Pretraga i sortiranje

Predmet: Uvod u Algoritme 17 - ESI053
Studijski program: Primenjeno softversko inženjerstvo

DEPARTMAN ZA RAČUNARSTVO I AUTOMATIKU
DEPARTMAN ZA ENERGETIKU, ELEKTRONIKU I KOMUNIKACIJE

Problem pretrage

- Ulazi:
  - niz vrednosti, npr. brojeva A = $[a_1, a_2, \ldots., a_n]$
  - vrednost koja se traži (ključ) *key*
- Izlaz:
  - Indeks (pozicija) tražene vrednosti
    (u posebnom slučaju tražene vrednosti nema u nizu)
- Napomena:
  - Ulazni niz nema određeni poredak (nije sortiran)

Primer:

Ulaz:	**8**	**4**	**2**	**9**	**3**	**6**
	(1)	(2)	(3)	(4)	(5)	(6)

Traži se vrednost **3**: izlaz je 5

Vrednosti mogu biti celi brojevi, brojevi u pokretnom zarezu, tekstovi, strukture (složeni tipovi) itd.

Linearna pretraga - algoritam

Naziva se i sekvencijalna pretraga – pronalazi samo prvu pojavu

```
LINEARNO-PRETRAŽI(A, key)
  for k=1 to A.length
      if A[k] == key
          return k
  return nije-nađen
```

Iteracije

A:	8	4	2	9	3	6
k:	(1)	(2)	(3)	(4)	(5)	(6)

key:3

A[*k*]==*key* ?

Suština trajanja pretrage je broj ovih pitanja.

- Koliko traje algoritam kada je ključ na prvoj poziciji?
- Koliko traje algoritam kada je ključ na poslednjoj poziciji?
- Da li smo morali da krenemo sa poređenjima od prvog elemementa?
- Da li je pretraga brža ako krenemo sa drugog kraja?
- Da li je bitan redosled poređenja?
- Da li se broj „pitanja" menja ako krenemo od sredine?
- Da li moramo sve pozicije da proverimo?

Linearna pretraga – pronađi sve

```
LINEARNO-PRETRAŽI-SVE(A, key)
rez = []          // indeksi-pronađenih
j = 0
for k=1 to A.length
    if A[k] == key
        j = j + 1
        rez[j] = k
return rez
```

- Koliko traje algoritam kada je ključ na prvoj poziciji?
- Koliko traje algoritam kada je ključ na poslednjoj poziciji?
- Da li smo morali da krenemo sa poređenjima od prvog elementa?
- Da li je pretraga brža ako krenemo sa drugog kraja?
- Da li je bitan redosled poređenja?
- Da li se broj „pitanja“ menja ako krenemo od sredine?
- Da li moramo sve pozicije da proverimo?

A:	**8**	**3**	**2**	**9**	**3**	**3**
k:	(1)	(2)	(3)	(4)	(5)	(6)

key: **3**

Izlaz: [2, 5, 6]

Linearna pretraga – algoritam sa „stražarem“

Ideja ubacivanja „stražara“ (engl. *sentinel*) je da se deo koda koji se ponavlja oslobodi potrebe za proverom kraja niza.

„Stražar“ je ovde vrednost ključa koja se ubacuje na kraj niza.

```
LINEANO-PRETRAŽI-SA-STRAŽAREM(A, key)
n = A.length
last = A[n]
A[n] = key
k = 1
while A[k] ≠ key
  k = k + 1
A[n] = last
if k < n or A[n] == key
  return k
else
  return nije-nađen
```

Asimptotska notacija

- Približno procenjujemo vreme izvršavanja algoritma u zavisnosti od veličine ulaza n (i to kada je n veliko).

- Θ-notacija – opisuje vreme izvršavanja algoritma. Odnosi se na obe granice trajanja algoritma (najkraće i najduže trajanje)
- O-notacija – koristi se da opiše najgore slučajeve
- Ω-notacija – koristi se da opiše najbolje slučajeve

- Ako je vreme izvršavanja aloritma $f(n)$ ono se opisuje … (naredni slajdovi)

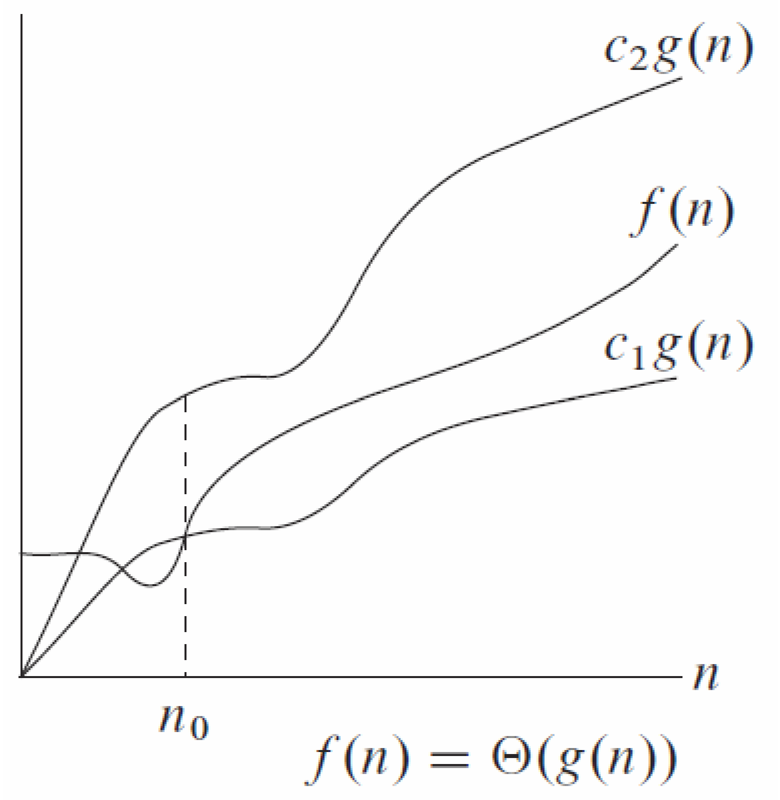
Θ-notacija (teta-notacija)

- za datu funkciju $g(n)$ sa $\Theta(g(n))$ se označava skup funkcija za koje važi

$$\Theta(g(n)) = \{f(n): \forall n \geq n_0, \exists c_1 > 0, \exists c_2 > 0 \Rightarrow 0 \leq c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)\}$$

- Ispravno je reći da je vreme izvršavanja $f(n) \in \Theta(g(n))$ mada se piše $f(n) = \Theta\big(g(n)\big)$ (ovo zbunjuje ☹)

- Θ-notacija asimptotski ograničava funkciju sa obe strane.



Primer Θ-notacije

Za neki algoritam važi da je ...

- Stvarno vreme izvršavanja je $f(n) = 100n^2 + 350n + 250$
- Za veliko n je $T(n) = \Theta(n^2)$

  što znači da je $g(n) = n^2$

  pa je $c_1 n^2 \leq f(n) \leq c_2 n^2$

  Primer:

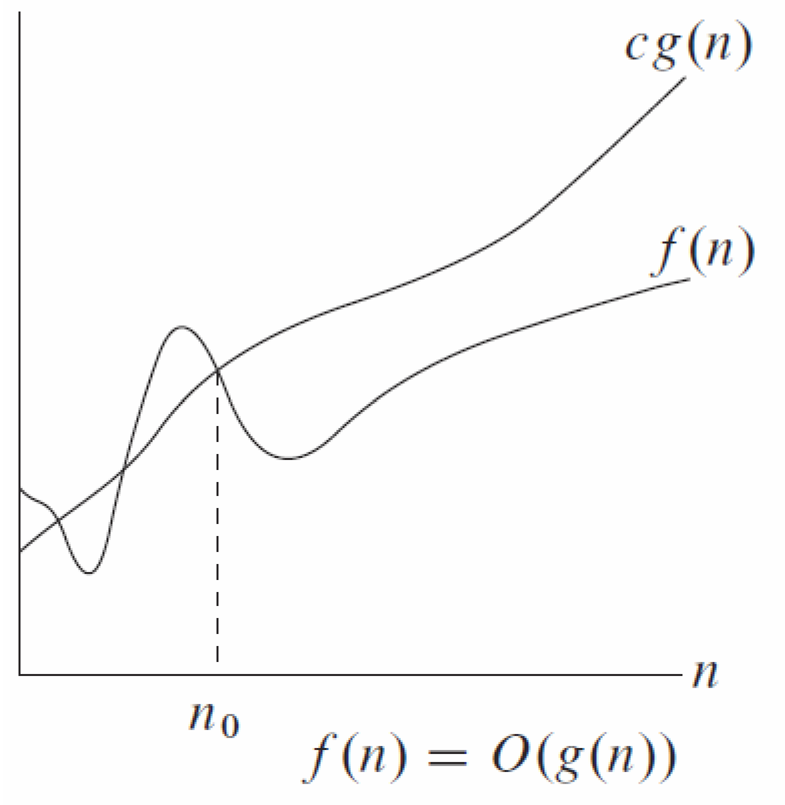
$$c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$$
$$c_1 \cdot n^2 \leq 100n^2 + 350n + 250 \leq c_2 \cdot n^2$$
$$c_1 \leq 100 + \frac{350}{n} + \frac{250}{n^2} \leq c_2$$
$$n_0 = 5$$
$$c_1 = 100$$
$$c_2 = 100 + \frac{350}{5} + \frac{250}{5^2} = 180$$

O-notacija („veliko o"-notacija)

- za datu funkciju $g(n)$ sa $O(g(n))$ se označava skup funkcija za koje važi

$$O(g(n)) = \{f(n) \colon \forall n \geq n_0, \exists c > 0, \Rightarrow 0 \leq f(n) \leq c \cdot g(n)\}$$

- *O*-notacija asimptotski ograničava funkciju **sa gornje strane**.
  Odnosi se na najgori slučaj.

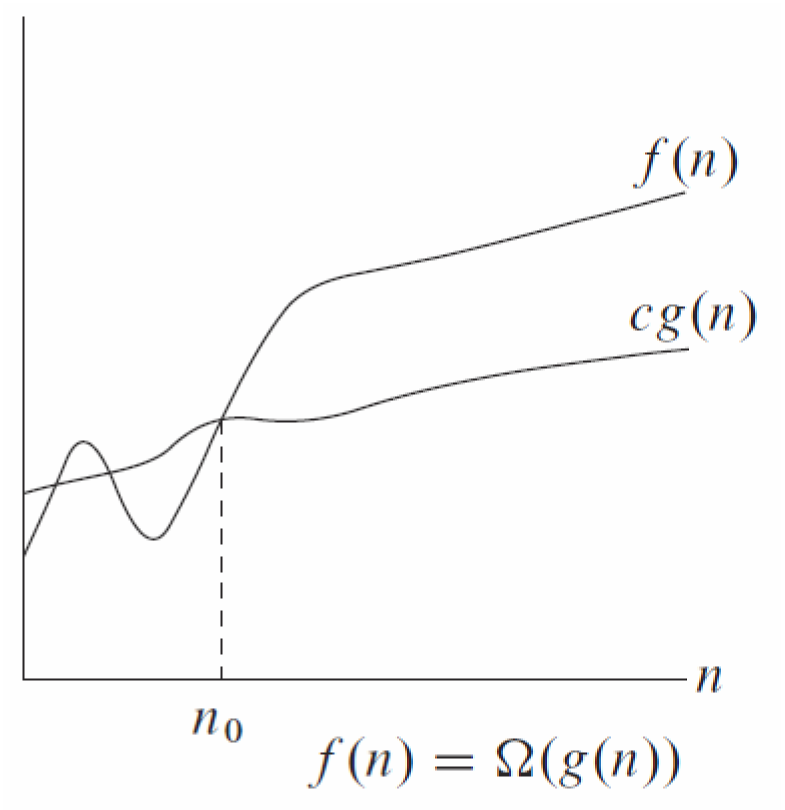


Ω-notacija („veliko omega“-notacija)

- za datu funkciju $g(n)$ sa $\Omega(g(n))$ se označava skup funkcija za koje važi

  $$\Omega(g(n)) = \{f(n)\colon \forall n \geq n_0, \exists c > 0, \Rightarrow 0 \leq c \cdot g(n) \leq f(n)\}$$

- Ω-notacija asimptotski ograničava funkciju **sa donje strane**.
  Označava najbolji slučaj.



Primer asimptotskih notacija

Za algoritam sortiranja umetanjem ...

- Najgore vreme je funkcija od n^2, te je vreme izvršavanja $O(n^2)$
- Najbolje vreme je funkcija od n, te je vreme izvršavanja $\Omega(n)$

(naravno, sve ovo za veliko n)

Analiza: Linearna pretraga – pronađi sve

LINEARNO-PRETRAŽI-SVE(A, *key*)	TRAJANJE	BROJ PROLAZA
rez = []	c_1	1
j = 0	c_2	1
for *k*=1 **to** A.*length*	c_3	$n+1$
if A[*k*] == *key*	c_4	n
j = *j* + 1	c_5	$0..n$
rez[*j*] = *k*	c_6	$0..n$
return *rez*	c_7	1

- Najgori slučaj:

$$T(n) = c_1 + c_2 + c_3(n+1) + c_4 n + c_5 n + c_6 n + c_7$$
$$T(n) = (c_3 + c_4 + c_5 + c_6)n + c_1 + c_2 + c_3 + c_7 = O(n)$$

- Najbolji slučaj:

$$T(n) = c_1 + c_2 + c_3(n+1) + c_4 n + c_7$$
$$T(n) = (c_3 + c_4)n + c_1 + c_2 + c_3 + c_7 = \Omega(n)$$

- Zaključak: vreme izvršavanja algoritma je $\Theta(n)$.

Analiza: Linearna pretraga – pronađi prvog

LINEARNO-PRETRAŽI(A, *key*)	TRAJANJE	BROJ PROLAZA
for *k*=1 **to** A.*length*	c_1	1..*n*+1
if A[*k*] == *key*	c_2	1..*n*
return *k*	c_3	1
return *nije-nađen*	c_4	0..1

- Najgori slučaj:

$$T(n) = c_1(n+1) + c_2 n + c_4$$
$$T(n) = (c_1 + c_2)n + c_1 + c_4 = O(n)$$

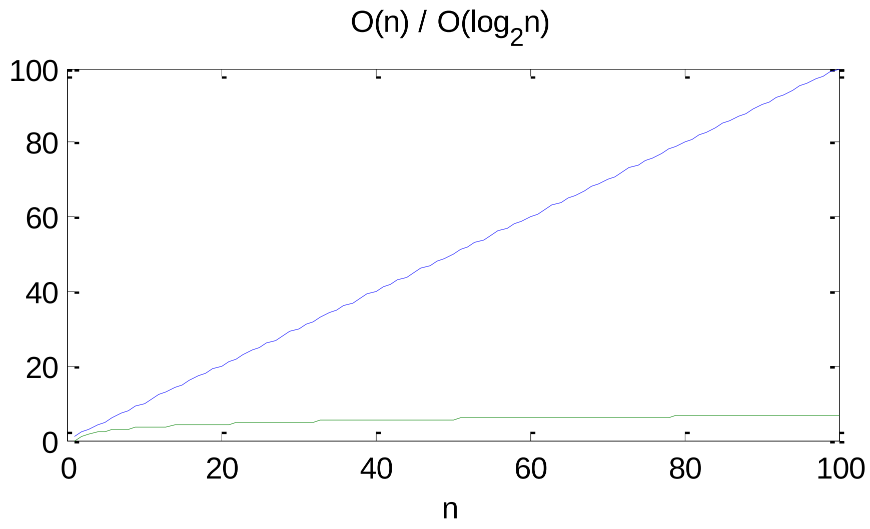
- Najbolji slučaj:

$$T(n) = c_1 + c_2 + c_3 = \Omega(1)$$

- Zaključak: vreme izvršavanja algoritma je $O(n)$.

