Katedra za računarsku tehniku i informatiku

Algoritmi i strukture podataka 2

Milo V. Tomašević
Marko Mišić
Maja Vukasović

Odsek za softversko inženjerstvo [SI]

I **Linearne strukture podataka**

II **Nelinearne strukture podataka**

III **Pretraživanje**

IV **Sortiranje**

Sortiranje

- Preuređivanje skupa podataka po nekom utvrđenom rasporedu
- Veoma česta aktivnost
- Cilj:
  - ✓ efikasnije pretraživanje
  - ✓ provera jednakosti
  - ✓ sistematizovani prikaz
- Spektar algoritama složenosti od O(n) do O(n^2)
- Poredak (↗,↘)određen vrednostima polja ključa

Sortiranje

- Sortiranje:
  - ✓ $K_1, ..., K_n \Rightarrow K_{p1} \leq ... \leq K_{pn}$
  - ✓ $p1, ..., pn$ – permutacija od $1..n$

- Stabilnost – $i < j$ i $K_{pi} = K_{pj} \Rightarrow pi < pj$

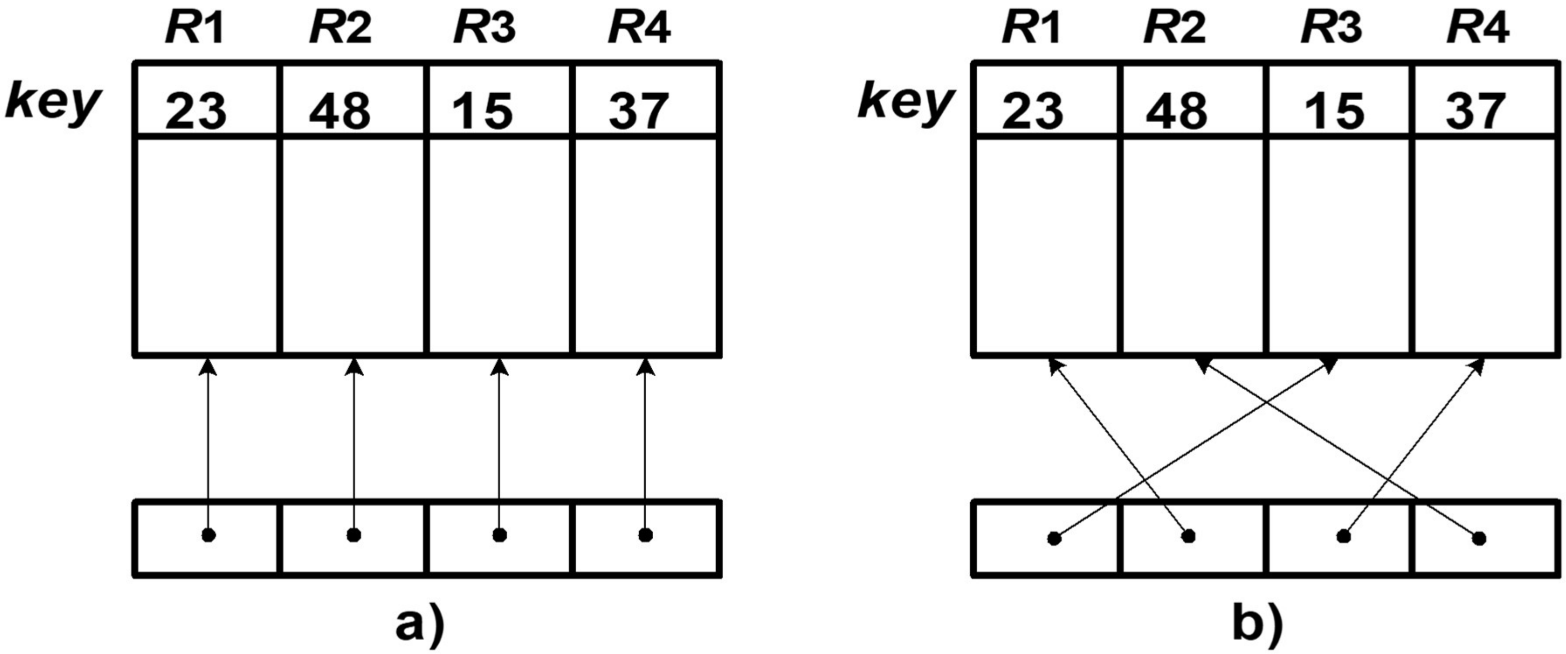
- Po mestu sortiranja:
  - ✓ unutrašnje
  - ✓ spoljašnje

Unutrašnje sortiranje

- Sortiranje kada su svi podaci u operativnoj memoriji
- Sortiranje nizova i lista
- Sortiranje *in situ*
- Indikatori performanse
  - ✓ broj koraka
  - ✓ broj poređenja ključeva
  - ✓ broj premeštanja ključeva

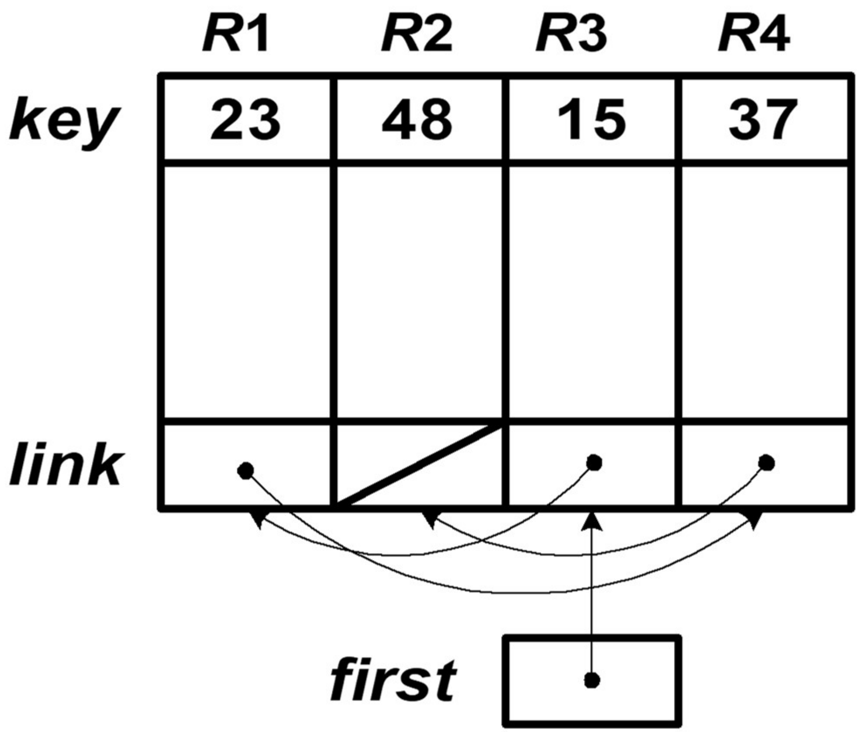
Unutrašnje sortiranje

- Sortiranje po adresi
- Izbegava premeštanje zapisa



Unutrašnje sortiranje

- Ulančavanje zapisa po poretku
- Izbegava premeštanje zapisa



Sortiranje poređenjem

- Isključivo zasnovano na poređenju ključeva
- Podela:
  - metodi umetanja
  - metodi selekcije
  - metodi zamene
  - metodi spajanja
- Direktni metodi
  - jednostavni
  - lošije perfromanse
- Poboljšani metodi (do O(*n* log *n*))

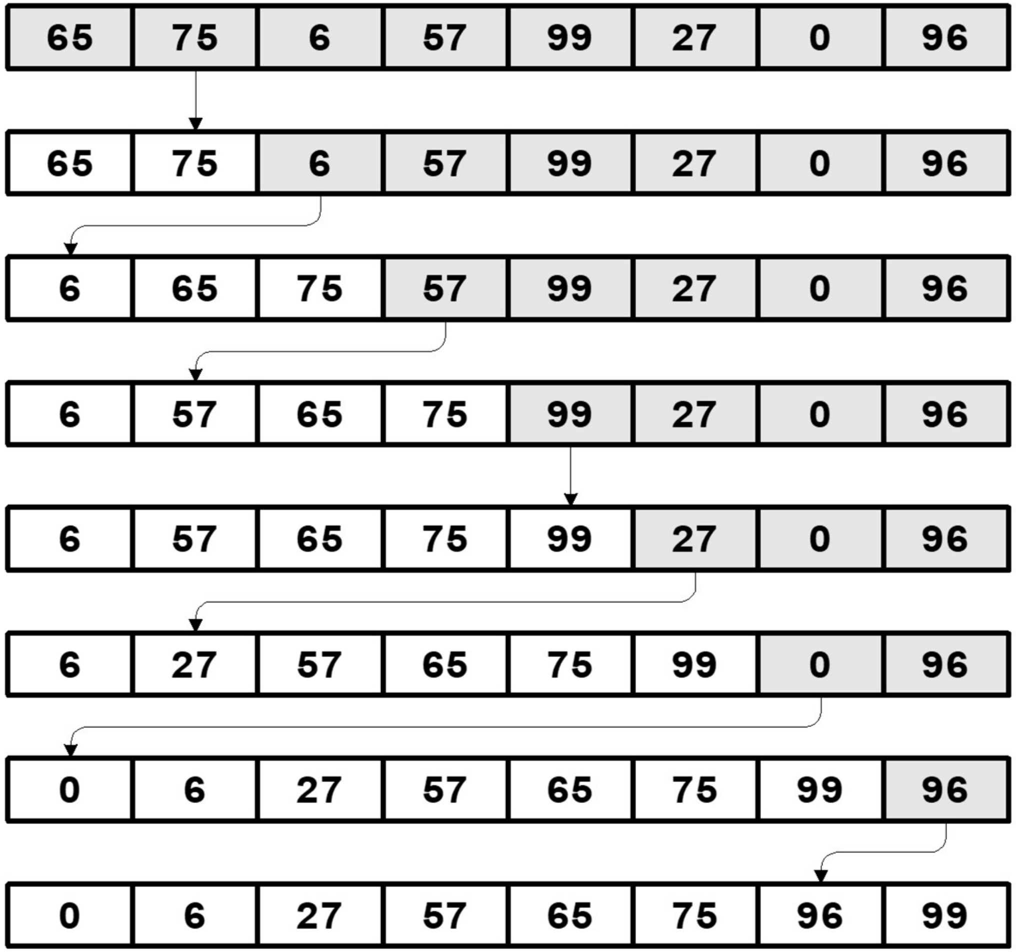
Metodi umetanja

- Princip – po jedan element iz neuređenog dela umeće se u uređeni deo

- **Direktno umetanje**

<u>INSERTION-SORT(*a*)</u>
for *i* = 2 **to** *n* **do**
    K = *a* [*i*]
    j = *i* – 1
    while (*j* > 0) and (*a* [*j*]) > *K*) **do**
        a [*j* + 1] = *a* [*j*]
        j = *j* –1
    end_while
    a [*j* + 1] = *K*
end_for

