



# ALGORITMI I STRUKTURE PODATAKA

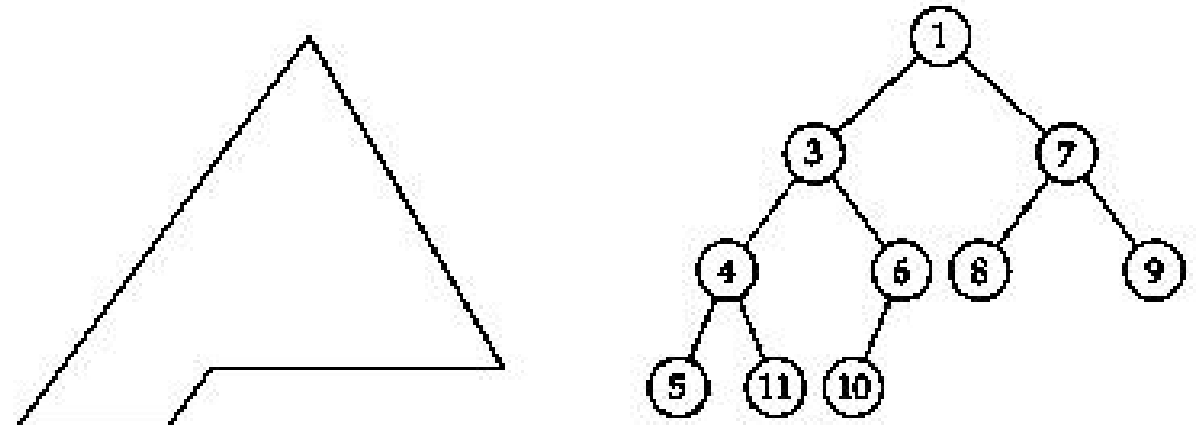
RAČUNSKÉ VEŽBE – TERMIN BR. 12 – BINARNI HIP; ALGORITMI SORTIRANJA

ALDINA AVDIĆ, DIPL. INŽ. – [apl.jaskovic@np.ac.rs](mailto:apl.jaskovic@np.ac.rs)

RAČUNARSKA TEHNIKA, SOFTVERSKO INŽENJERSTVO, INFORMATIKA I MATEMATIKA

# Binarni hip

- × Min-hip
- × Max-hip
- × Binarni hip (binary heap) je hip struktura podataka organizovana po principu binarnog stabla. Može se posmatrati kao binarno stablo sa dva dodatna ograničenja:
  - × **Svojstvo oblika:** Stablo je kompletno binarno stablo ako svi nivoi stabla, osim možda poslednjeg, u potpunosti popunjeni, a u slučaju da poslednji nivo stabla nije popunjen, čvorovi tog nivoa se popunjavaju s leva na desno.
  - × **Svojstvo hipa:** Svi čvorovi su ili “veći ili jednaki” ili “manji ili jednaki” od svakog čvora koji mu je dete u zavisnosti od toga koja funkcija za poređenje se koristi u implementaciji.
- × Stablo sa funkcijom za poređenje “veće ili jednako” ( $\geq$ ) se zove **max-heap**, a sa funkcijom za poređenje “manje ili jednako” ( $\leq$ ) se zove **min-heap**.



Slika 6.21: Kompletno binarno stablo.

# Uklanjanje najmanjeg čvora iz hipa

## Listing 6.10: Uklanjanje čvora iz hipa

```
// Ulaz: hip predstavljen nizom  $a$   
// Izlaz: hip bez uklonjenog korena i ključ uklonjenog korena
```

```
algorithm heap-deletemin( $a$ )
```

```
     $x = a[1];$   
     $a[1] = a[n];$   
     $n = n - 1;$   
    heap-siftdown( $a, 1$ );
```

```
    return  $x$ ;
```