Problem pretrage – unapređenje?

- Složenost prikazanih algoritama pretrage je $O(n)$.
  Pri tome niz podataka nije sortiran.
- Bolje rešenje se dobija pretragom niza sortiranih podataka!
  - Ako su podaci brojevi onda se sortiraju u neopadajućem redosledu
  - Ako su podaci stringovi onda se sortiraju u leksikografskom poretku
  - ...
- Algoritam binarne pretrage ima vreme izvršavanja $O(\log_2 n)$



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)

Problem sortiranja

- Ulaz: sekvenca brojeva $[a_1, a_2, \ldots., a_n]$

- Izlaz: permutacija $[a'_1, a'_2, \ldots., a'_n]$
  tako da je $a'_1 \leq a'_2 \leq \ldots. \leq a'_n$

- Primer:
  - Ulaz: **8 4 2 9 3 6**
  - Izlaz: **2 3 4 6 8 9**

k_1	k_2	k_3	...	k_n
a_1	a_2	a_3	...	a_n
b_1	b_2	b_3	...	b_n
...	...	...	...	...

- Broj koji je u ulaznoj sekvenci se naziva **ključ** (*sort key*)
- Često se u ulaznoj sekvenci nalaze elementi, tj. podatke čini niz struktura sačinjenih od ključa i dodatnih (satelitskih) podataka (*satellite data*)

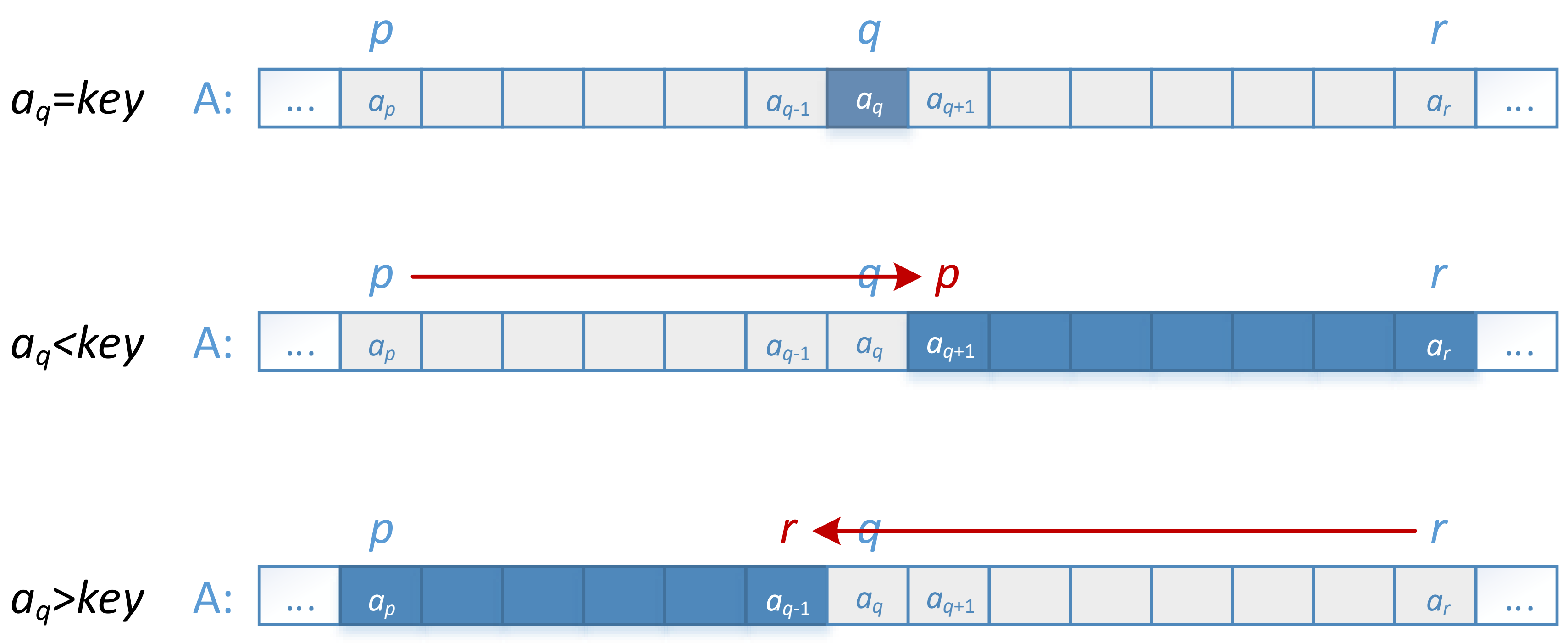
Binarna pretraga

n	log$_2$(*n*)
1.024	10
1.048.576	20
1.073.741.824	30

- Zahteva da je sortiran niz koji se pretražuje.
- Složenost 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 intresa i postupak se sprovodi iterativno u podnizovima koji se smanjuju (prepolovljavaju) sve dok ne ostane samo jedan elemenat u podnizu.

Binarna pretraga (2)

- U svakoj iteraciji se posmatra deo podataka od indeksa p do indeksa r
- Posmatra se sredina niza, tj. indeks $q = \left\lfloor \frac{p+r}{2} \right\rfloor$ (simbol $\lfloor \cdot \rfloor$ označava zaokruživanje na dole)



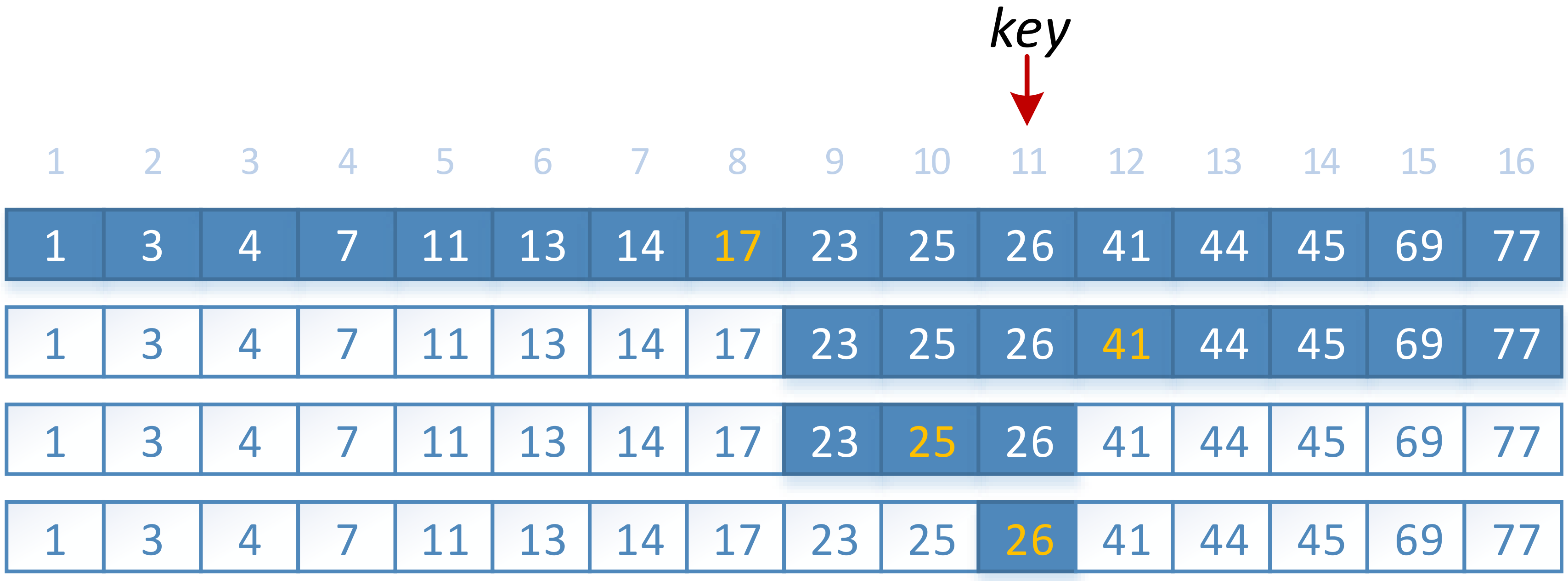
Binarna pretraga - algoritam

```
BINARNO-PRETRAŽI(A, key)
p = 1                       // leva granica
r = A.length                // desna granica
while p <= r
    q = ⌊(p + r)/2⌋         // sredina
    if A[q] == key
        return q            // pronađen, gotovo!
    elseif A[q] > key
        r = q - 1
    else
        p = q + 1
return nije-nađen
```