Direktno umetanje



Direktno umetanje

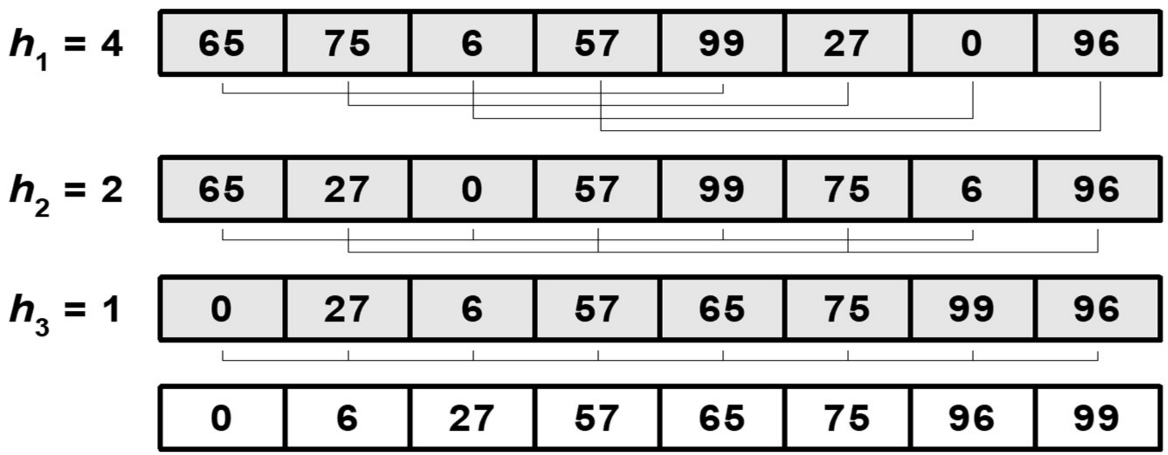
- Najbolji slučaj – uređen niz, $C_{min} = n - 1 \Rightarrow O(n)$
- Najgori slučaj – obrnuto uređen niz,
  $C_{max} = M_{max} = \sum(i - 1) \Rightarrow O(n^2)$
- Prosečan slučaj bolji od najgoreg samo za faktor 1/2
- Vrlo dobar za male nizove i skoro uređene nizove

Poboljšanja

- Problemi
  - broj poređenja
  - broj premeštanja
- Binarno pretraživanje uređenog dela
  - smanjuje broj poređenja, ali ne i premeštanja
  - isti red složenosti
  - za uređen niz čak i lošije
- Jednostruko ulančana lista umesto niza
  - vektor indeksa simulira pokazivače
  - smanjuje broj premeštanja, ali ne i poređenja
  - dodatni prostor

Shellsort

- Umetanje sa smanjenjem inkrementa *h*
- Grupe elemenata na ekvidistantnom razmaku *h*
- Grupe se sortiraju metodom direktnog umetanja
- Inkrementi se smanjuju sve do 1



Shellsort

```
SHELL-SORT(a, h)
for i = 1 to t do
    inc = h[i]
    for j = inc + 1 to n do
        y =a[j]
        k = j – inc
        while (k ≥ 1) and (y < a[k]) do
            a[k + inc] = a[k]
            k = k – inc
        end_while
        a[k + inc] = y
    end_for
end_for
```

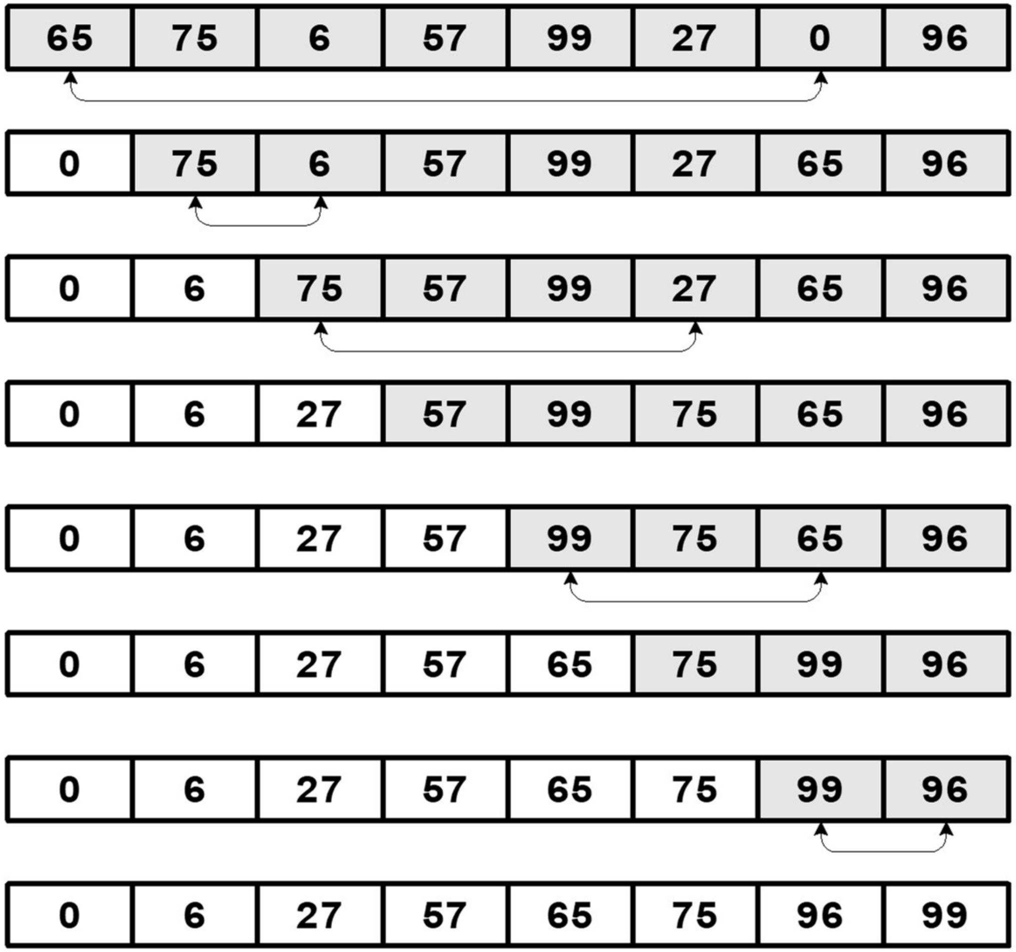
Shellsort

- Direktno sortiranje
  - ✓ male nesortirane grupe
  - ✓ veće dosta sortirane grupe
- Sekvenca inkrementa $h_1, h_2, ..., h_t$
  - ✓ $h_{i+1} < h_i,\ 1 \leq i < t$
  - ✓ $h_t = 1$
- Knuth – $h_{i-1} = 3h_i + 1$, $h_t = 1$, $t = \log_3 n - 1$
- Bolji uzajamno prosti inkrementi
- Složenost $O(n(\log n)^2)$ (empirijski $O(n^{1.3})$)

Metodi selekcije

- Princip – selektuje najmanji element iz neuređenog i stavlja ga na kraj uređenog dela
- Ponekad procesiranje neuređenog dela u strukturu koja olakšava selekciju (prioritetni red)
- **Direktna selekcija**
- Nema dodatnog procesiranja neuređenog dela
- Sličnosti i razlike sa direktnim umetanjem
- Poređenja u neuređenom delu (ne može da počne dok nema sve podatke)

Direktna selekcija



Direktna selekcija

```
SELECTION-SORT(a)
for i = 1 to n - 1 do
    min = a[i]
    pos = i
    for j = i + 1 to n do
        if (a[j] < min) then
            min = a[j]
            pos = j
        end_if
    end_for
    a[pos] = a[i]
    a[i] = min
end_for
```

Metodi selekcije

- Jedna zamena i $n - i$ poređenja po koraku
- $C = \sum (n - i) \Rightarrow O(n^2)$
- Nema razlike između najboljeg i najgoreg slučaja
- Problem – broj poređenja
- **Kvadratna selekcija**
- Selekcija po grupama – lokalni i globalni minimumi
- Složenost $O(n\sqrt{n})$

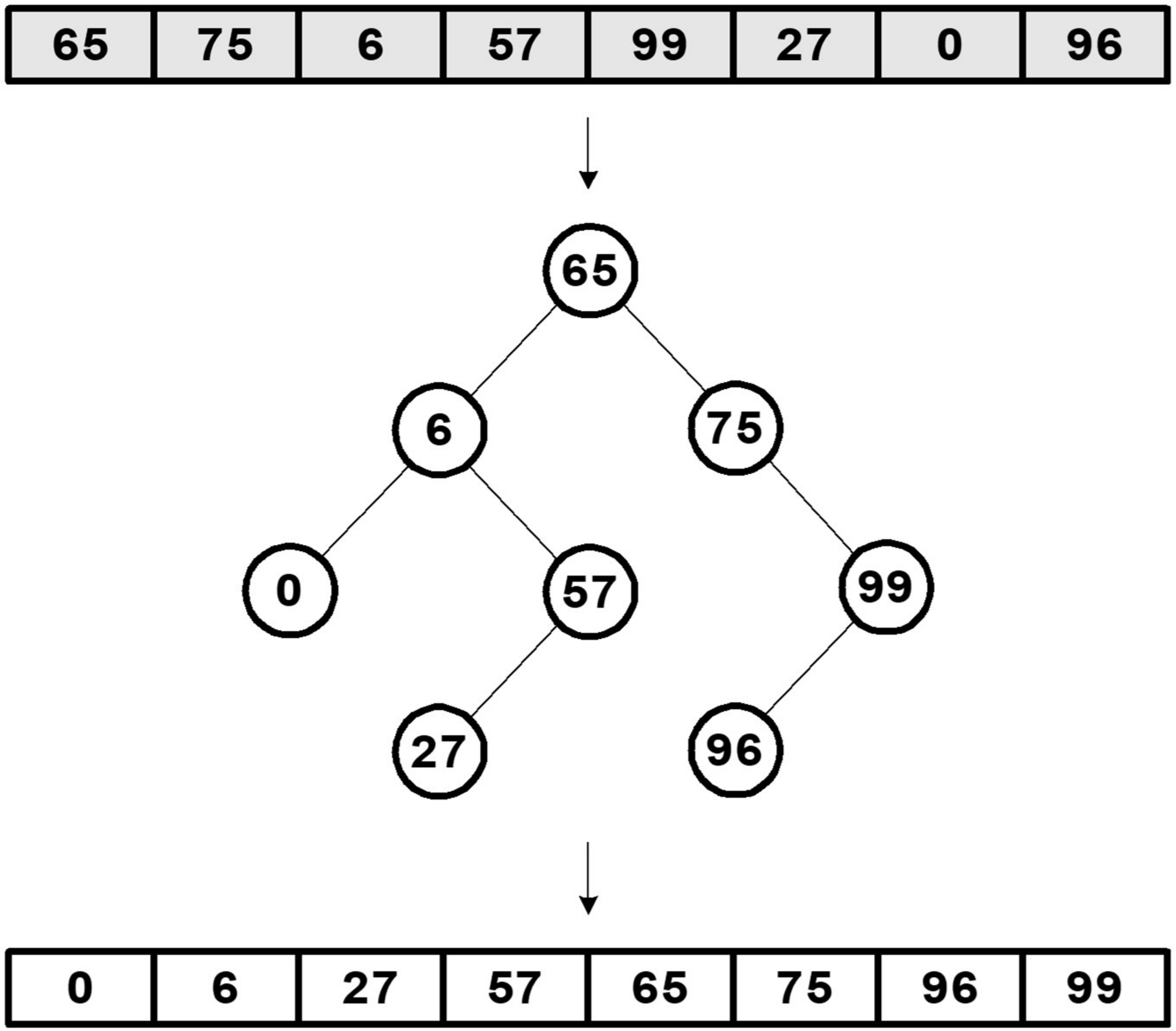
Kvadratna selekcija

65	75	6	57	99	27	0	96
0	65	75	6	57	99	27	96
0	6	65	75	57	99	27	96
0	6	27	65	75	57	99	96
0	6	27	57	65	75	99	96
0	6	27	57	65	75	99	96
0	6	27	57	65	75	99	96
0	6	27	57	65	75	96	99

