Dizajn i analiza algoritama

Lekcija 9

leto 2019/2020

Prof. dr Branimir M. Trenkić

Problem sortiranja niza

- Vraćamo se ***problemu*** koji smo već rešavali – ***sortiranje niza***

Problem sortiranja niza

- Podsetimo se kako smo ***definisali*** problem:
- Dat je ***niz neuređenih brojeva*** - treba preurediti brojeve tog niza tako da oni obrazuju ***rastući niz***

- ***Ulaz***: Dat je niz a od n elemenata $a_1, a_2, \ldots, a_n$
- ***Zadatak***: ***Naći permutaciju*** svih ***indeksa*** elemenata niza $i_1, i_2, \ldots, i_n$ tako da ***novi prvi*** element a_{i1}, ***novi drugi*** element a_{i2} i tako dalje, ***novi n-ti*** element a_{in} u nizu ***zadovoljavaju uslov***

$$a_{i_1} \leq a_{i_2} \leq \cdots \leq a_{i_n}$$

Algoritmi za sortiranje niza

- Vraćamo se ***problemu*** koji smo već rešavali – ***sortiranje niza***

I. Sortiranje zamenjivanjem (***bubble-sort***)

II. Sortiranje umetanjem (***insert-sort***)

III. Sortiranje izborom (***select-sort***)

- ***Iterativni*** algoritmi
- ***Kvadratni*** algoritmi - $T(n) = O(n^2)$
- Posoje li ***efikasniji algoritmi*** koji rešavaju ovaj problem?

Merge-sort

- ***Sortiranje objedinjavanjem*** (***merge-sort***)
- Izvorno smo ***problem*** sortiranja ***definisali za nizove*** brojeva
- Isti problem možemo postaviti i za listu brojeva implementirane pomoću ***jednostruko povezane liste***
- (Ponoviti strukture podataka niz i povezana lista!)
  - ***Predavanje*** – lista brojeva implementirana pomoću ***niza***
  - ***Knjiga*** - lista brojeva implementirana pomoću ***jednostruko povezane liste***

Merge-sort

- Tipičan primer algoritma tipa ***"podeli-pa-vladaj"***
- ***Ideja*** (***tri koraka***)

1. ***Podeliti*** dati niz u ***dve polovine*** (otprilike po ***n/2*** elemenata u svakoj)
2. ***Sortirati*** ***(rekurzivno)*** obe polovine ***nezavisno***
3. ***Objediniti*** sortirane polovine u ceo sortiran niz

Merge-sort

- Ili grafički,
- ***Ideja*** (tri koraka)

Merge-sort

- Ili kroz ***primer***,

Dat je niz: $a = [8, 9, 2, 1, 3, 6, 2, 1, 7]$

1. **Podeliti** niz u dve polovine (levu i desnu)