Binarna pretraga – rekurzivni algoritam

```
BINARNO-PRETRAŽI(A, key)
 REKURZIVNO-BINARNO-PRETRAŽI(A, 1, A.length, key)

REKURZIVNO-BINARNO-PRETRAŽI(A, p, r, key)
  if p > r
    return nije-nađen
  else
    q = ⌊(p + r)/2⌋
    if A[q] == key
      return q
    elseif A[q] > key
       REKURZIVNO-BINARNO-PRETRAŽI(A, p, q-1, key)
    else
       REKURZIVNO-BINARNO-PRETRAŽI(A, q+1, r, key)
```

Primer binarne pretrage

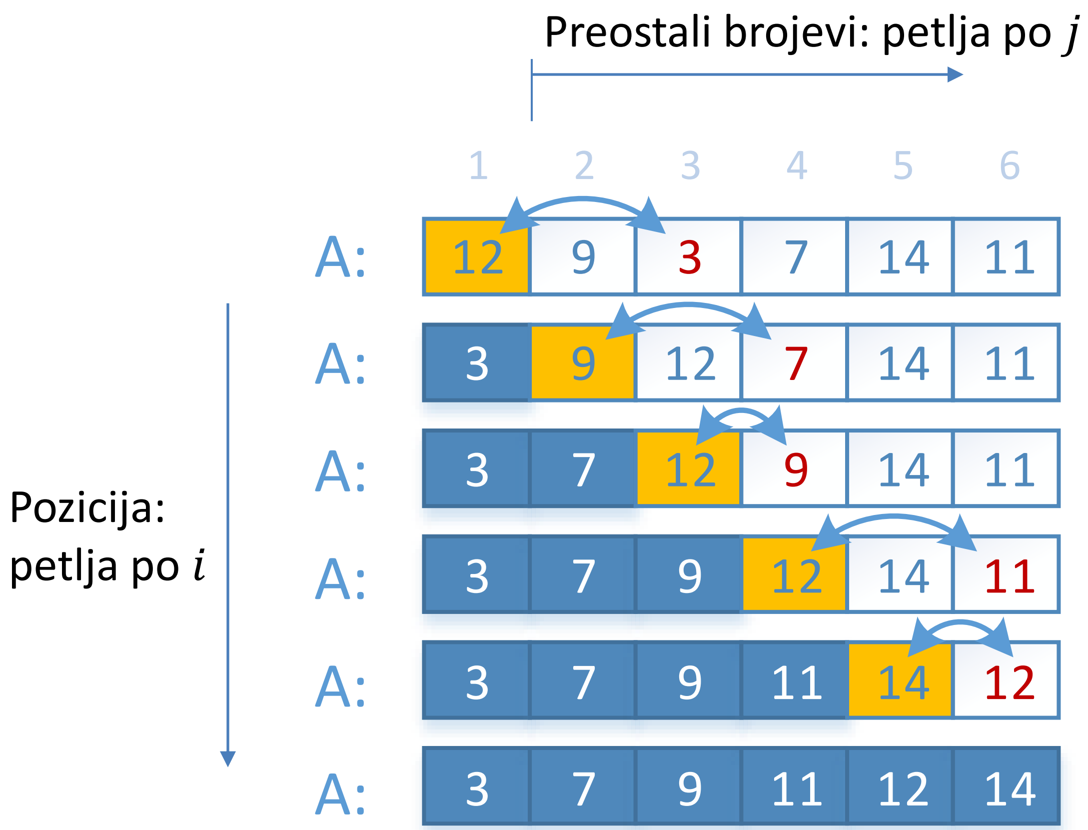


Sortiranje izborom (*Selection sort*)

```
SORTIRAJ-IZBOROM(A)
for i = 1 to A.length-1
   indMin = i
   for j = i+1 to A.length
      if A[j] < A[indMin]
         indMin = j
   A[i] ↔ A[indMin]        // zameni
```

Primer *Selection sort*

```
SORTIRAJ-IZBOROM(A)
  for i = 1 to A.length-1
      indMin = i
      for j = i+1 to A.length
          if A[j] < A[indMin]
              indMin = j
      A[i] ↔ A[indMin]
```



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

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

$$(n-1)+(n-2)+(n-3)+\ldots+2+1=\sum_{k=1}^{n-1}k$$

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

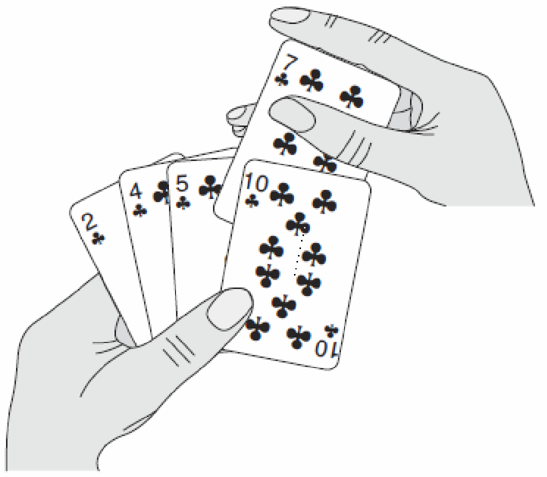
- Odakle vidimo da je složenost algoritma $\Theta(n^2)$
- Dodatna osobina:
  Maksimalan broj zamena elemenata u nizu je $n-1$, tj. $\Theta(n)$

  Primetiti da zamena vrednosti uključuje zapisivanje.

```
SORTIRAJ-IZBOROM(A)
for i = 1 to A.length-1
   indMin = i
   for j = i+1 to A.length
      if A[j] < A[indMin]
         indMin = j
   A[i] ↔ A[indMin]
```

Sortiranje umetanjem (Insertion sort)

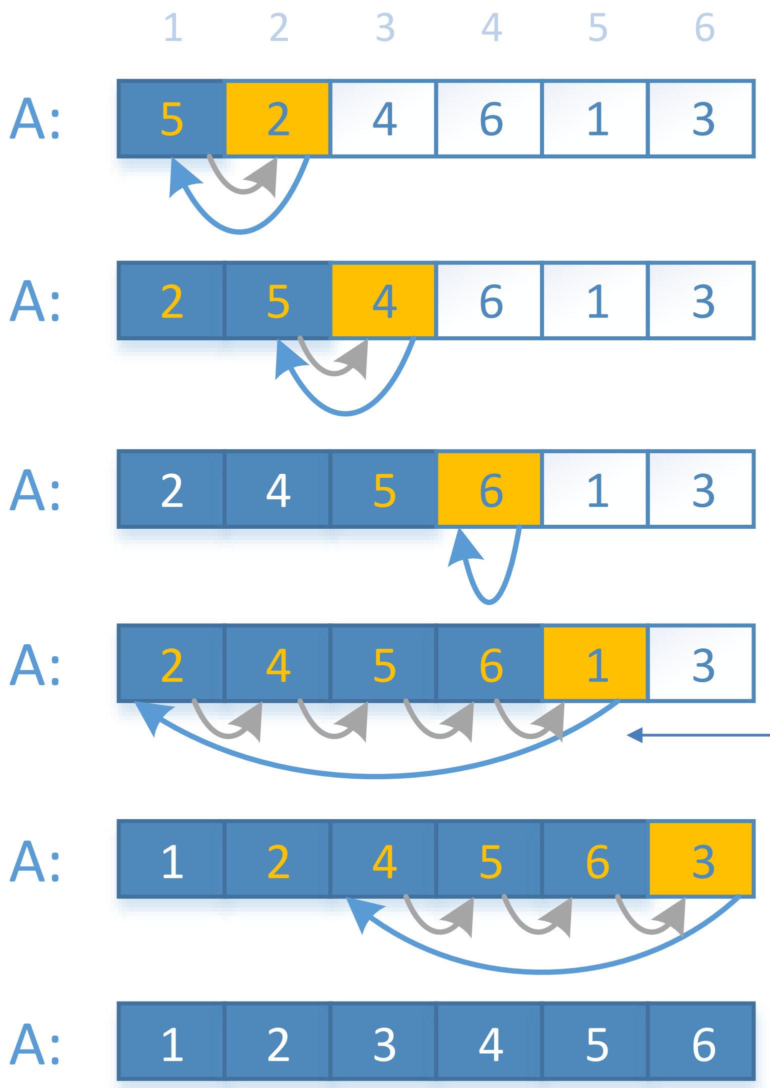
- Ideja algoritma je slična postupkuređanja karata u ruci.



```
SORTIRAJ-UMETANJEM(A)
for j = 2 to A.length
  key = A[j]
  i = j-1
  while i>0 & A[i]>key
    A[i+1] = A[i]
    i = i-1
  A[i+1] = key
```