Sortiranje pomoću BST

- Elementi neuređenog niza se umeću u stablo binarnog pretraživanja
- *Inorder* obilazak
- Isti ključevi:
  - ✓ u desno podstablo
  - ✓ ulančana lista
- Složenost O(n log n), ali nije garantovana
- Pogodan za dinamičke skupove podataka

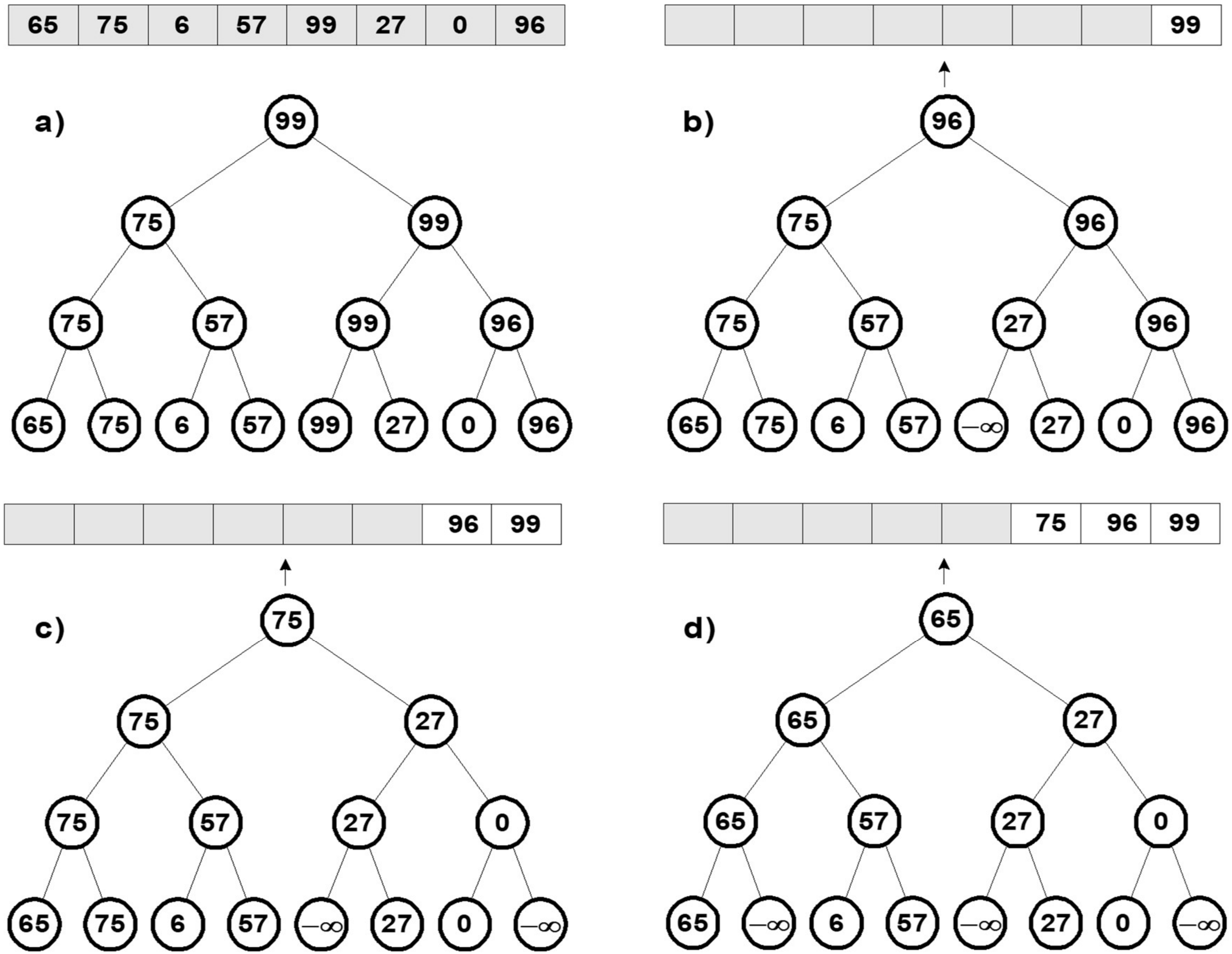
Sortiranje pomoću BST



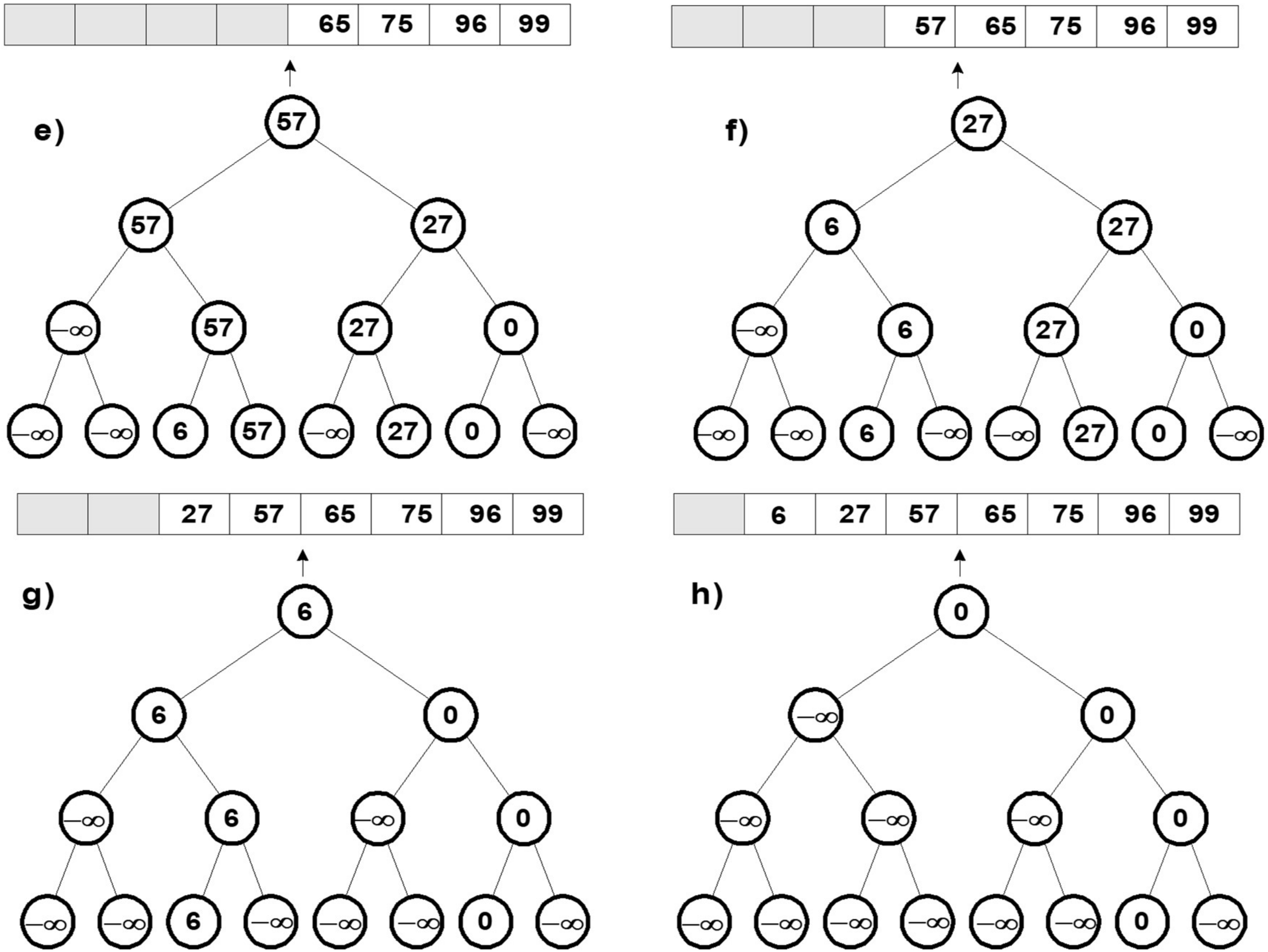
Stablo selekcije

- Problem garantovane performanse
- Balansirano stablo
- Poređenja po kup-sistemu
- Selekcija iz korena
- Ažuriranje po jednoj putanji od lista do korena
- Performanse
  - ✓ generisanje stabla – O(n)
  - ✓ selekcija – O(n log n)

Stablo selekcije



Stablo selekcije



Heapsort

- Nedostaci stabla selekcije
  - ✓ dodatni prostor
  - ✓ nepotrebna poređenja
- Struktura selekcije - heap
  - ✓ kompletno ili skoro kompletno binarno stablo
  - ✓ otac veći ili jednak sa oba sina
- Sekvencijalna implementacija $a\ [1:n]$
  - ✓ $a\ [i] \geq a\ [2i]$ i $a\ [i] \geq a\ [2i + 1]$, $1 \leq i < 2i < 2i + 1 \leq n$
- Efikasna implementacija prioritetnog reda
- Sortiranje na mestu