   $a_1 = [8, 9, 2, 1, 3], a_2 = [6, 2, 1, 7]$

2. **Sortirati** (***rekurzivno***) obe polovine nezavisno

   $a_1 = [1, 2, 3, 8, 9], a_2 = [1, 2, 6, 7]$

3. **Objediniti** sortirane polovine u ceo sortiran niz

   $a = [1, 1, 2, 2, 3, 6, 7, 8, 9]$

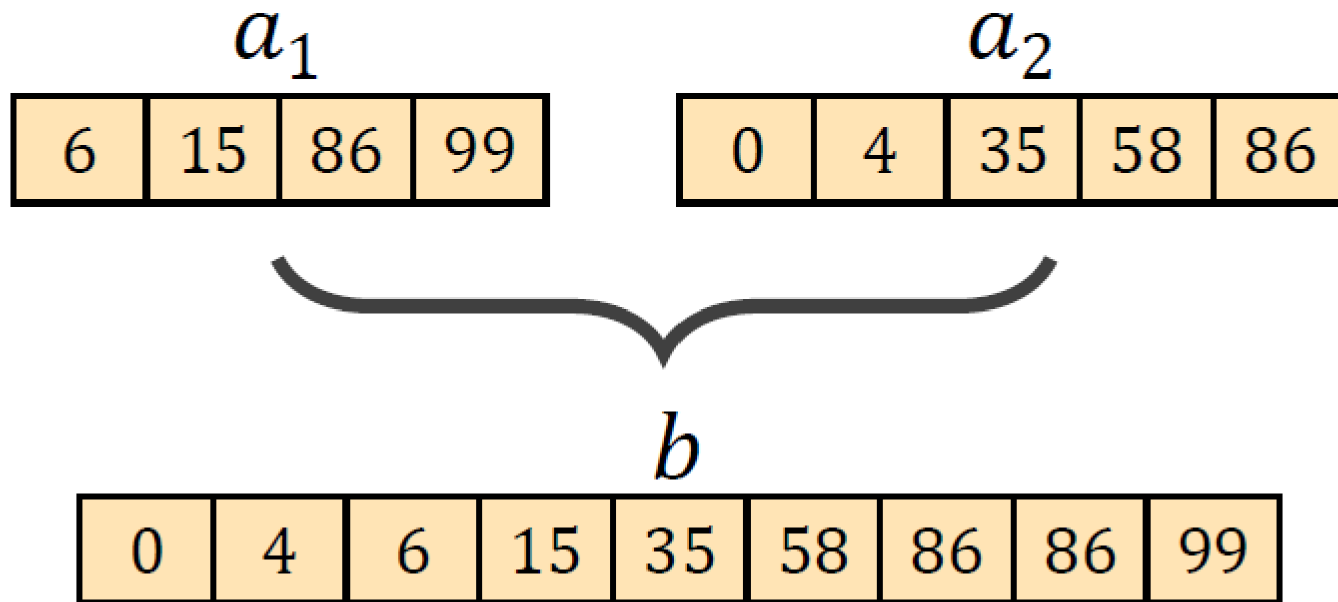
Postupak objedinjavanja

- U okviru ideje za rešavanje problema može se uočiti jedan ***pomoćni postupak*** – ***objedinjavanje*** (***merge***) ***dva*** sortirana niza ***u jedan*** sortirani niz

- Prvo ćemo analizirati i algoritamski rešiti ovaj pomoćni postupak

Postupak objedinjavanja

- ***Objedinjavanje*** (***merge***) dva sortirana niza u jedan sortirani niz
- Primer:



Postupak objedinjavanja

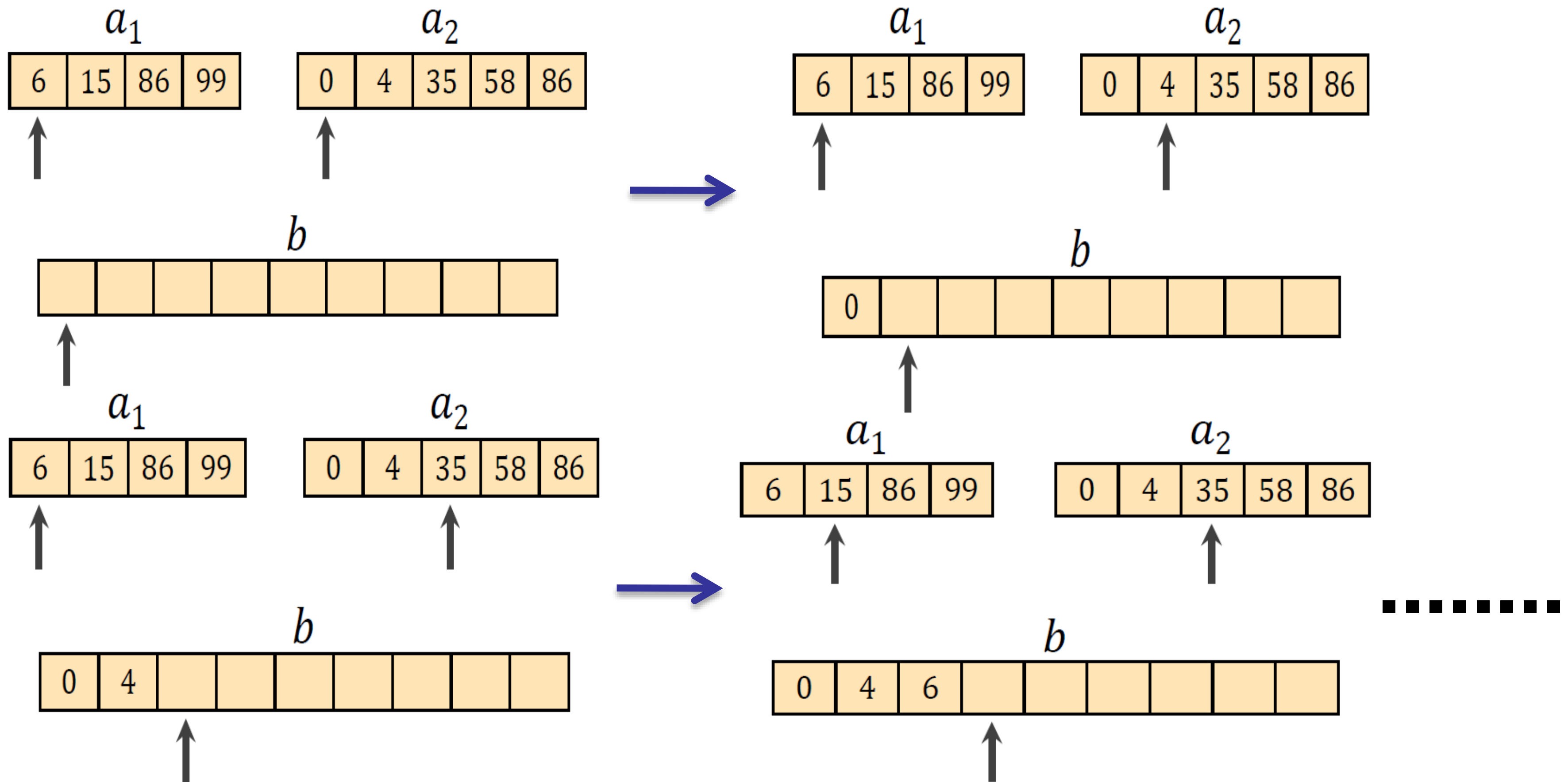
- ***Jednostavno rešenje*** objedinjavanja:

- Oba niza se ***paralelno ispituju*** od početka do kraja
- U svakom koraku ***upoređuju se tekući elementi*** oba niza
  i. Bira se manji element od njih
  ii. Ako su oba ista – uzima se bilo koji
- ***Izabrani element*** je ***naredni element*** u objedinjenom nizu

Postupak objedinjavanja

- ***Jednostavno rešenje*** demonstracija:

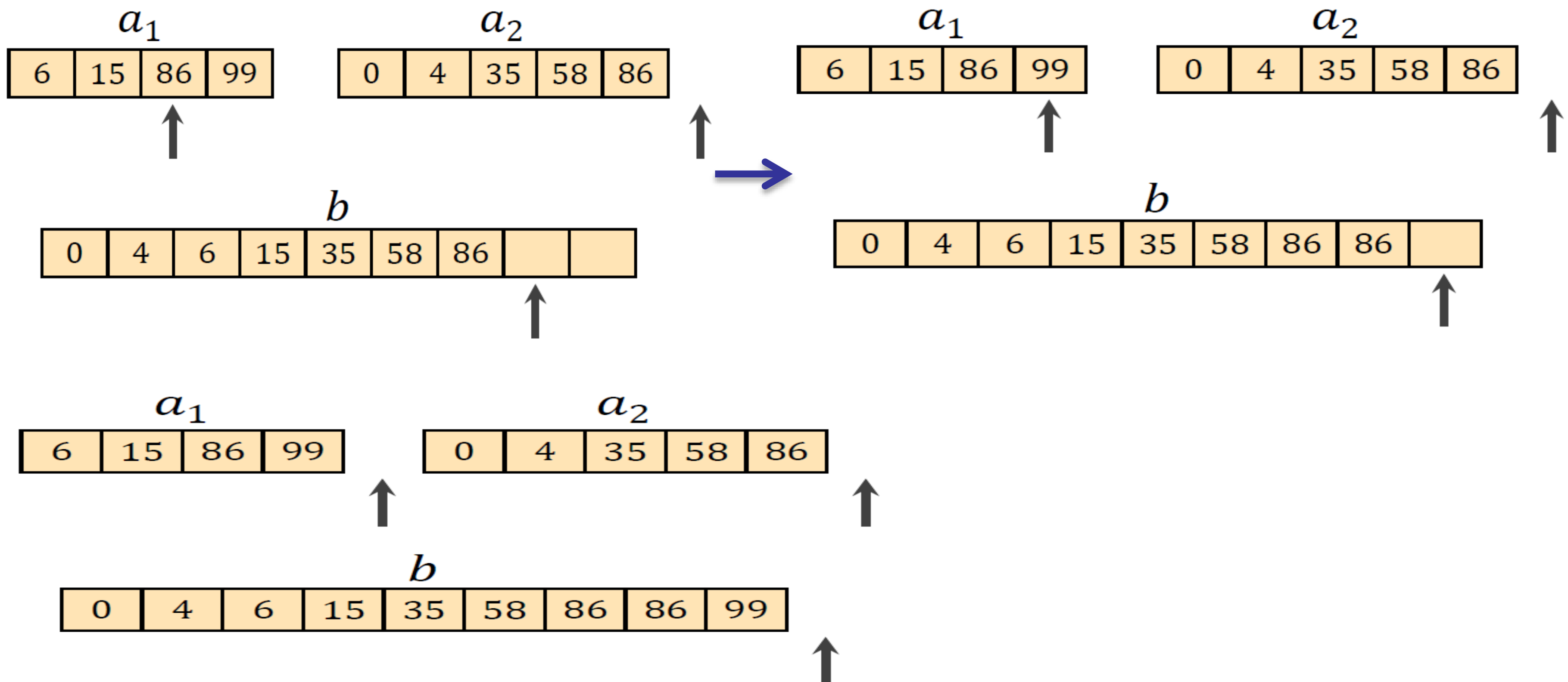
Početna pozicija:



Postupak objedinjavanja

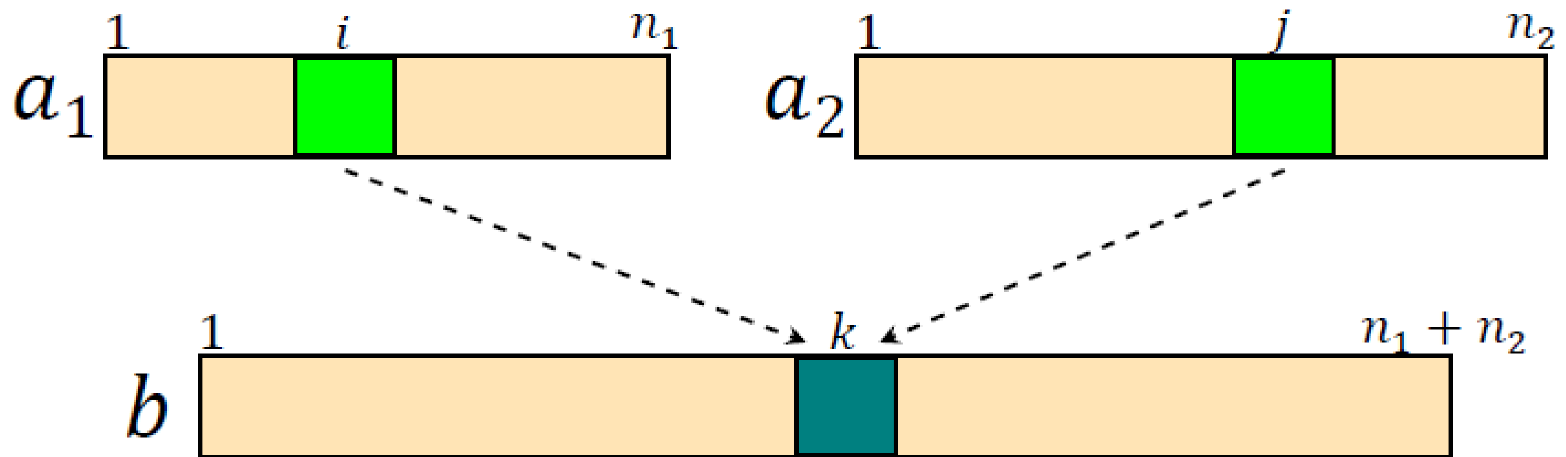
- ***Jednostavno rešenje*** demonstracija:

I na kraju:



Postupak objedinjavanja

- ***Detaljniji opis*** ***jednog koraka*** u postupku objedinjavanja (merge) dva sortirana niza u jedan sortirani niz:



- $a_1[i] < a_2[j] \Rightarrow b[k] = a_1[i],\ i = i + 1$
- $a_1[i] \geq a_2[j] \Rightarrow b[k] = a_2[j],\ j = j + 1$
- $k = k + 1$

Postupak objedinjavanja

- ***Algoritam***

- Nakon ove ***detaljne analize postupka*** za rešavanje problema objedinjavanja dva sortirana niza u jedan sortirani niz – možemo kreirati odgovarajući ***algoritam u pseudo-kodu***:

Postupak objedinjavanja

- ***Algoritam***

```
// Ulaz:  dva sortirana niza a1 i a2 dužine n1 i n2
// Izlaz: objedinjeni nizovi a1 i a2 u sortiran niz b
algorithm merge(a1, n1, a2, n2)
   i = 1; j = 1; k = 1; // indeksi nizova a1, a2 i b
   while ((i <= n1) && (j <= n2)) do
      if (a1[i] < a2[j]) then
         b[k] = a1[i]; i = i + 1;
      else
         b[k] = a2[j]; j = j + 1;
      k = k + 1;
   while (i <= n1) do
      b[k] = a1[i]; i = i + 1; k = k + 1;
   while (j <= n2) do
      b[k] = a2[j]; j = j + 1; k = k + 1;

   return b;
```

Postupak objedinjavanja

- ***Algoritam***
- Analiza vremena izvršavanja algoritma

$$merge(a_1, n_1, a_2, n_2)$$

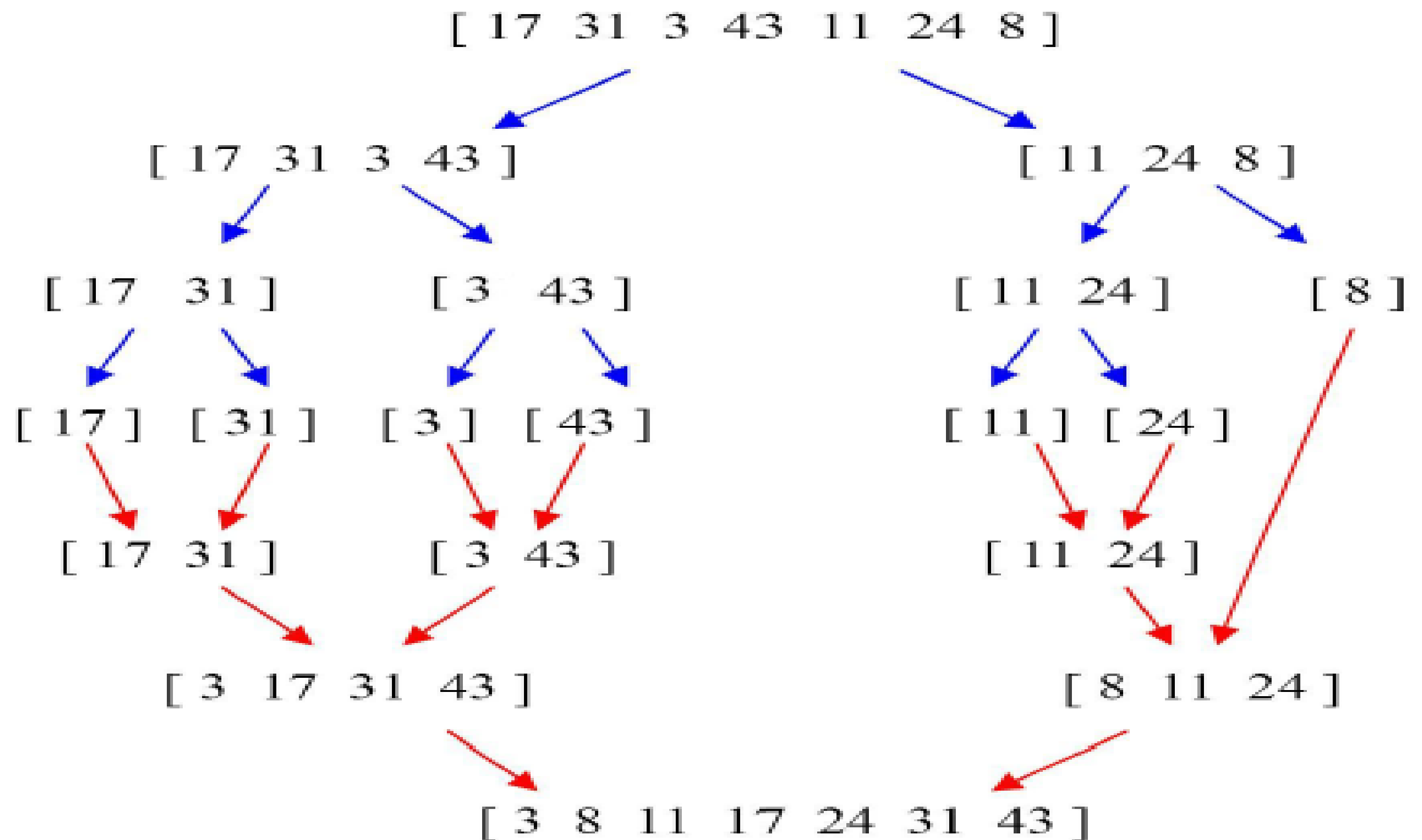
- $O(n_1) + O(n_2) = O(n_1 + n_2)$

Sortiranje objedinjavanjem

- ***Rekurzivni algoritam***
- ***Dodatno objašnjenje*** ***koraka 2***
- Korak 2: ***Sortirati*** ***(rekurzivno)*** obe polovine nezavisno
- ***Kako sortirati niz rekurzivno*** pomoću koraka 1 (***deljenje***)?
- Odgovor proističe iz činjenice – ***niz dužine 1 je po definiciji sortiran niz!***
  - Prema tome, ***korak 1 rekurzivno nastavljamo*** do god ne dođemo ***do nizova dužine 1***
  - Nakon toga, vršimo ***postupak objedinjavanja***

Sortiranje objedinjavanjem

- ***Rekurzivni algoritam***
- ***Izvršavanje – stablo izvršavanja:***



Sortiranje objedinjavanjem

- ***Rekurzivni algoritam***

```
// Ulaz:  niz a, njegov broj elemenata n
// Izlaz: sortiran niz a
algorithm merge-sort(a, n)

   if (n == 1) then  // bazni slučaj
      return a;
   else                  // opšti slučaj
      for i = 1 to n/2 do a1[i] = a[i];
      for i = 1 to n/2 do a2[i] = a[n/2+i];

      a1 = merge-sort(a1,n/2);
      a2 = merge-sort(a2,n/2);

      a = merge(a1,n/2,a2,n/2);

      return a;
```

Analiza algoritma

