
$$T(n) = 2T(n/2) + \Theta(n)$$

$$T(n) = \Theta(n \log_2 n)$$

...

# Vreme izvršavanja *Quicksort*-a (2)

- Balansirano partitionisanje
  - Prosečan slučaj je mnogo bliži najboljem nego najgorem slučaju
- Npr. primer podele 1:9 daje  $T(n) = T(9n/10) + T(n/10) + \Theta(n)$



$$\log_{\frac{10}{9}} n = \frac{\log_2 n}{\log_2 \frac{10}{9}} = c \log_2 n$$

$$O(n \log_2 n)$$

# Vreme izvršavanja *Quicksort*-a (3)

- Poželjno je izbeći nebalansirane podele
  - Npr. ako je na početku niz sortiran u opadajućem redosledu
- Poboljšanje rešenja: ne uzimati uvek poslednji element kao pivot
  - Na početku Partition procedure zameniti  $A[r]$  sa slučajno izabranim elementom iz  $A[p..r]$ , čime je pivot slučajno odabran.
- Ideja: slučajno izabrati tri elementa i onaj koji ima srednju vrednost postaviti za pivota.

# Zaključak

- Algoritmi pretrage

Algoritam	Najgori slučaj	Najbolji slučaj	Zahteva sortiran niz?
Linearna pretraga	$\Theta(n)$	$\Theta(1)$	Ne
Binarna pretraga	$\Theta(\log_2 n)$	$\Theta(1)$	Da

- Algoritmi sortiranja

Algoritam	Najgori slučaj	Najbolji slučaj	Broj zamena (najgori slučaj)	Radi u mestu?
Selection sort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n)$	Da
Insertion sort	$\Theta(n^2)$	$\Theta(n)$	$\Theta(n^2)$	Da
Merge sort	$\Theta(n \log_2 n)$	$\Theta(n \log_2 n)$	$\Theta(n \log_2 n)$	Ne
Quicksort	$\Theta(n^2)$	$\Theta(n \log_2 n)$	$\Theta(n^2)$	Da