Heapsort

- Dve faze algoritma:
  - ✓ generisanje heap-a
  - ✓ procesiranje heap-a

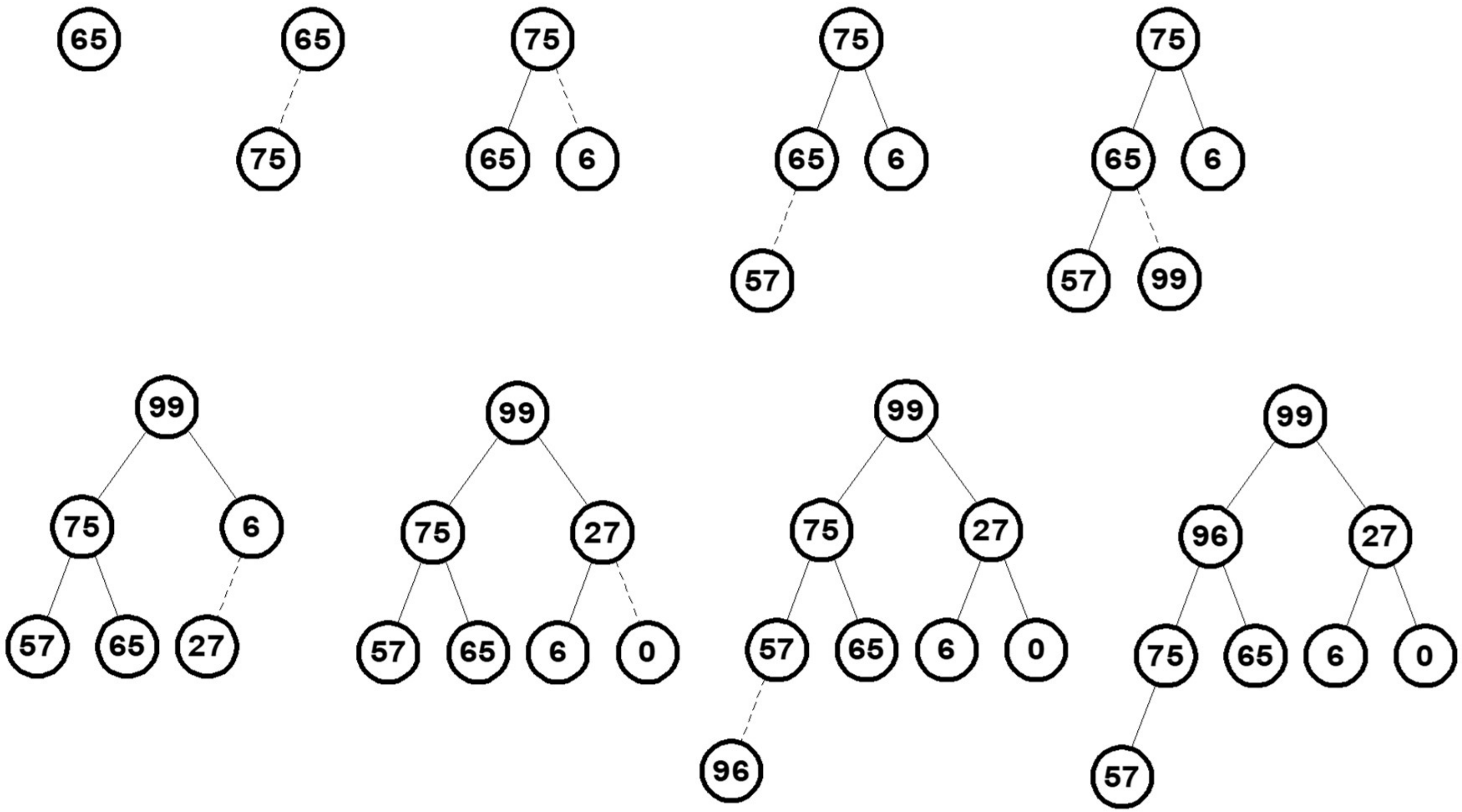
```
HEAPSORT(a)
for i = 2 to n do
      nhe = a[i]
      s = i
      f = s/2
      while ((s > 1) and (a[f] < nhe)) do
            a[s] = a[f]
            s = f
            f = s/2
      end_while
      a[s] = nhe
end_for
```

Heapsort

```
for i = n downto 2 do
    last =a[i]
    a[i] = a[1]
    f = 1
    if ((i –1) ≥ 3 and (a[3] > a[2])) then
        s = 3
    else
        s = 2
    end_if
    while (s ≤ i – 1) and (a[s] > last) do
        a[f] = a[s]
        f = s
        s = 2f
        if ((s + 1) ≤ i – 1) and (a[s + 1] > a[s]) then
            s = s + 1
        end_if
    end_while
    a[f] = last
end_for
```

Heapsort

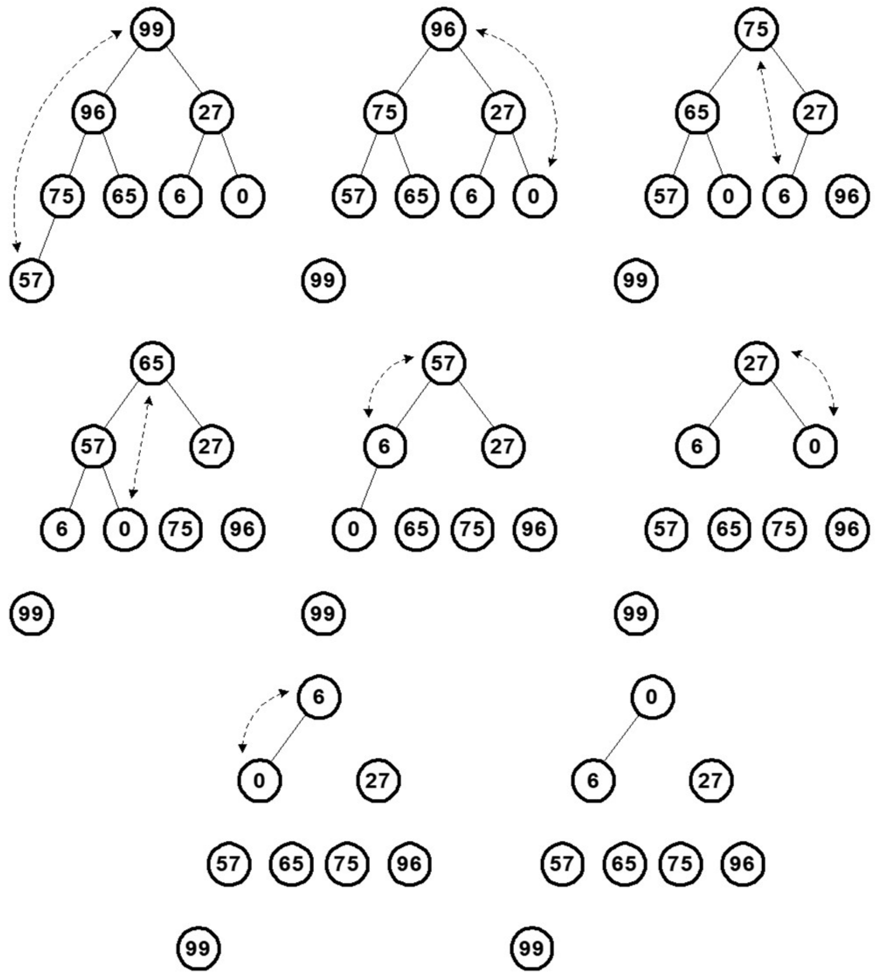
65	75	6	57	99	27	0	96



99	96	27	75	65	6	0	57

Heapsort

99	96	27	75	65	6	0	57



0	6	27	57	65	75	96	99

Heapsort

- Alternativna realizacija
  - ✓ ADJUST pretvara u heap stablo sa korenom *i* kada su mu oba podstabla već heap-ovi
  - ✓ poziva se i pri generisanju i pri procesiranju

- Performanse
  - ✓ generisanje – O(n log n) (sa ADJUST čak O(n))
  - ✓ procesiranje – O(n log n)

- Prosečan broj zamena 0.5n log n

- Garantovan najgori slučaj O(n log n)

- Za izdvajanje k najvećih ključeva (k << n) – ~O(n)

Heapsort

```
ADJUST(a, i, n)
K = a [i]
j = 2i
while (j ≤ n) do
    if ((j < n) and (a [j] < a[j + 1])) then
        j = j + 1
    end_if
    if (K ≥ a [j ]) then
        a [j/2] = K
        return
    else
        a [j/2] = a [j]
        j = 2j
    end_if
end_while
a [j/2] = K
```

```
HEAPSORT-1(a)
for i = n/2 downto 1 do
    ADJUST(a, i, n)
end_for
for i = n - 1 downto 1 do
    a [i + 1] ↔ a [1]
    ADJUST(a, 1, i)
end_for
```

Metodi zamene

- Princip – zamena mesta dva elementa
  koji nisu u pravilnom poretku
- Primenjuje se i u drugim metodima
- **Direktna zamena (*bubblesort*)**
- Zamena mesta dva susedna elementa
- Najveći element izađe na početak uređenog dela
- Optimizacije
  - ✓ najviša pozicija na kojoj je bila zamena u prolazu
  - ✓ kraj – prolaz bez zamena

Bubblesort

	65	75	6	57	99	27	0	96
pos = 7	65	6	57	75	27	0	96	99
pos = 5	6	57	65	27	0	75	96	99
pos = 4	6	57	27	0	65	75	96	99
pos = 3	6	27	0	57	65	75	96	99
pos = 2	6	0	27	57	65	75	96	99
pos = 1	0	6	27	57	65	75	96	99

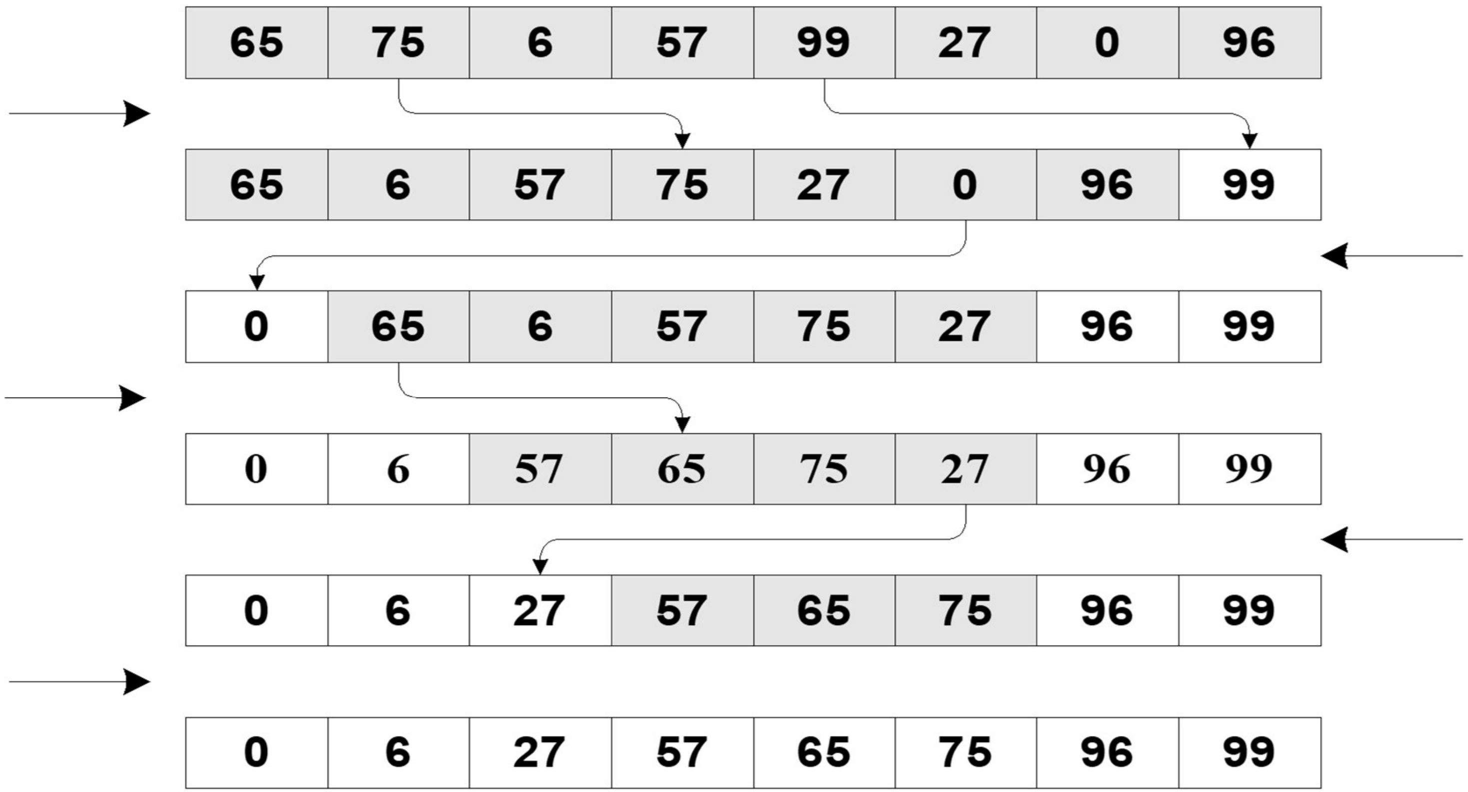
Bubblesort

```
BUBBLESORT(a)
pos = n
repeat
    bound = pos
    pos = 0
    for i = 1 to bound - 1 do
        if (a[i] > a[i + 1]) then
            a[i] ↔ a[i + 1]
            pos = i
        end_if
    end_for
until pos = 0
```