- ***Analizu vremena izvršavanja*** algoritma možemo izvršiti na ***dva načina***:

1. Korišćenjem ***rekurentne jednačine***
2. ***Neformalni način*** koji proističe iz same analize postupka

Analiza algoritma

- Analiza korišćenjem ***rekurentne jednačine***

- Vreme izvršavanja algoritma ***merge-sort(a, n)***:

$$T(n) = \begin{cases} c_1, & \text{ako je } n = 1 \\ 2T(n/2) + c_2 n + c_3 n, & \text{ako je } n > 1 \end{cases}$$

- Rešenje: $T(n) = O(n\log_2 n)$
- ***Zašto?***

Analiza algoritma

- Analiza korišćenjem ***rekurentne jednačine***
- Vreme izvršavanja algoritma ***merge-sort(a, n)***:

$$T(n)=\begin{cases} c_1, & \text{ako je } n=1 \\ 2T(n/2)+c_2 n+c_3 n, & \text{ako je } n>1 \end{cases}$$

```
// Ulaz:  niz a, njegov broj elemenata n
// Izlaz: sortiran niz a
algorithm merge-sort(a, n)

   if (n == 1) then  // bazni slučaj
      return a;
   else                  // opšti slučaj
      for i = 1 to n/2 do a1[i] = a[i];
      for i = 1 to n/2 do a2[i] = a[n/2+i];

      a1 = merge-sort(a1,n/2);
      a2 = merge-sort(a2,n/2);

      a = merge(a1,n/2,a2,n/2);

      return a;
```

Analiza algoritma

- Analiza korišćenjem ***rekurentne jednačine***

- Vreme izvršavanja algoritma ***merge-sort(a, n)***:
- Objediniti konstante
- Neka je $\boldsymbol{c = max(c_1, c_2+c_3)}$

$$T(n)=\begin{cases} c, & \text{ako je } n=1 \\ 2T(n/2)+cn, & \text{ako je } n>1 \end{cases}$$

Analiza algoritma

- Analiza korišćenjem ***rekurentne jednačine***

$$T(n) = 2T(n/2) + cn$$
$$= 2\big(2T(n/4) + cn/2\big) + cn = 4T(n/4) + 2cn$$
$$= 4\big(2T(n/8) + cn/4\big) + 2cn = 8T(n/8) + 3cn$$
$$\vdots$$
$$= n \cdot T(1) + \log n \cdot cn$$
$$= cn + \log n \cdot cn$$
$$= O(n \log n)$$

$2^iT(n/2^i) + icn$ **(i = broj nivoa)**

Iz početnog uslova T(1) = c za $n/2^i = 1$ iz čega sledi da je $n = 2^i$ i $i = \log n$