Primer *Insertion sort*

Od 2. do poslednje pozicije: petlja po *j*




```
SORTIRAJ-UMETANJEM(A)
  for j = 2 to A.length
      key = A[j]
      i = j-1
      while i>0 & A[i]>key
          A[i+1] = A[i]
          i = i-1
      A[i+1] = key
```

Narandžasti je ključ i pomera se u levo dok mu se ne pronađe mesto

- najmanje 0 pomeranja
- najviše $j-1$ pomeranja

Vreme izvršavanja *Insertion sort*

SORTIRAJ-UMETANJEM(A)	TRAJANJE	BROJ PROLAZA
for j = 2 **to** A.*length*	c_1	n
key = A[j]	c_2	n-1
i = j-1	c_3	n-1
while i>0 & A[i]>*key*	c_4	Σt_j, j=2..n
A[i+1] = A[i]	c_5	Σt_j-1, j=2..n
i = i-1	c_6	Σt_j-1, j=2..n
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)

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

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

$$T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4(n-1) + c_7(n-1)$$
$$T(n) = (c_1 + c_2 + c_3 + c_4 + c_7)n - (c_2 + c_3 + c_4 + c_7)$$
$$T(n) = \Omega(n)$$

- Najgori slučaj (A je sortiran ali u obrnutom redosledu), $t_j = j$

$$T(n) = c_1 n + (c_2 + c_3 + c_7)(n-1) + c_4\left(\frac{n(n+1)}{2} - 1\right) + (c_5 + c_6)\left(\frac{n(n-1)}{2}\right)$$
$$T(n) = \frac{c_4 + c_5 + c_6}{2} n^2 + \left(c_1 + c_2 + c_3 + \frac{c_4 - c_5 - c_6}{2} + c_7\right) n - (c_2 + c_3 + c_4 + c_7)$$
$$T(n) = O(n^2)$$

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* (3)

- 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 ½ ukupnog broja poređenja (½ je konstantan faktor koji se zanemaruje u ½ n^2), pa 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.

Sortiranje objedinjavanjem - *Merge sort*

- 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 asimtotskoj 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

- Ovo je paradigma koja nalazi brojne primene u algoritmima

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

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

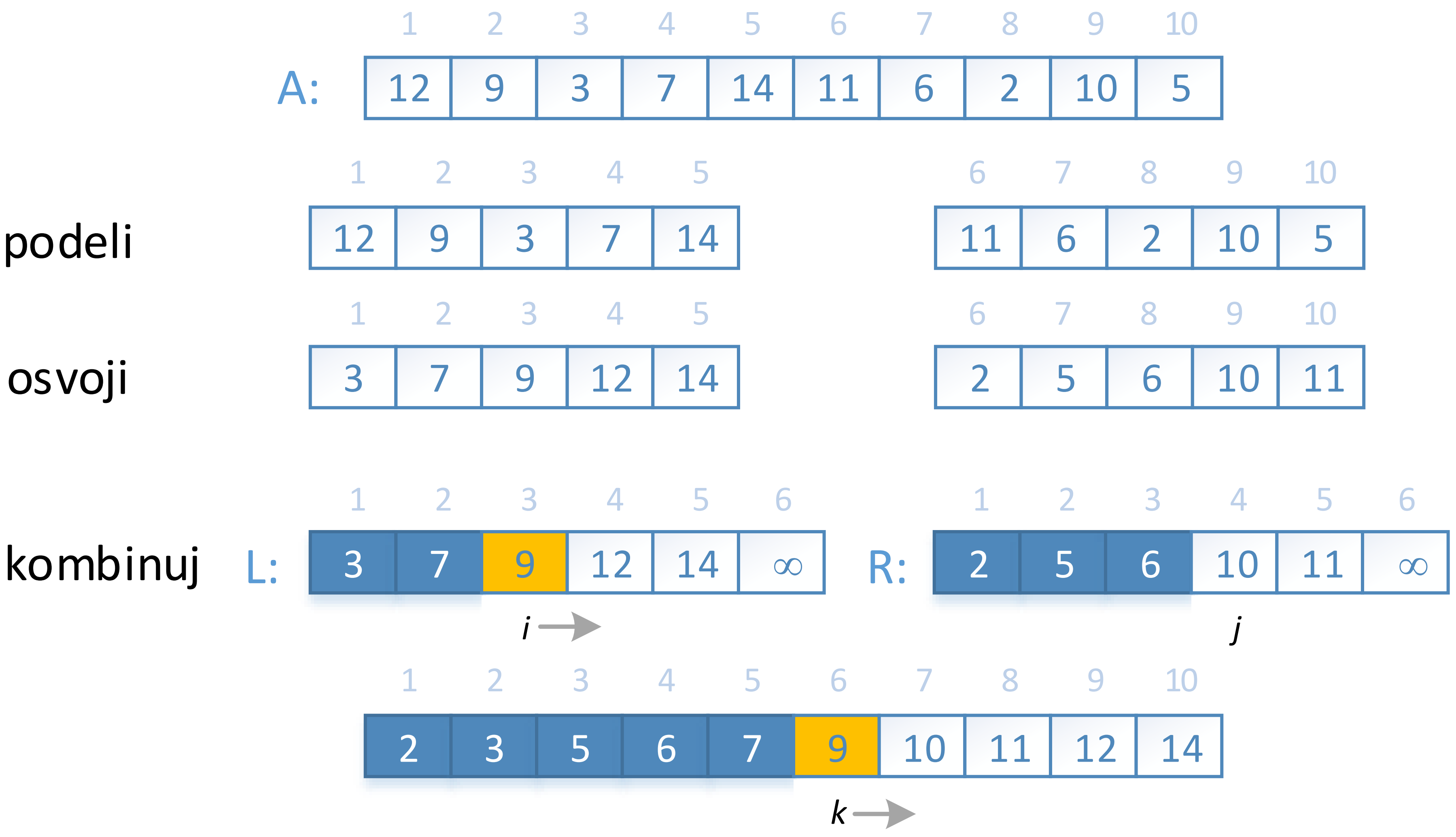
Merge sort – Algoritam (1)

```
SORTIRAJ-OBJEDINJAVANJEM(A)
  SORTIRAJ-OBJEDINJAVANJEM-KORAK(A, 1, A.length)

SORTIRAJ-OBJEDINJAVANJEM-KORAK(A, p, r)
  if p < r
     q = ⌊(p + r)/2⌋
     SORTIRAJ-OBJEDINJAVANJEM-KORAK(A, p, q)
     SORTIRAJ-OBJEDINJAVANJEM-KORAK(A, q+1, r)
     OBJEDINI(A, p, q, r)
```