Bubblesort

- Najbolji slučaj – sortiran niz
  - ✓ $M_{min} = 0$, $C_{min} = n - 1 \Rightarrow O(n)$
- Najgori slučaj – obrnuto sortiran niz
  - ✓ $M_{max} = C_{max} = 0.5(n^2 - n) \Rightarrow O(n^2)$
- Prosečni slučaj
  - ✓ $C_{ave} = 0.5(n^2 - n \ln n)$, $M_{ave} = 0.25(n^2 - n) \Rightarrow O(n^2)$
- Poboljšanje – ***Shakersort***
- Alterativno menja smer prolaska
- Manje poređenja, ali isti red složenosti

Shakersort


65 75 6 57 99 27 0 96
65 6 57 75 27 0 96 99
0 65 6 57 75 27 96 99
0 6 57 65 75 27 96 99
0 6 27 57 65 75 96 99
0 6 27 57 65 75 96 99

Quicksort

- Poređenja i zamene na većoj udaljenosti
- Particijsko sortiranje (Hoare)
- Razdvojni element (pivot)
- Donja particija + pivot + gornja particija
- Rekurzivna podela na particije do jedinične veličine

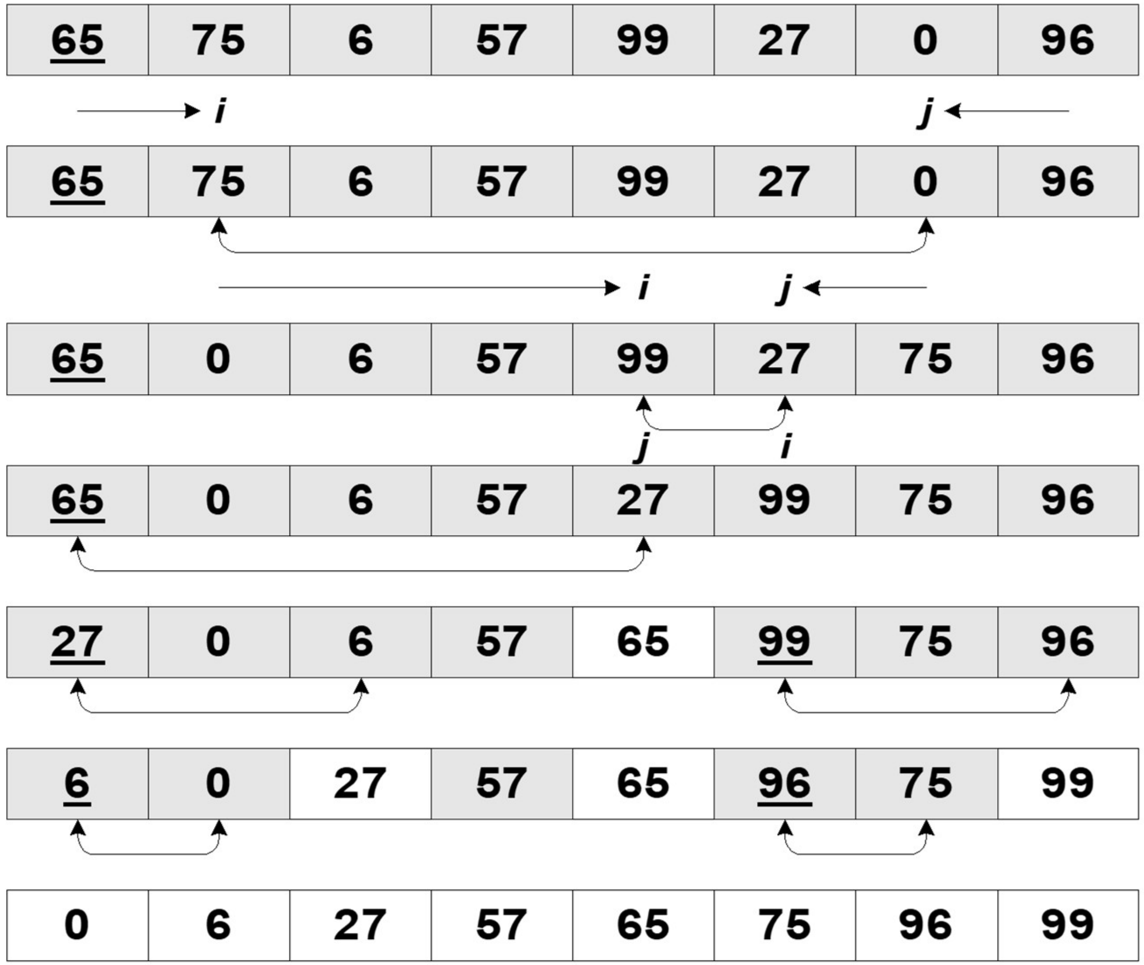
```
QUICKSORT(a, low, high)
j = PARTITION(a, low, high)
QUICKSORT(a, low, j - 1)
QUICKSORT(a, j + 1, high)
```

Quicksort

```
PARTITION(a, down, up)
i = down
j = up
pivot = a[down]
while (i < j) do
    while ((a[i] ≤ pivot) and (i < j)) do
        i = i + 1
    end_while
    while (a[j] > pivot) do
        j = j – 1
    end_while
    if (i < j) then
        a[i] ↔ a[j]
    end_if
end_while
```

```
a[down] = a[j]
a[j] = pivot
return j
```

Quicksort


65 75 6 57 99 27 0 96
i
j
65 75 6 57 99 27 0 96
i
j
65 0 6 57 99 27 75 96
j
i
65 0 6 57 27 99 75 96
27 0 6 57 65 99 75 96
6 0 27 57 65 96 75 99
0 6 27 57 65 75 96 99

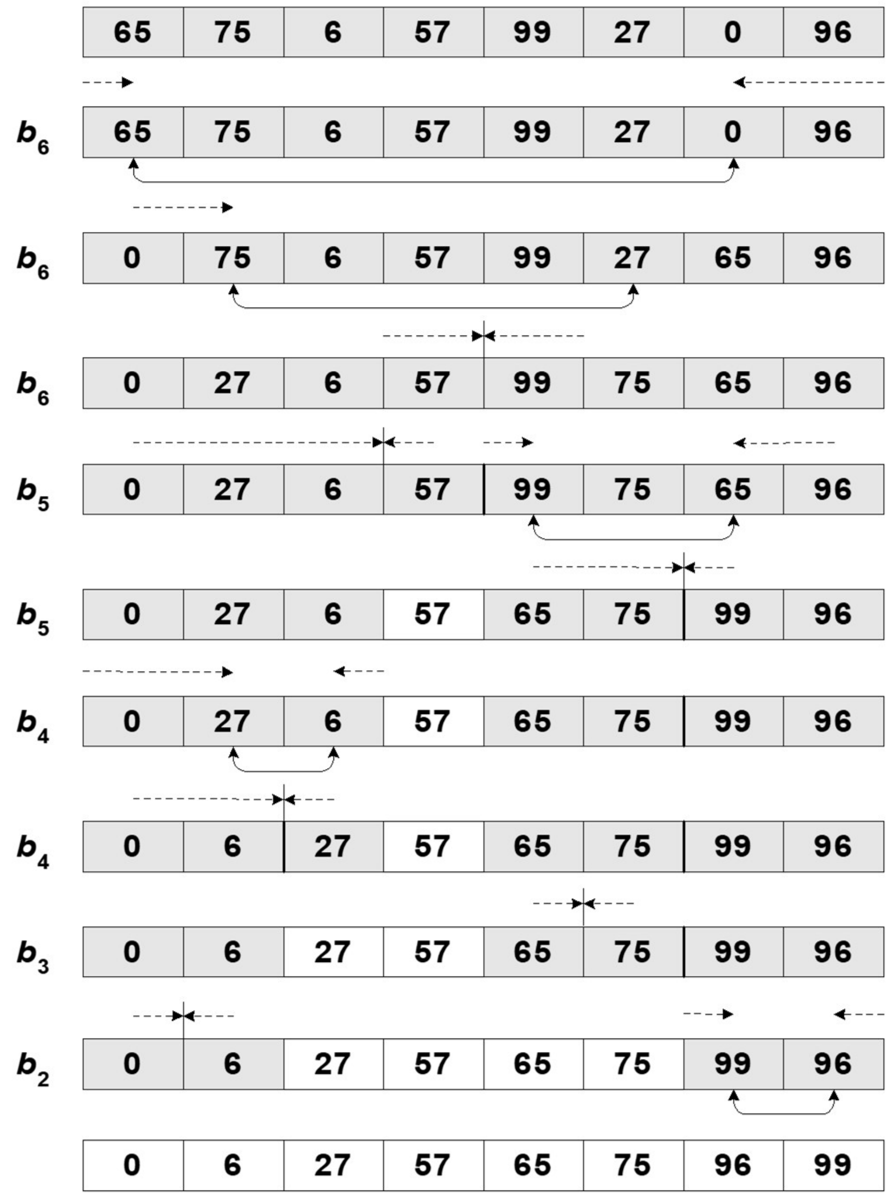
Quicksort

- Iterativna realizacija sa stekom
- Najbolji slučaj – jednake particije $\Rightarrow$ O(n log n)
- Najgori slučaj – jedna particija $\Rightarrow$ O(n^2)
- Prosečan slučaj gori od najboljeg za 38% $\Rightarrow$ O(n log n)
- Veliki uticaj izbora pivota
  - ✓ slučajan izbor
  - ✓ srednji od tri (ili više) elementa particije
  - ✓ srednja vrednost (meansort)

Pobitno razdvajanje

- Binarna reprezentacija ključa $(b_{m\text{-}1}\ b_{m\text{-}2}\ ....b_0)_2$
- Počinje se od najstarijeg bita
- Donja ($b_i = 0$) i gornja ($b_i = 1$) particija
- Složenost – $O(n \log n)$

Pobitno razdvajanje



65 = 1000001
75 = 1001011
6 = 0000110
57 = 0111001
99 = 1100011
27 = 0011011
0 = 0000000
96 = 1100000