## Listing 6.9: Prosejavanje čvora u hipu

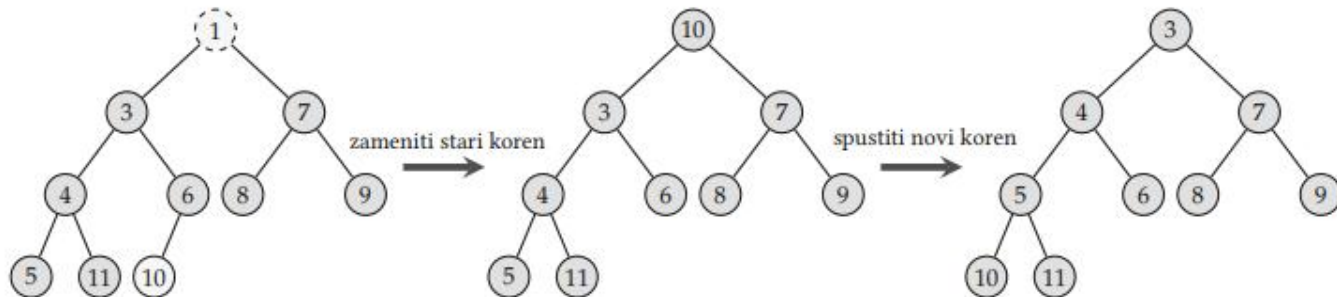
```
// Ulaz: indeks čvora  $a_i$  u hipu predstavljen nizom  $a$   
// Izlaz: čvor  $a_i$  pomeren nadole do svog pravog mesta u hipu
```

```
algorithm heap-siftdown( $a, i$ )
```

```
     $j = 2 * i;$  //  $a_j$  je levo dete čvora  $a_i$   
    if ( $(j < n) \ \&\& \ (a[j+1] < a[j])$ ) then  
         $j = j + 1;$ 
```

```
    //  $a_j$  je manje dete čvora  $a_i$   
    if ( $(j \leq n) \ \&\& \ (a[i] > a[j])$ ) then  
        swap( $a[i], a[j]$ );  
        heap-siftdown( $a, j$ );
```

```
    return;
```



# Dodavaje novog čvora u hip

## Listing 6.8: Dodavanje čvora u hip

```
// Ulaz: hip predstavljen nizom  $a$ , čvor  $x$   
// Izlaz: hip proširen za čvor  $x$   
algorithm heap-insert( $a, x$ )
```

```
     $n = n + 1;$   
     $a[n] = x;$   
    heap-bubbleup( $a, n$ );
```

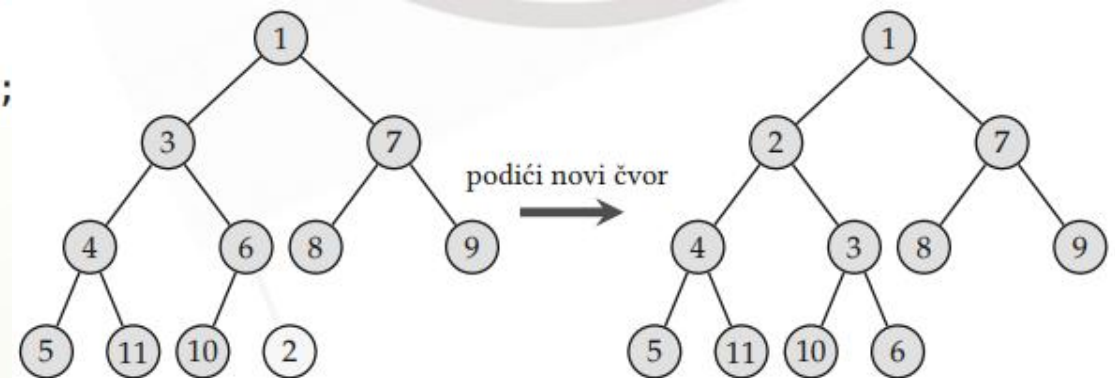
```
    return;
```

## Listing 6.7: Isplivavanje čvora u hipu

```
// Ulaz: indeks čvora  $a_i$  u hipu predstavljen nizom  $a$   
// Izlaz: čvor  $a_i$  pomeren naviše do svog pravog mesta u hipu  
algorithm heap-bubbleup( $a, i$ )
```