…

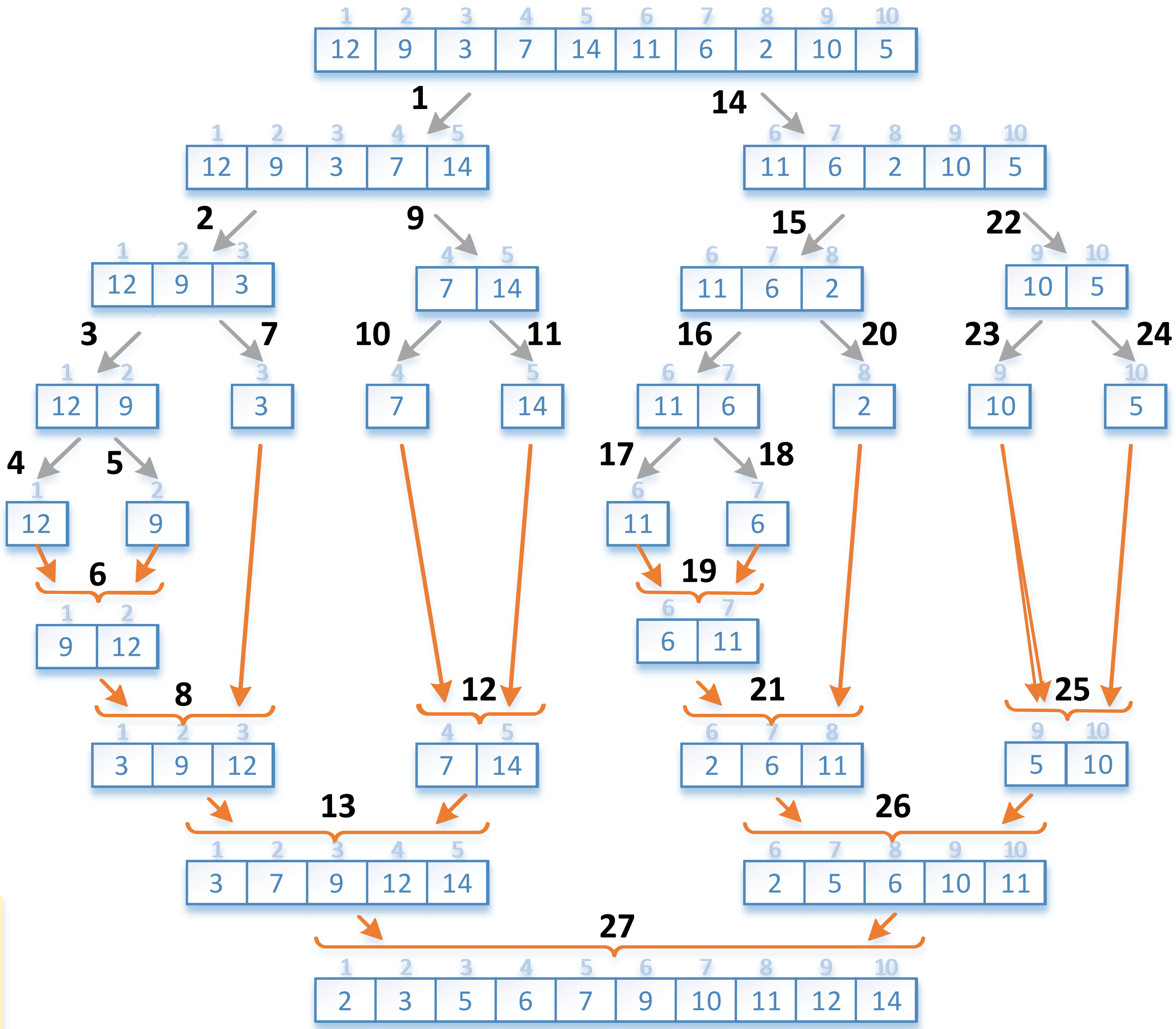
```
OBJEDINI(A, p, q, r)
  n₁ = q - p + 1                // # elem. u levom podnizu
  n₂ = r - q                    // # elem. u desnom podnizu
  for i = 1 to n₁               // kopiraj levi
      L[i] = A[p + i - 1]
  for j = 1 to n₂               // kopiraj desni
      R[j] = A[q + j]
  L[n₁ + 1] = ∞                 // dodaj ∞ da bude > R[n₂]
  R[n₂ + 1] = ∞                 // dodaj ∞ da bude > L[n₁]
  i = 1                         // indeks u levom
  j = 1                         // indeks u desnom podnizu
  for k = p to r                // "spoji" levi i desni
      if L[i] ≤ R[j]
          A[k] = L[i]           // kopiraj levi jer je manji
          i = i + 1             // pomeri se u levom podnizu
      else
          A[k] = R[j]           // kopiraj desni jer je manji
          j = j + 1             // pomeri se u desnom podnizu
```

Primer *Merge sort*



Primer *Merge sort*

```
SORTIRAJ-OBJEDINJAVANJEM(A)
  SORTIRAJ-OBJEDINJAVANJEM-KORAK(A, 1, A.length)

SORTIRAJ-OBJEDINJAVANJEM-KORAK(A, p, r)
  if p < r
      q = ⌊(p + r)/2⌋
      SORTIRAJ-OBJEDINJAVANJEM-KORAK(A, p, q)
      SORTIRAJ-OBJEDINJAVANJEM-KORAK(A, q+1, r)
      OBJEDINI(A, p, q, r)
```