Metodi spajanja

- **Direktno spajanje**

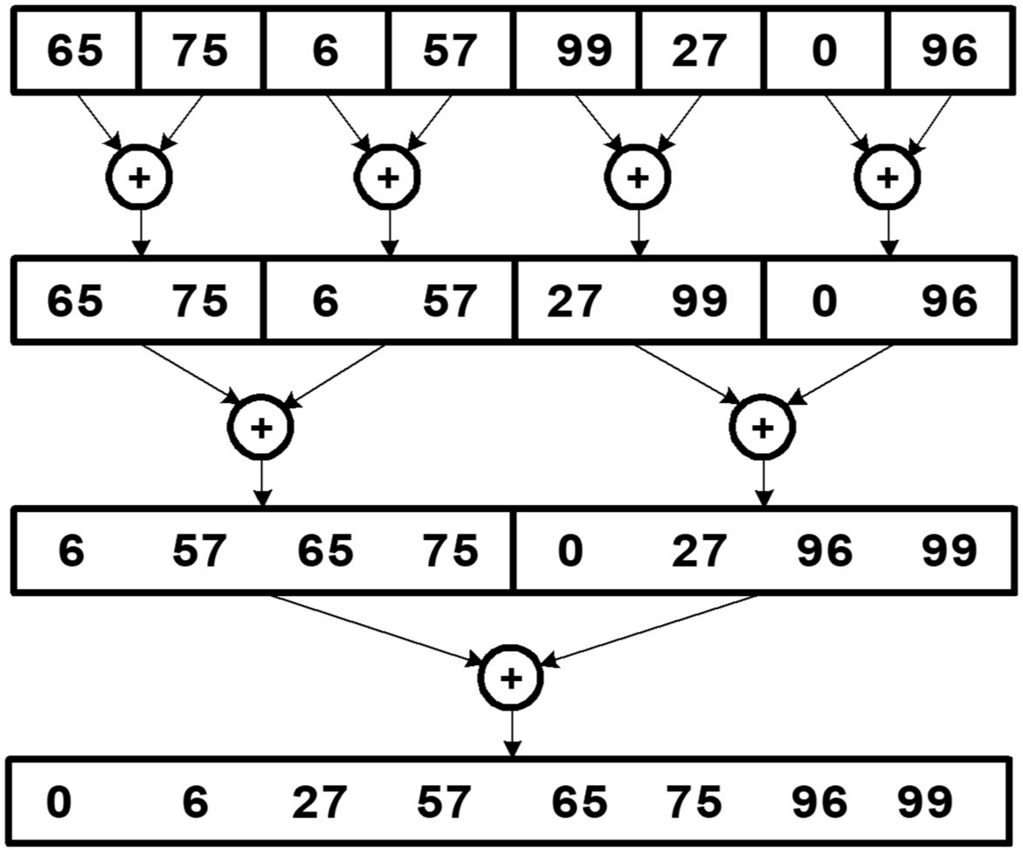
- Susedni elementi se spoje u uređene dvojke, pa u uređene četvorke, ...

- Potreban pomoćni niz

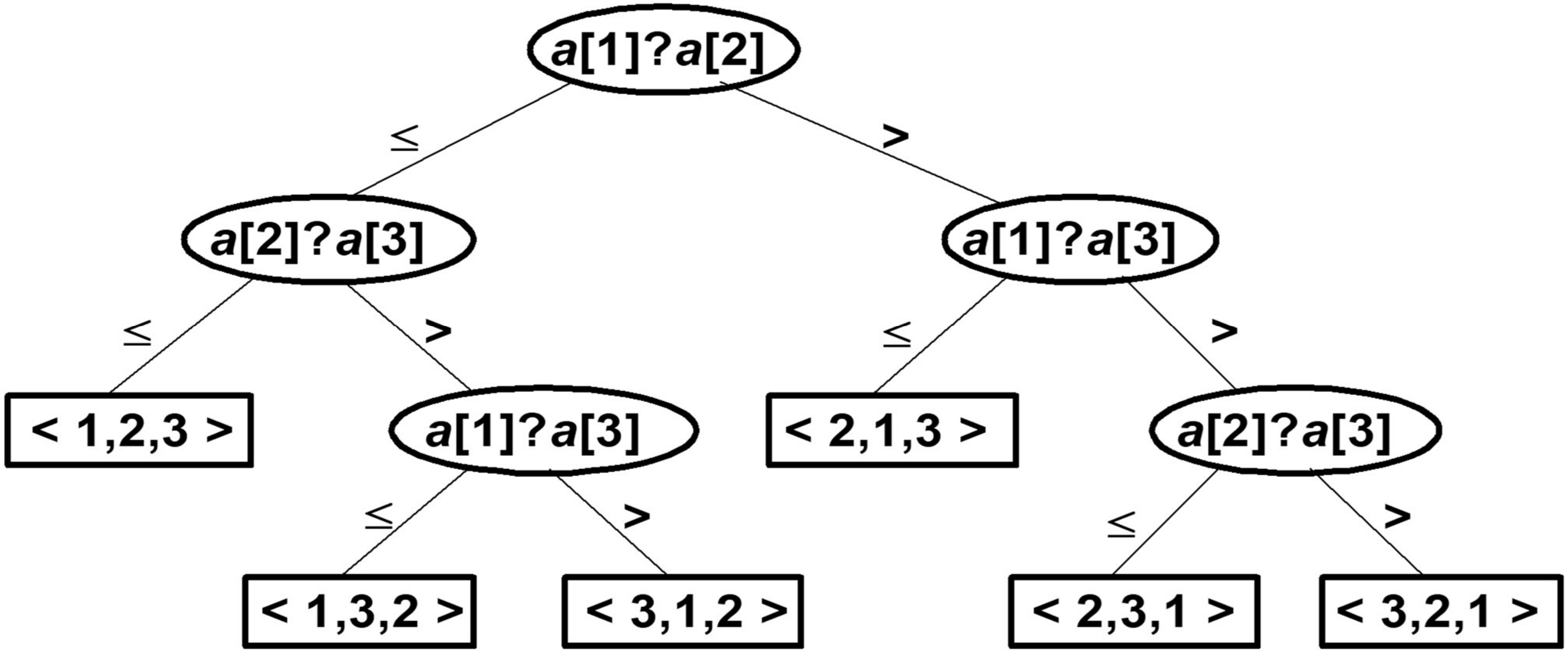
- Implementacija sa ulančanim listama
  - ✓ nema potrebe za dodatnim nizom
  - ✓ izbegava premeštanje

- Performanse
  - ✓ broj prolaza $\lceil \log n \rceil \Rightarrow O(n \log n)$
  - ✓ garantovane performanse

Metodi spajanja



Performanse



Stablo odlučivanja

Performanse

- Stablo odlučivanja
  - ✓ čvorovi predstavljaju poređenja
  - ✓ listovi predstavljaju moguće sortirane poretke
  - ✓ visina stabla predstavlja najgori slučaj

$$l = 2^h = n!$$
$$n! > (n/e)^n$$
$$h \geq \log n! > (n/e)^n = n \log n - n \log e \Rightarrow O(n \log n)$$

- Garantovane perfomanse u najgorem slučaju ne mogu biti bolje od O(n log n)

- Prosečna performansa – PE/e $\Rightarrow$ O(n log n)

Metodi linearne složenosti

- Operacije sa više ishoda
- Određene pretpostavke o ključevima omogućavaju metode linearne složenosti
- **Sortiranje brojanjem**
- Celobrojni ključevi u opsegu 1..k
- Za svaki ključ se odredi broj manjih i jednakih ključeva
- Stabilan metod
- Složenost – $O(n + k) \Rightarrow O(n)$

Sortiranje brojanjem

```
COUNTING-SORT(a)
for i = 1 to k do
      C[i] = 0
end_for
for j = 1 to n do
      C[a[j]] = C[a[j]] + 1
end_for
for i = 2 to k do
      C[i] = C[i] +C[i - 1]
end_for
for j = n downto 1 do
      B[C[a[j]]] = a[j]
      C[a[j]] = C[a[j]] –1
end_for
```

Sortiranje brojanjem

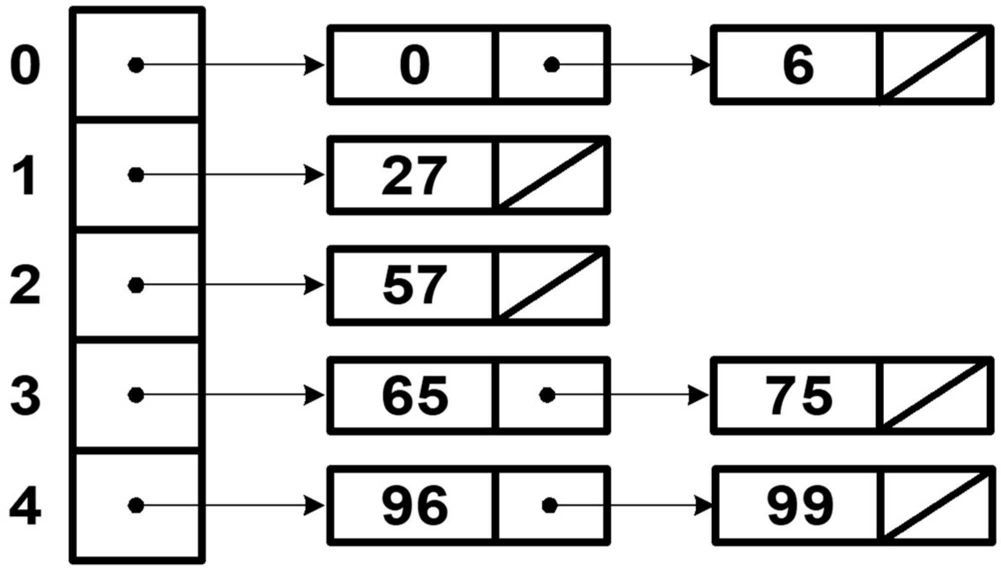
a)	*A*	1	3	5	1	3	2	1		*C*	3	1	2	0	1
b)										*C*	3	4	6	6	7
c)	*B*			1						*C*	2	4	6	6	7
d)	*B*			1	2					*C*	2	3	6	6	7
e)	*B*			1	2		3			*C*	2	3	5	6	7
f)	*B*		1	1	2		3			*C*	1	3	5	6	7
g)	*B*		1	1	2		3	5		*C*	1	3	5	6	6
h)	*B*		1	1	2	3	3	5		*C*	1	3	4	6	6
i)	*B*	1	1	1	2	3	3	5		*C*	0	3	4	6	6

Adresno sortiranje

- Varijanta umetanja, slično heširanju
- Funkcija f razvrstava ključeve u m klasa ekvivalencije
- Klasa ekvivalencije – uređena ulančana lista
- $x \leq y \Rightarrow f(x) \leq f(y)$
- Na kraju, liste se objedine
- Performanse
  - ✓ m blisko n, male liste $\Rightarrow O(n)$
  - ✓ dodatni prostor

Adresno sortiranje

$f = K\,/20$



Radixsort

- Poziciona reprezentacija ključa (znakovi)
- Razdvajanje u redove na osnovu pojedinog znaka
- Počinje se sa najmlađim znakom
- Stabilan metod
- Implementacija sa listama
- Složenost O(kn), k – broj znakova ključa
  - ✓ za k = const $\Rightarrow$ O(n)
  - ✓ pogodno za predsortiranje

Radixsort

Q_0	0			Q_0	0	6
Q_1				Q_1		
Q_2				Q_2	27	
Q_3				Q_3		
Q_4				Q_4		
Q_5	65	75		Q_5	57	
Q_6	6	96		Q_6	65	
Q_7	57	27		Q_7	75	
Q_8				Q_8		
Q_9	99			Q_9	96	99

0 65 75 6 96 57 27 99 **0 6 27 57 65 75 96 99**

Statistika poretka

- Određivanje k –tog najmanjeg elementa ($1 \leq k \leq n$)