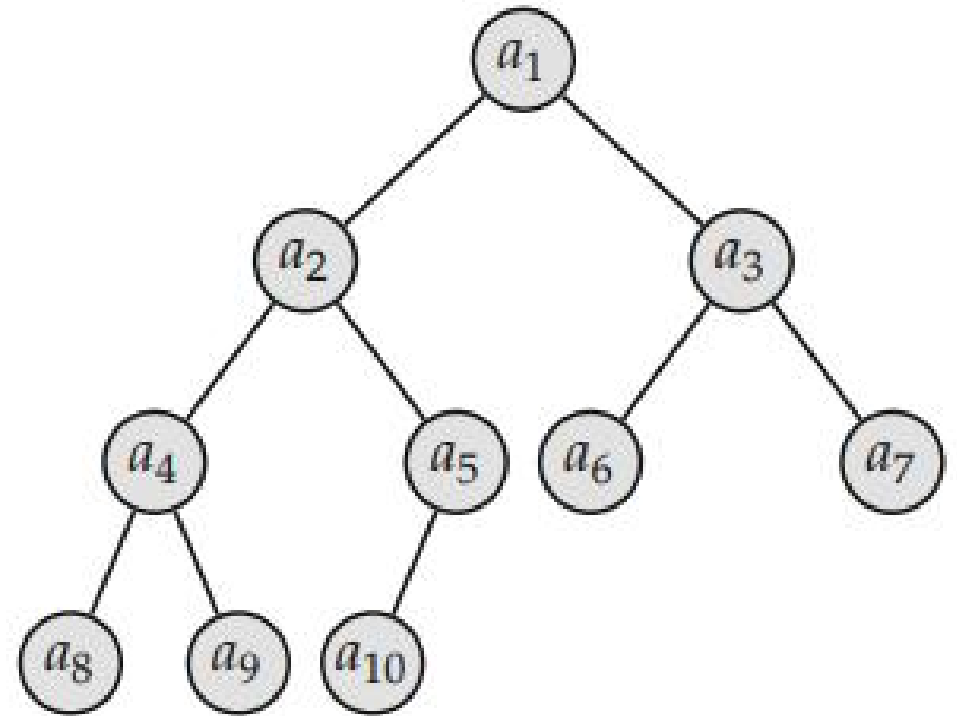
```
     $j = i / 2;$  //  $a_j$  je roditelj čvora  $a_i$   
    if (( $j > 0$ ) && ( $a[i] < a[j]$ )) then  
        swap( $a[i], a[j]$ );  
        heap-bubbleup( $a, j$ );
```

```
    return;
```



# Predstavljanje hipa preko niza

- × Levi potomak čvora  $a[i]$  je  $a[2*i]$
- × Desni  $a[2*i+1]$
- × Roditelj čvora  $a[i]$  je  $a[i/2]$ , z



# Konstruisanje hipa

## Listing 6.11: Konstruisanje hipa

```
// Ulaz: niz  $a$  od  $n$  neuređenih elemenata  
// Izlaz: preuređen niz  $a$  tako da predstavlja binarni hip  
algorithm heap-make( $a$ )  
  for  $i = n/2$  downto 1 do  
    heap-siftdown( $a, i$ );  
  
return;
```



# Heap sort

```
1. build a heap      1, 2, 4, 5, 3, 6
2. turn this heap into a sorted list


deleteMin
1, 2, 4, 5, 3, 6      swap 1 and 6
6, 2, 4, 5, 3,      1      restore heap
2, 6, 4, 5, 3,      1
2, 3, 4, 5, 6,      1

deleteMin
2, 3, 4, 5, 6,      1      swap 2 and 6
6, 3, 4, 5,      2, 1      restore heap
3, 6, 4, 5,      2, 1
3, 5, 4, 6,      2, 1

deleteMin
3, 5, 4, 6,      2, 1      swap 3 and 6
6, 5, 4,      3, 2, 1      restore heap
4, 5, 6,      3, 2, 1      restore heap

deleteMin
4, 5, 6,      3, 2, 1      swap 4 and 6
6, 5,      4, 3, 2, 1      restore heap
5, 6,      4, 3, 2, 1

deleteMin
6,      5, 4, 3, 2, 1
```



# Zadatak 1

Precizno objasniti postupak rada algoritma sortiranja *Heapsort*.

Usvojiti pogodnu memorijsku reprezentaciju  
i navesti njene prednosti.