Analiza algoritma

- Neformalna analiza
- Može se na ***jednostavan način*** pokazati da je ***vremenska složenost*** ovog algoritma jednaka $O(n\log_2 n)$
- Pretpostavimo da je ***veličina niza*** potencija broja 2, tj. $n = 2^m$
- Pošto se pri svakom rekurzivnom pozivu niz deli na dva podniza, sve do dužine ***1*** - proizlazi da je ***broj nivoa*** deljenja niza jednak $\log_2 n$

Analiza algoritma

- Neformalna analiza
- **Šta se dešava na k-tom nivou?**
- ***Na k-tom nivou*** niz je ***podeljen na 2^k podnizova*** ***dužine $n/2^k$***
- ***Objedinjavanje*** sortiranih nizova ***na k-tom nivou*** ima složenost $2^k \cdot O(n/2^k) = O(n)$
- A pošto ***ima $\log_2 n$ nivoa***, proizlazi da je ukupna složenost jednaka $O(n\log_2 n)$

Sortiranje razdvajanjem

- ***Sortiranje razdvajanjem*** (***quick-sort***)
- Ili ***"brzo" sortiranje***
- Reč je opet o ***rekurzivnom algoritmu***
- Jedan je od ***najčešće korišćenih*** metoda u praksi
- Mada ***u najgorem slučaju nije bolji od kvadratnih algoritama*** za sortiranje – ***u praktičnim primenama*** se pokazao kao ***vrlo efikasan***, odatle je i dobio ime "brzi" metod sortiranja

Sortiranje razdvajanjem

- ***Sortiranje razdvajanjem*** (***quick-sort***)
- ***Ideja*** (***dva*** koraka)

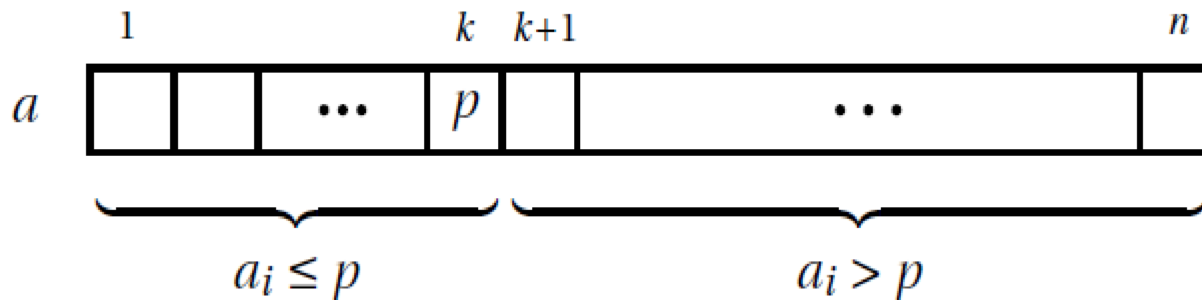
1. ***Podeliti niz*** na ***dva dela***, tako da su svi elementi prvog dela ***manji*** od svih elemenata drugog dela

2. ***Sortirati*** (***rekurzivno***) prvi i drugi deo niza nezavisno

Sortiranje razdvajanjem

- ***Sortiranje razdvajanjem*** (***quick-sort***)
- ***Ideja*** (***dva*** koraka)

1. ***Izabrati*** element niza ***p*** (***pivot***) i ***preurediti niz*** tako da p zauzima svoje pravo mesto u sortiranom nizu



2. ***Sortirati*** (***rekurzivno***) delove niza ***levo i desno*** od ***p*** nezavisno

Sortiranje razdvajanjem

- ***Sortiranje razdvajanjem*** (***quick-sort***)
- ***Ideja*** (konkretnije)

Sortiranje razdvajanjem

- ***Izbor*** dobrog ***pivota*** je ***ključni korak*** u algoritmu
- ***Cilj*** prilikom izbora – ***veličine*** oblasti sa obe strane pivota otprilike ***podjednake***
- Na taj način se dolazi do ***efikasnog sortiranja***
- ***Element*** koji se nalazi na sredini sortiranog niza naziva se ***medijana***
- Nalaženje medijane nije jednostavan postupak
- Pivot se obično bira ***na jednostavan način*** – u nadi da je izabrani element blizu medijane