- Specijalni slučajevi
  - ✓ $k = 1 \Rightarrow$ minimum
  - ✓ $k = n \Rightarrow$ maksimum
  - ✓ $k = (n+1)/2 \Rightarrow$ srednji element

```
MINIMUM(a)
min = a [1]
for i = 2 to n do
    if (min > a [i ]) then
        min = a [i ]
    end_if
end_for
return min
```

$\Rightarrow O(n)$

Statistika poretka

- Za malo *k* – naći minimum *k* puta, heapsort, ...
- Deljenje na particije

```
FIND(a, low, high, k)
j = PARTITION(a, low, high)
i = low + k – 1
if (j = i) then
    return a[j]
end_if
if (j > i) then
    return FIND(a, low, j - 1, k)
else
    return FIND(a, j + 1, high, k - j)
end_if
```

Statistika poretka

- Srednji slučaj – O(*n*), najgori slučaj – O(n^2)

- Poboljšani algoritam linearne složenosti
  - ✓ deli niz na $\lfloor n/5 \rfloor$ grupa po 5 elemenata
  - ✓ nalazi srednji element za svaku grupu
  - ✓ pozivom FIND nalazi srednji element *m* od srednjih elemenata grupa
  - ✓ podeli ulazni niz na dve particije oko srednjeg elementa *m* kao pivota (pozicija *j*)
  - ✓ ako je $k = j$ traženi element je $a[j]$,
    ako je $k < j$ poziva FIND(1, j - 1, k),
    ako je $k > j$ poziva FIND(j + 1, n, k - j)

Spoljašnje sortiranje

- Sortiranje podataka na diskovima – datoteka
- Specifičnosti spoljašnjih medijuma
- Sotirana sekvenca zapisa – ran
- Princip spoljašnjeg sortiranja:
  - ✓ podela datoteke
  - ✓ formiranje manjih ranova
  - ✓ progresivno povećavanje ranova spajanjem
- Osnovni postupak – dvoulazno spajanje
- Optimizacije

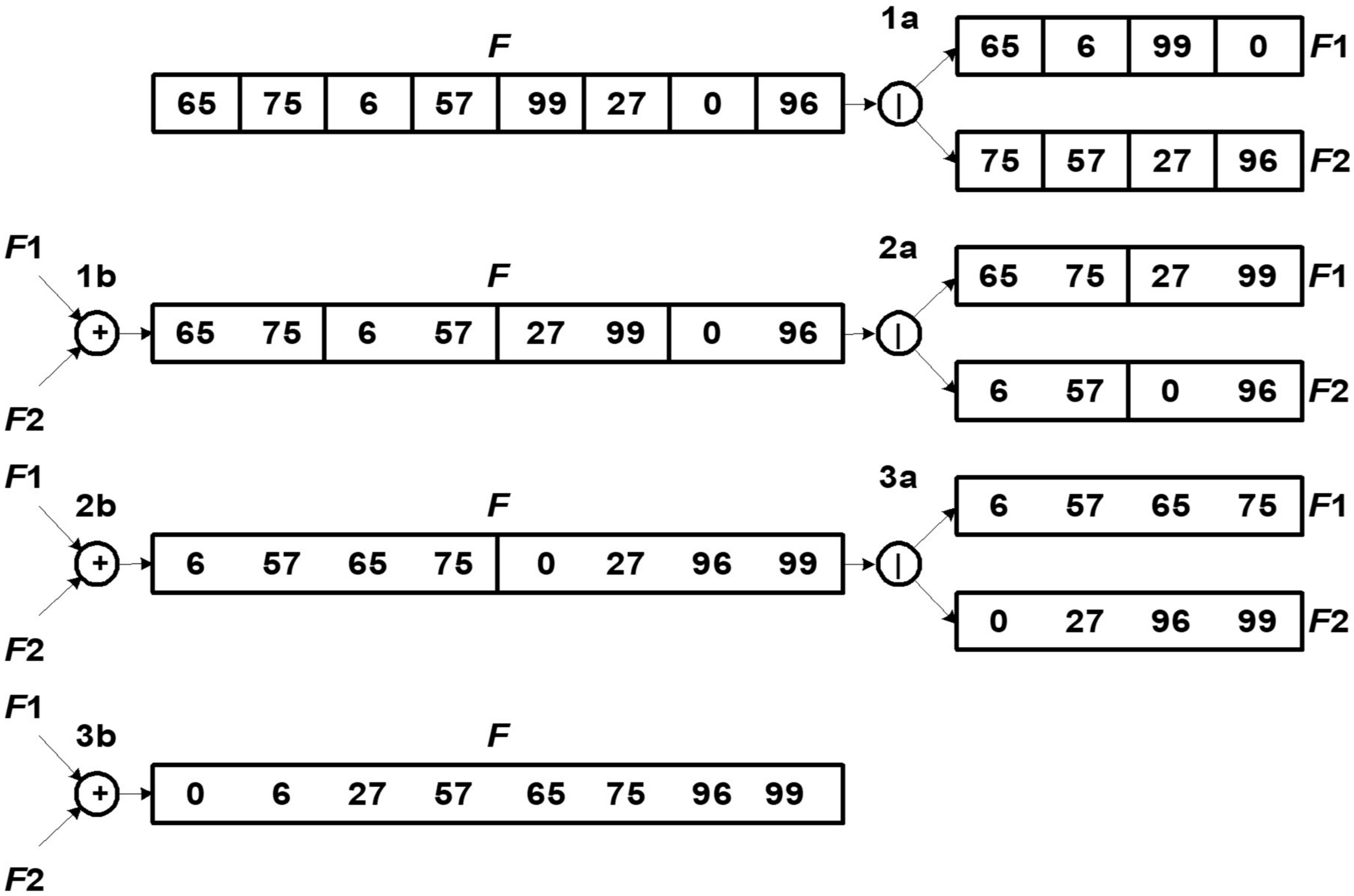
Spoljašnje sortiranje

		18 57	*a*
		20 30 65 99	*b*
c	18	57	*a*
		20 30 65 99	*b*
c	18 20	57	*a*
		30 65 99	*b*
c	18 20 30	57	*a*
		65 99	*b*
c	18 20 30 57	-	*a*
		65 99	*b*
c	18 20 30 57 65	-	*a*
		99	*b*
c	18 20 30 57 65 99		

Direktno spajanje

- Nebalansirano direktno spajanje
  - ✓ polazi od ranova sa po jednim zapisom
  - ✓ progresivno udvostručava ranove spajanjem u svakom prolazu
  - ✓ dve ulazne i jedna izlazna datoteka
- Prolaz
  - ✓ faza podele
  - ✓ faza spajanja
- Faza podele ne doprinosi sortiranju

Direktno spajanje



Balansirano direktno spajanje

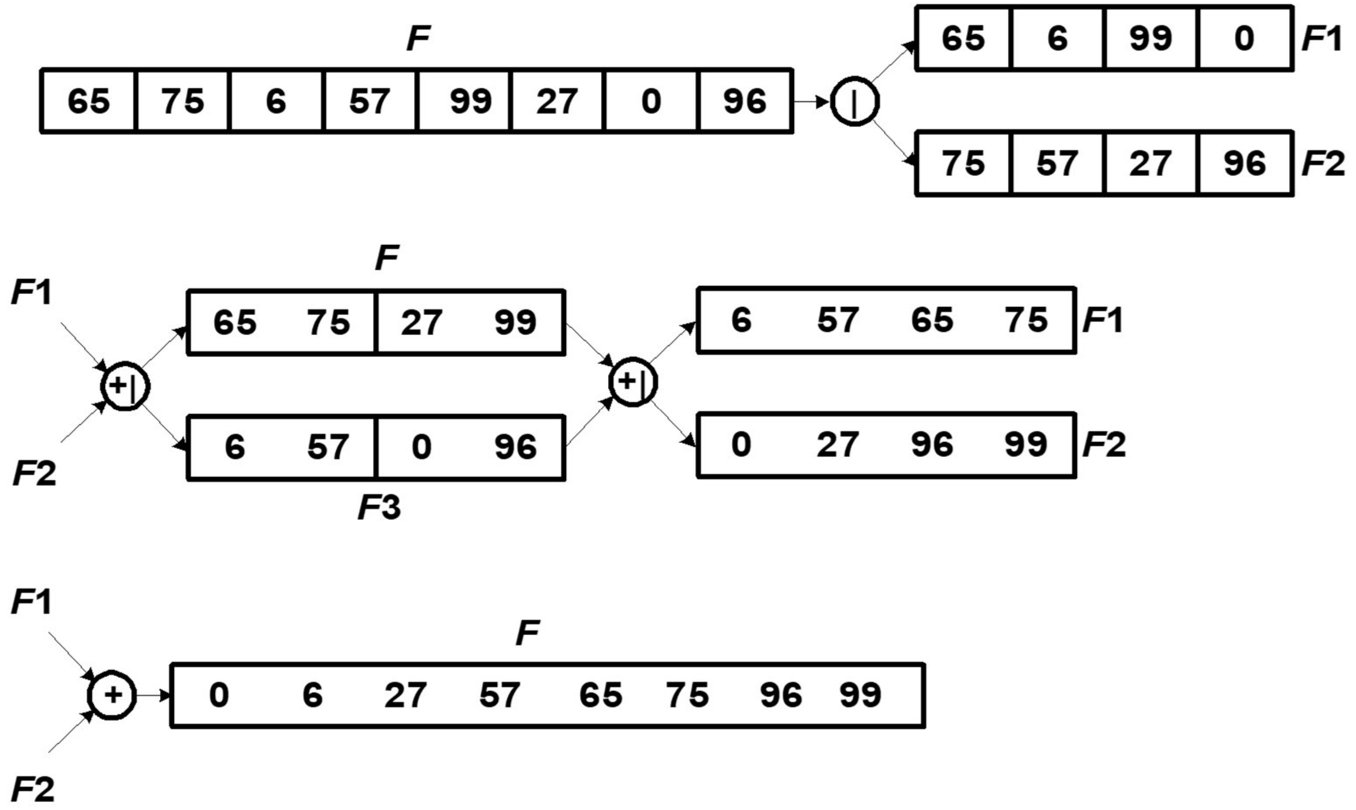
- Dve ulazne i dve izlazne datoteke

- Podela se izbegava naizmeničnim slanjem u izlazne datoteke

- Alternacija ulaznih i izlaznih datoteka

- Performanse
  - ✓ broj prolaza - O(log *n*)
  - ✓ broj kopiranja - O(*n*)
  - ✓ fiksna složenost - O(*n* log *n*)
  - ✓ optimizacija – duži početni ranovi
  - ✓ mana – fiksna dužina ranova