Za usvojenu strukturu demonstrirati rad algoritma  
po koracima pri sortiranju niza:

57 42 69 11 35 28 7 19

u neopadajućem poretku.

Grubo izvesti performanse u najgorem i prosečnom slučaju.

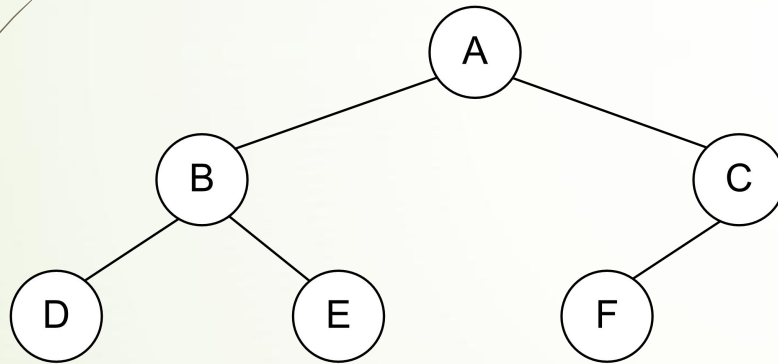


# Zadatak 1 - Rešenje

- × *Heapsort* : efikasan algoritam za sortiranje, zasnovan na strukturi podataka zvanoj *heap* (eng. gomila)
- × *Heap* : vrsta binarnog stabla kod kojeg je vrednost u roditeljskom čvoru veća od vrednosti smeštenim u sinove - najveća vrednost je smeštena u koren stabla
- × Prednost u odnosu na stablo selekcije:
  - × nije potreban dodatni memorijski prostor za konstrukciju stabla, jer se sve izmene vrše nad originalnim nizom
  - × nema replikacije ključeva, a binarno stablo može da se formira nad osnovnim nizom čime se postiže efikasnija implementacija
  - × prilikom sortiranja ne vrše se ponekad nepotrebna ažuriranja i poređenja

# Zadatak 1 – Rešenje

- × Smeštanje skoro potpunog binarnog stabla u niz:
  - × indeks roditeljskog čvora od čvora indeksa A je  $A \div 2$
  - × indeks levog potomka od čvora indeksa A je  $2*A$
  - × indeks desnog potomka od čvora indeksa A je  $2*A+1$
  - × koren stabla ima najmanji indeks



1	2	3	4	5	6
A	B	C	D	E	F

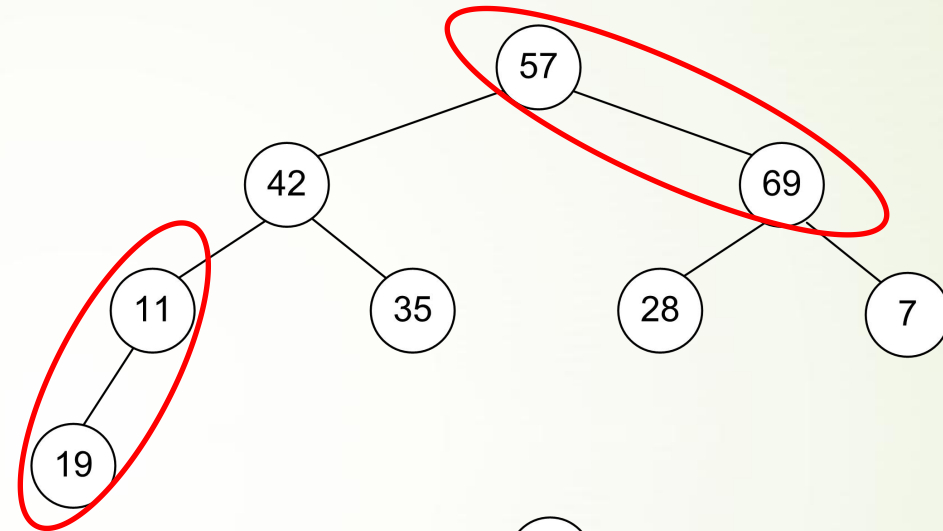
## Zadatak 1 - Rešenje

- × Neuređen niz se najpre preuredi u heap
  - novi ključ se dodaje na kraj heap-a. Sve dok je veći od svog oca (koji je već u heap-u), vrši se zamena sa ocem.
- × Sve dok se ne obradi svih  $n$  elemenata
- × vrednost u korenu menja mesto sa poslednjim elementom nesortiranog dela niza (onim delom koji je još uvek u heap-u)
- × nova vrednost u korenu se propagira niz stablo sve dok ne uspostavi odnos roditelj-potomak koji važi u heap-
- × Ovako dobijen niz je uređen neopadajuće
  