Sortiranje razdvajanjem

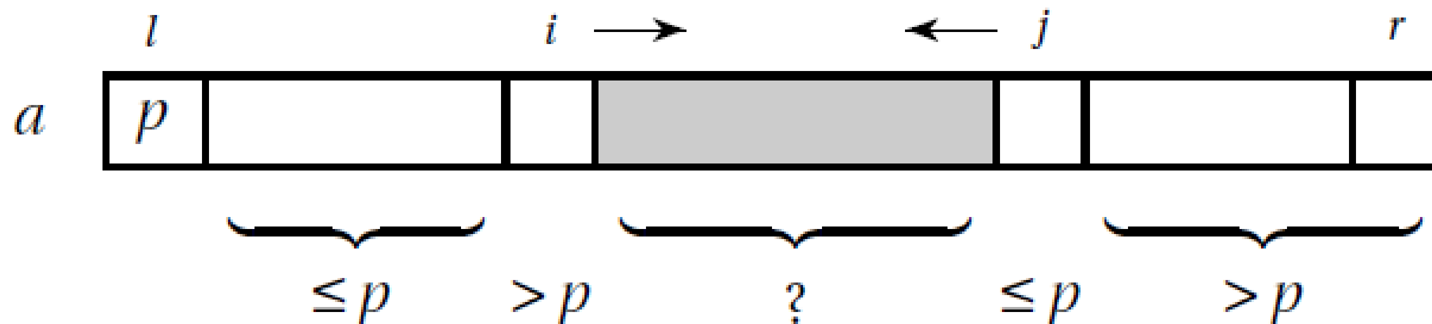
- Izbor pivota:
  - ***Prvi element*** niza
  - ***Poslednji element*** niza
  - ***Srednji element*** niza
  - ***Slučajni element*** niza

- Analiziramo slučaj razdvajanja niza kada se bira ***njegov prvi element za pivota***

Sortiranje razdvajanjem

- Razdvajanja niza kada se bira njegov prvi element za pivota
- Pretpostavimo da želimo ***da podelimo podniz*** $a_l, \ldots, a_r$, gde je $1 \leq l \leq r \leq n$ i neka je pivot $p = a_l$

1. Od ***levog*** kraja, preskočiti sve elemente $\leq p$
2. Od ***desnog*** kraja, preskočiti sve elemente $> p$
3. $a_i > p$ *i* $a_j \leq p \Rightarrow swap(a_i, a_j)$



Sortiranje razdvajanjem

- ***Algoritam***

```
// Ulaz:  podniz niza a ograničen indeksima od l do r
// Izlaz: preuređen podniz i indeks u kome se nalazi pivot
algorithm partition(a, l, r)

    // Pretpostavlja se da je a_{r+1} = +∞
    p = a[l];
    i = l + 1; j = r;
    while (i < j) do
        while (a[i] <= p) do i = i + 1;
        while (a[j] > p)  do j = j - 1;
        swap(a[i],a[j]);
    swap(a[l],a[j]);

    return j;
```

Tehnički problem koji smo ignorisali!
Šta ako je pivot najveći element u podnizu?

Vraća indeks tog pivota u preiređenom nizu!

Sortiranje razdvajanjem

- ***Algoritam***

```
// Ulaz:  podniz niza a ograničen indeksima od l do r
// Izlaz: sortiran podniz
algorithm quick-sort(a, l, r)

   if (l < r) then
      k = partition(a,l,r);
      quick-sort(a,l,k-1);
      quick-sort(a,k+1,r);

   return a;
```

Napomena: Ako želimo da sortiramo ceo niz, poziv je *quick-sort(a, 1, n)*

Analiza algoritma

1) Analiza vremenske složenosti algoritma
   partition (a, 1, n)

$$T(n) = O(n)$$

Analiza algoritma

2) Analiza vremenske složenosti algoritma
   quick-sort(a, 1, n)

$$T(n) = \begin{cases} O(1), & n = 1 \\ T(k-1) + T(n-k) + O(n), & n > 1 \end{cases}$$

```
// Ulaz:  podniz niza a ograničen indeksima od l do r
// Izlaz: sortiran podniz
algorithm quick-sort(a, l, r)

  if (l < r) then
    k = partition(a,l,r);
    quick-sort(a,l,k-1);
    quick-sort(a,k+1,r);

  return a;
```

Analiza algoritma

- Rešenje za ***quick-sort(a, 1, n)***?

- Najbolji slučaj: $k \approx n/2$

$$T(n) = 2T(n/2) + cn$$

$$\Rightarrow \quad T(n) = O(n \log n)$$

- Najgori slučaj: $k = 1$ ili $k = n - 1$

$$T(n) = T(n-1) + T(1) + cn = T(n-1) + cn$$

$$\Rightarrow \quad T(n) = O(n^2)$$

- U praksi: $T(n) = O(n \log n)$