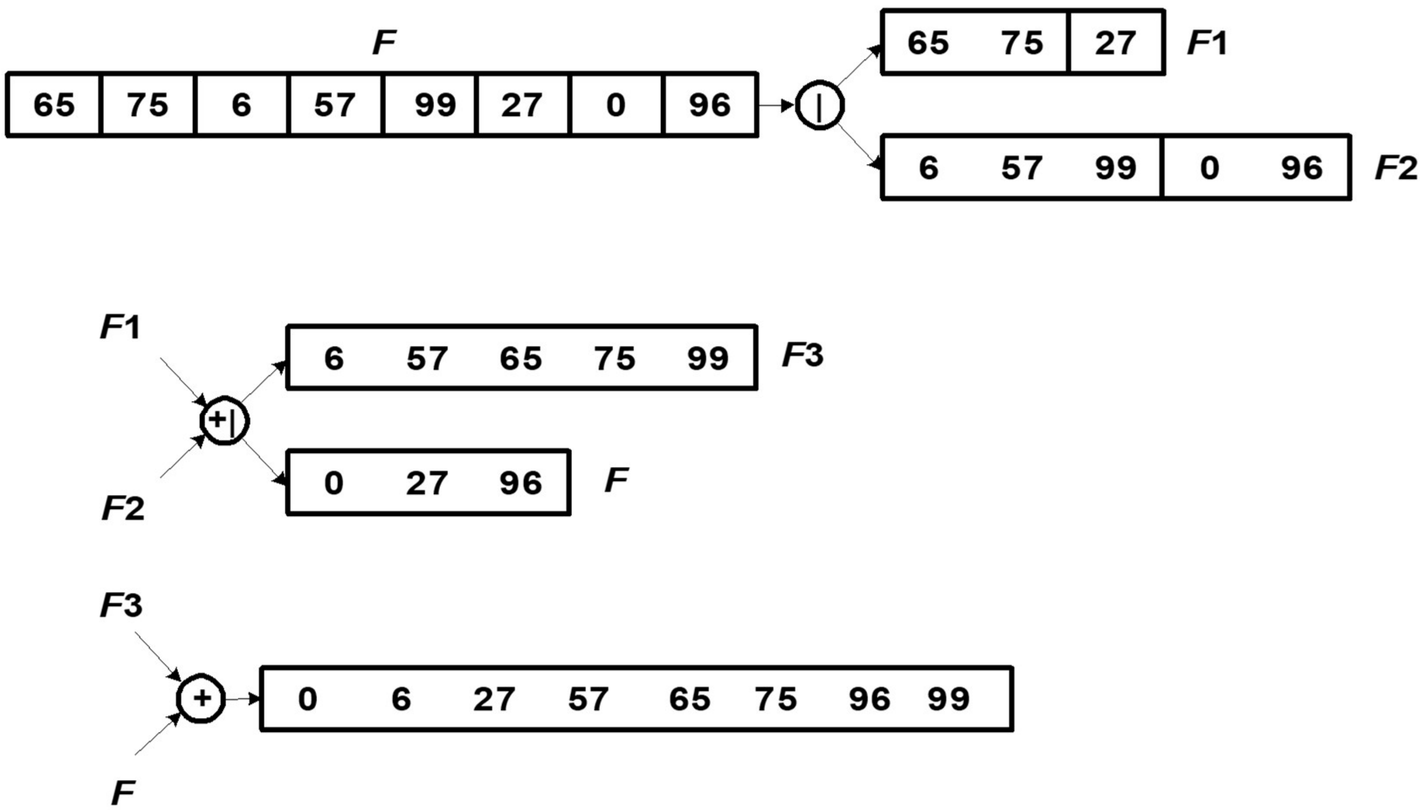
Balansirano direktno spajanje



Prirodno spajanje

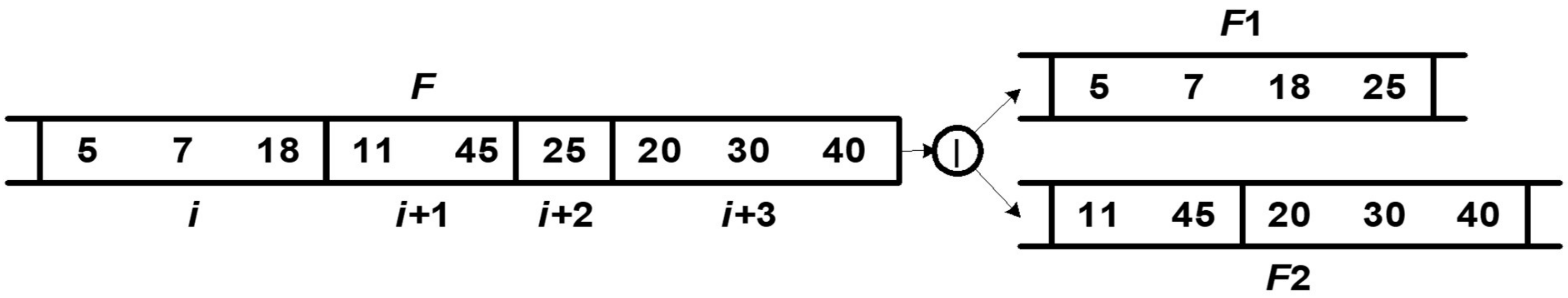
- Podela datoteke na uređene sekvence

- Adaptivno određivanje ranova



Prirodno spajanje

- Mogućnost rekombinacije ranova
- Dodatno smanjivanje broja ranova
- Performanse
  - ✓ zavise od prethodne uređenosti datoteke
  - ✓ dodatna poređenja



Višestruko spajanje

- Performanse zavise od broja prolaza, a broj prolaza od broja ranova

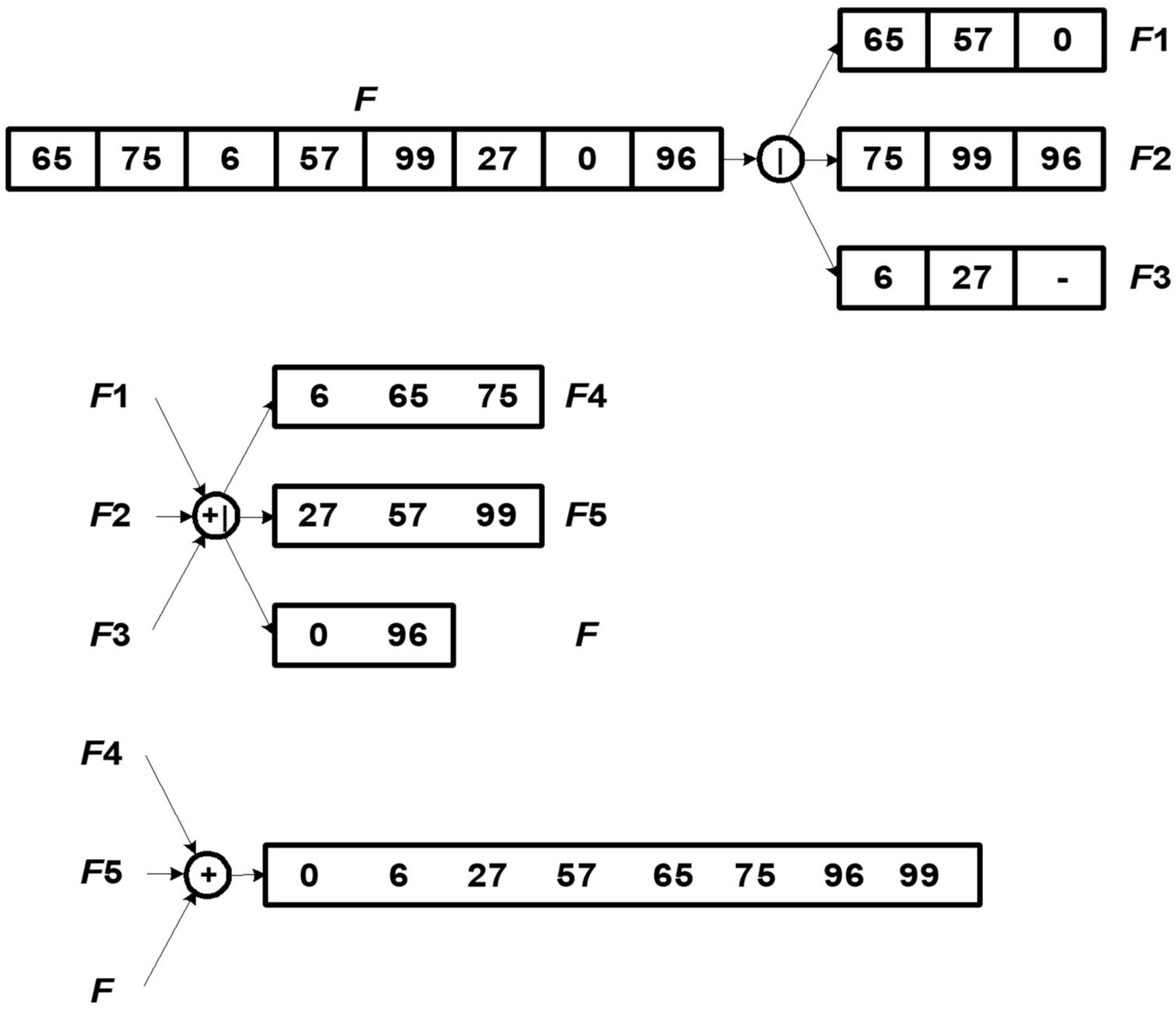
- Dodatno smanjivanje broja ranova višestrukm spajanjem

- Nebalansirano *m*-tostruko spajanje
  - ✓ *m* ulaznih i jedna izlazna datoteka

- Nebalansirano *m*-tostruko spajanje
  - ✓ *m* ulaznih i *m* izlaznih datoteka

- Performanse – $O(\log_m n)$

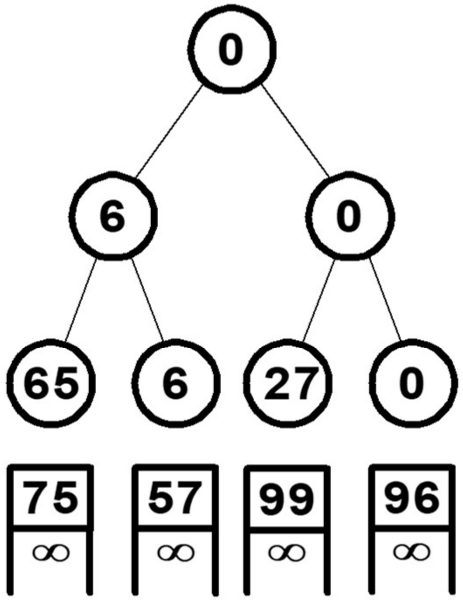
Višestruko spajanje



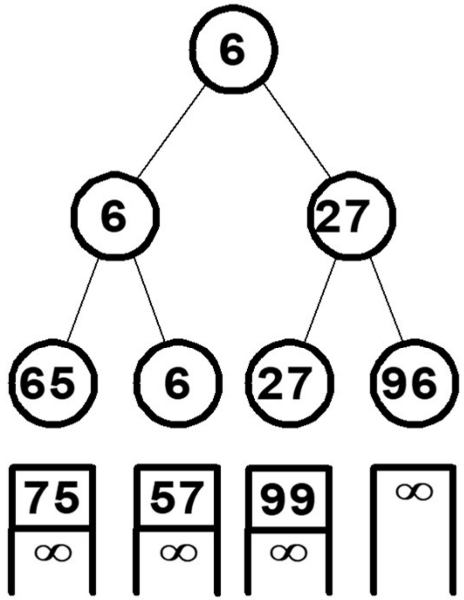
Višestruko spajanje

- Selekcija sa zamenom

- Stablo selekcije



a)



b)

Polifazno spajanje

faza	*F*1	*F*2	*F*3	broj zapisa
1	21(1)	13(1)	-	34
2	8(1)	-	13(2)	26
3	-	8(3)	5(2)	24
4	5(5)	3(3)	-	25
5	2(5)	-	3(8)	24
6	-	2(13)	1(8)	26
7	1(21)	1(13)	-	21
8	-	-	1(34)	34