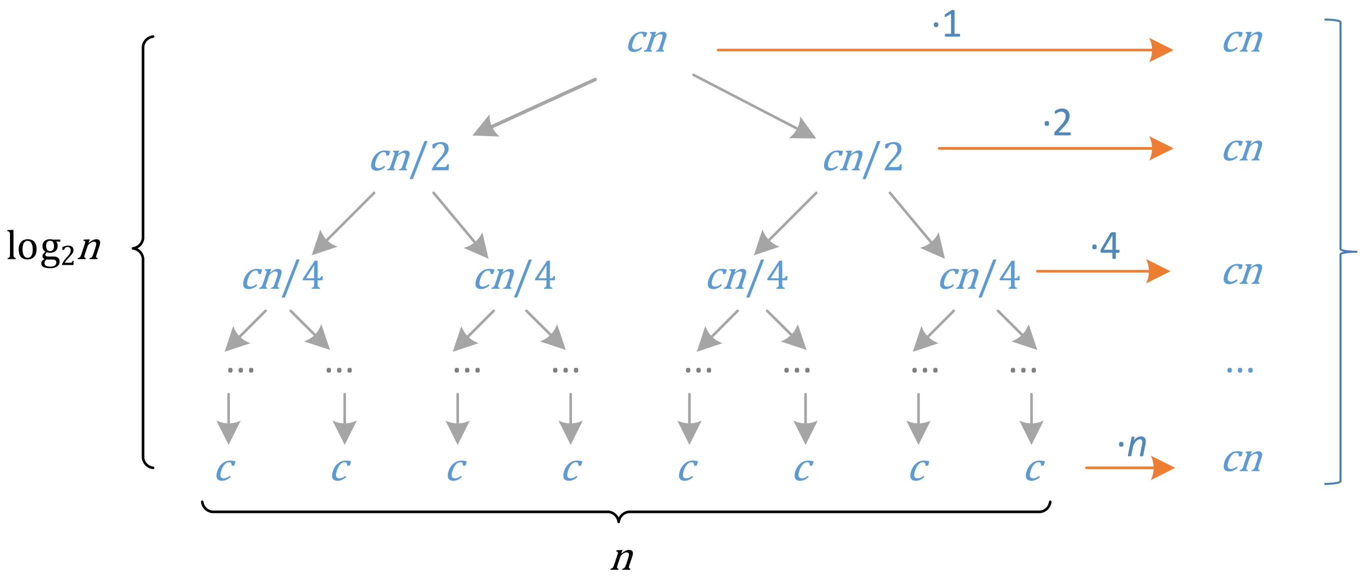


Legenda:
- podeli
- osvoji

Vreme izvršavanja *Merge sort*

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

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

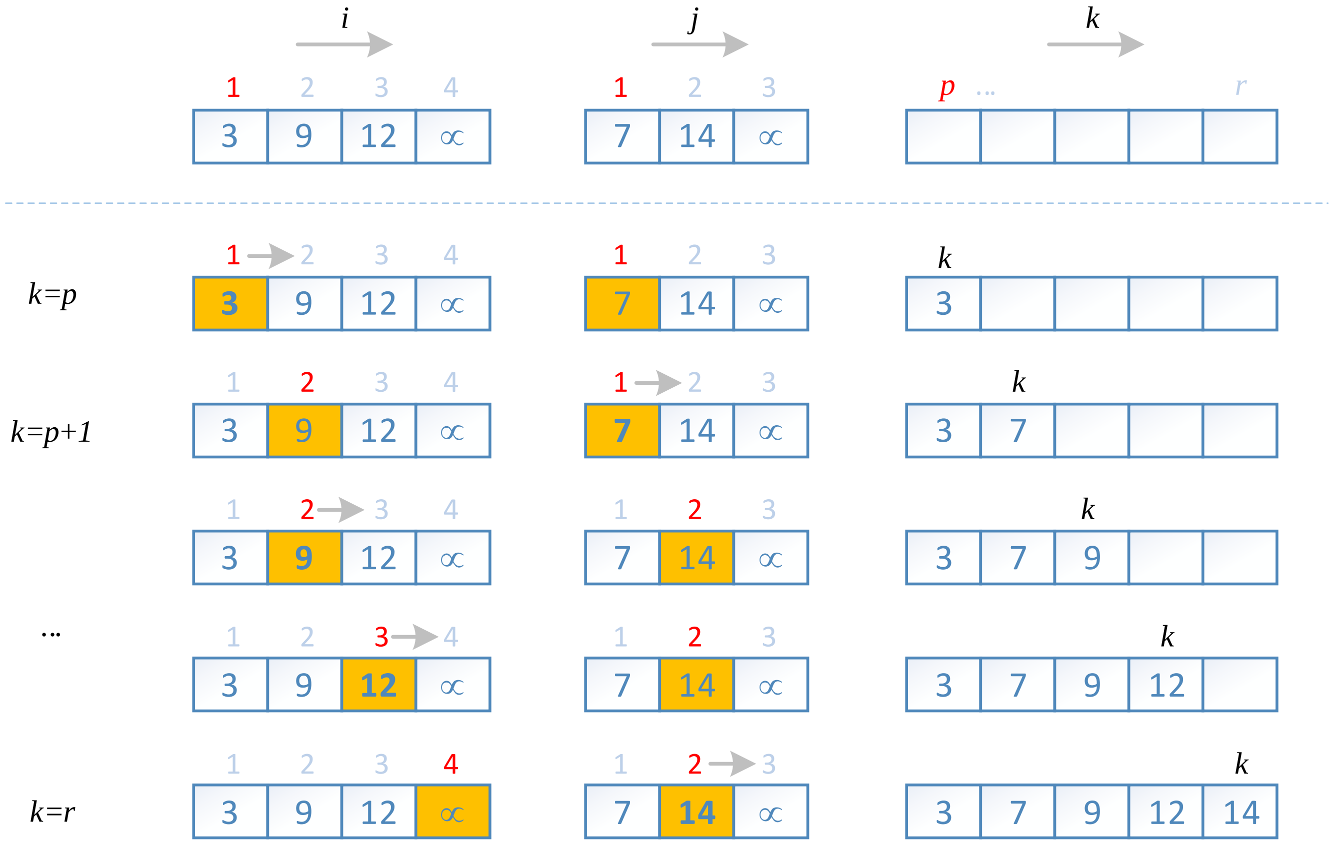


Suma = $cn \log_2 n$

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

Korak objedini u *Merge sort*

- Linearna složenost




```
OBJEDINI(A, p, q, r)
   n1 = q – p + 1
   n2 = r – q
   for i = 1 to n1
       L[i] = A[p + i - 1]
   for j = 1 to n2
       R[j] = A[q + j]
   L[n1 + 1] = ∞
   R[n2 + 1] = ∞
   i = 1
   j = 1
  for k = p to r
      if L[i] ≤ R[j]
         A[k] = L[i]
         i = i + 1
      else
         A[k] = R[j]
         j = j + 1
```

Sortiranje razdvajanjem - *Quicksort*

- *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 asimtotskoj 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

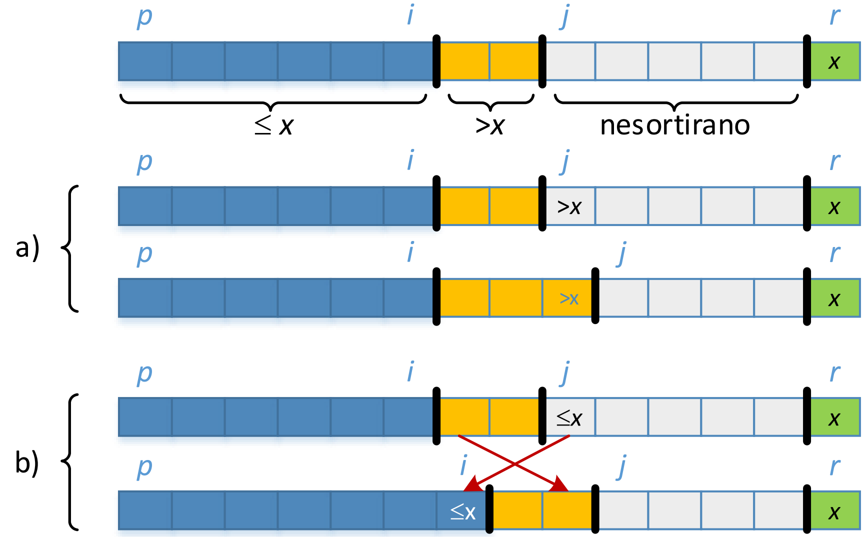
Quicksort – Algoritam (1)

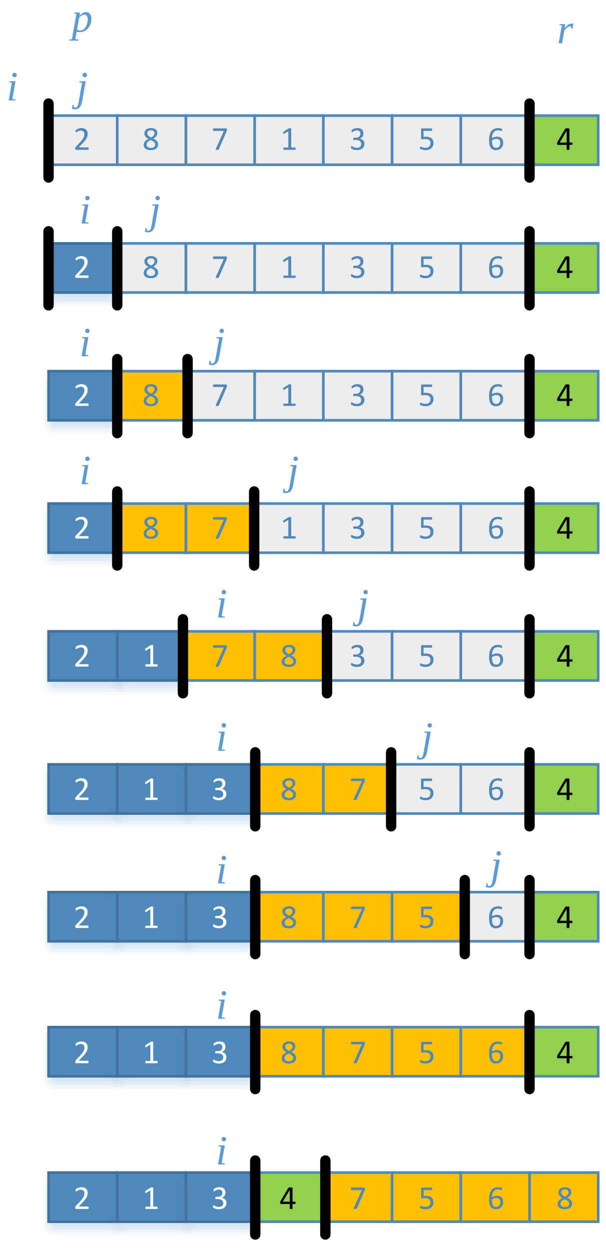
```
SORTIRAJ-RAZDVAJANJEM(A)
  SORTIRAJ-RAZDVAJANJEM-KORAK(A, 1, A.length)

SORTIRAJ-RAZDVAJANJEM-KORAK(A, p, r)
  if p < r
      q = PODELI(A, p, r)
      SORTIRAJ-RAZDVAJANJEM-KORAK(A, p, q-1)
      SORTIRAJ-RAZDVAJANJEM-KORAK(A, q+1, r)
```

…

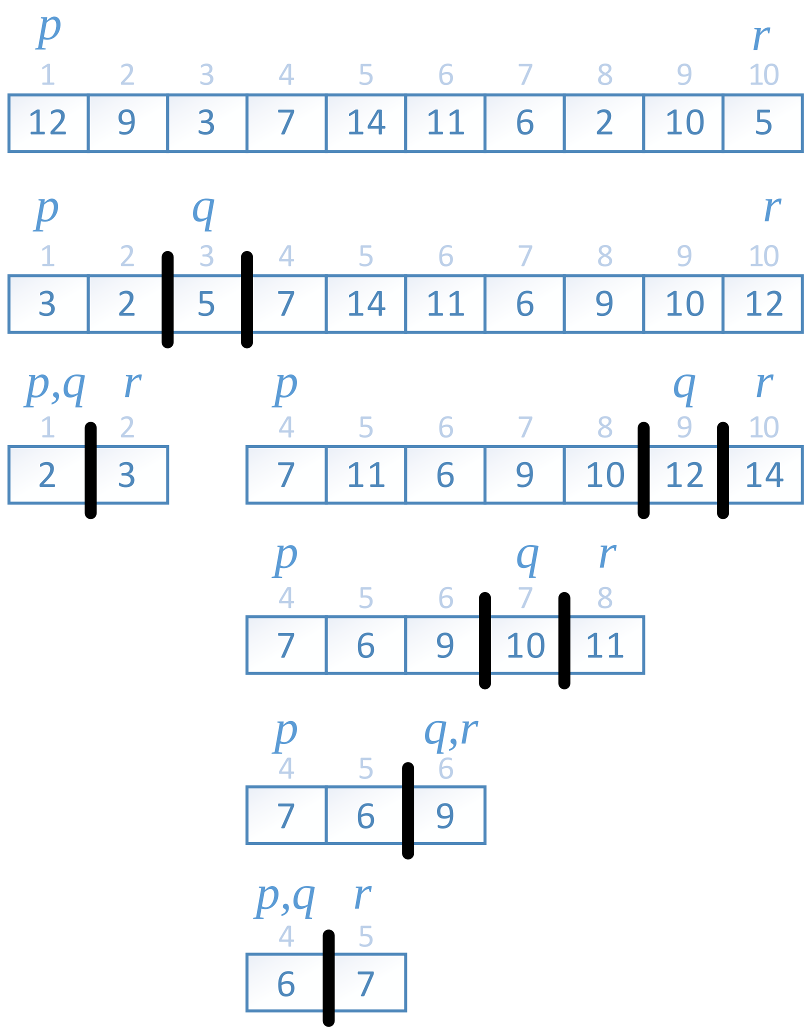
Quicksort – Algoritam (2)

```
PODELI(A, p, r)
x = A[r]               // pivot
i = p - 1
for j = p to r-1
    if A[j] ≤ x
        i = i + 1
        A[i] ↔ A[j]
A[i+1] ↔ A[r]
return i+1             // pozicija pivota
```





Quicksort - Primer



Vreme izvršavanja *Quicksort*-a

- Vreme izvršavanja zavisi od odnosa vrednosti u nizu
  - Ako je podela (engl. *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 (n-ti element je pivot)
  - Npr. svi elementi su manji od pivota
  - Tada je $T(n) = T(n-1) + T(0) + \Theta(n)$

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

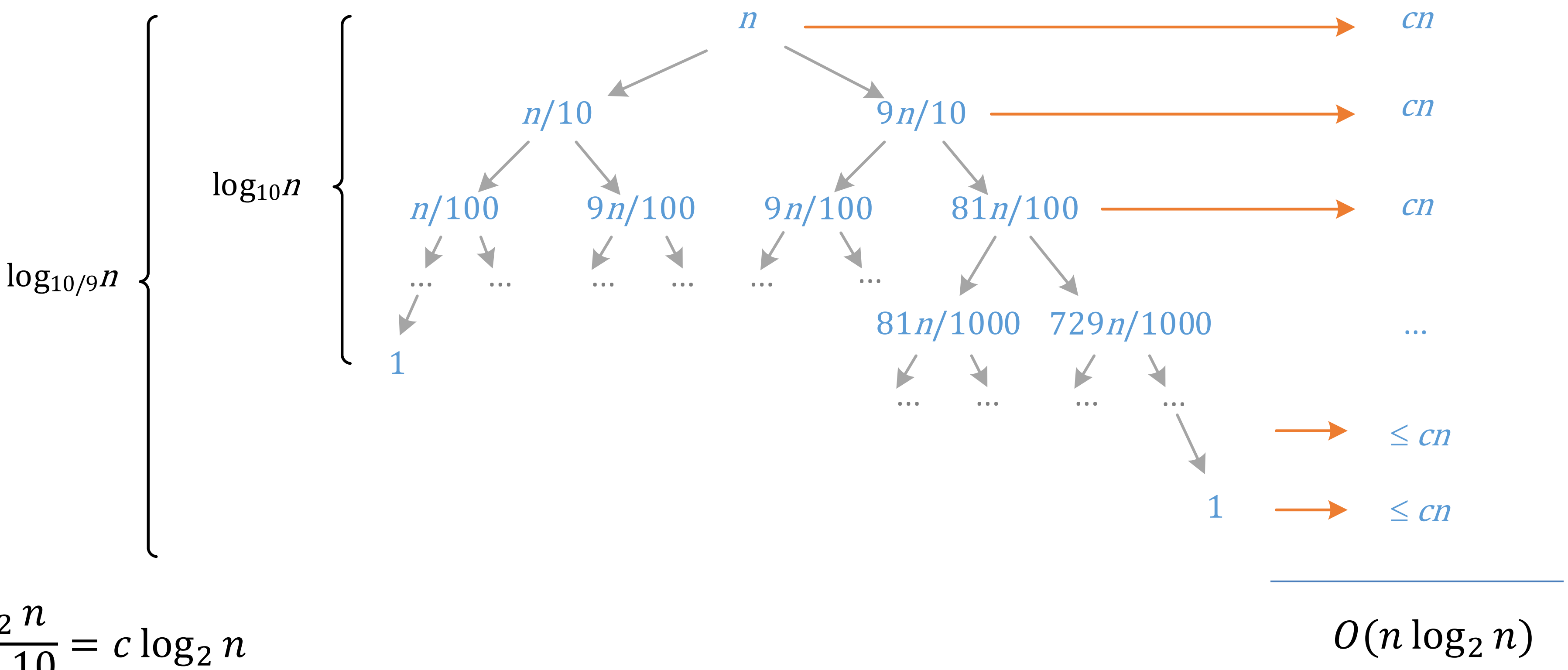
- Najbolji slučaj: od n elemenata se proizvode 2 podgrupe sa $n/2$ elemenata
  - Tada je $T(n) = 2T(n/2) + \Theta(n)$

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

…

Vreme izvršavanja *Quicksort*-a (2)

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



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

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 PODELI (*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 (pronađi jednog)

Algoritam	Najgori slučaj	Najbolji slučaj	Zahteva sortiran niz?
Linearna pretraga	$O(n)$	$\Omega(1)$	Ne
Binarna pretraga	$O(\log_2 n)$	$\Omega(1)$	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	$O(n^2)$	$\Omega(n)$	$O(n^2)$	Da
Merge sort	$\Theta(nlog_2n)$	$\Theta(nlog_2n)$	$\Theta(nlog_2n)$	Ne
Quicksort	$O(n^2)$	$\Omega(nlog_2n)$	$O(n^2)$	Da