- × Vremenska složenost  $O(n \log n)$
- × najbolji, najgori i prosečan slučaj se razlikuju za multiplikativnu konstantu
- × Algoritam nije stabilan

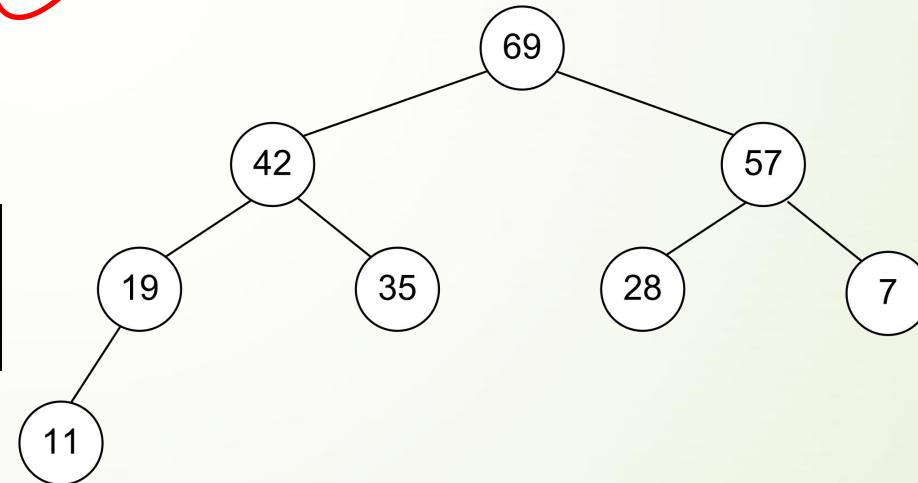
# Zadatak 1 - Rešenje

× Preuređivanje niza u heap: 57 42 69 11 35 28 7 19

1	2	3	4	5	6	7	8
57	42	69	11	35	28	7	19



1	2	3	4	5	6	7	8
<b>69</b>	42	<b>57</b>	<b>19</b>	35	28	7	<b>11</b>

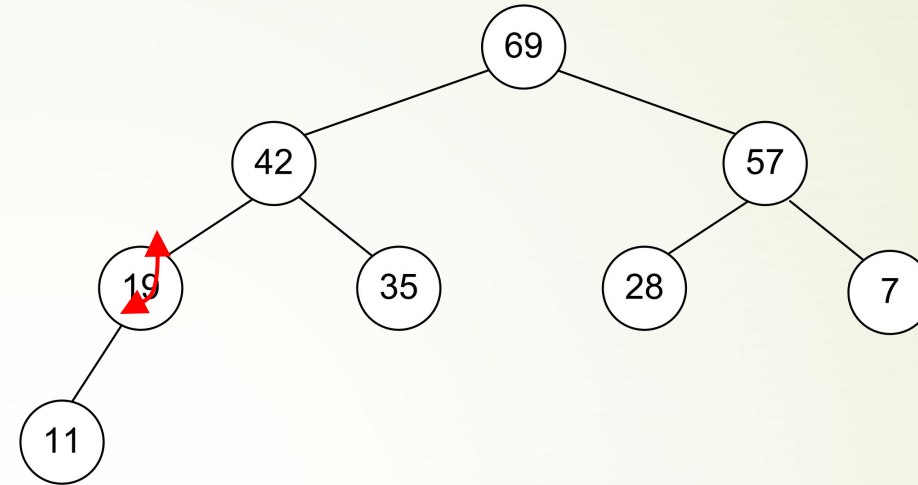


# Zadatak 1 - Rešenje

× Faza selekcije

1	2	3	4	5	6	7	8
69	42	57	19	35	28	7	11

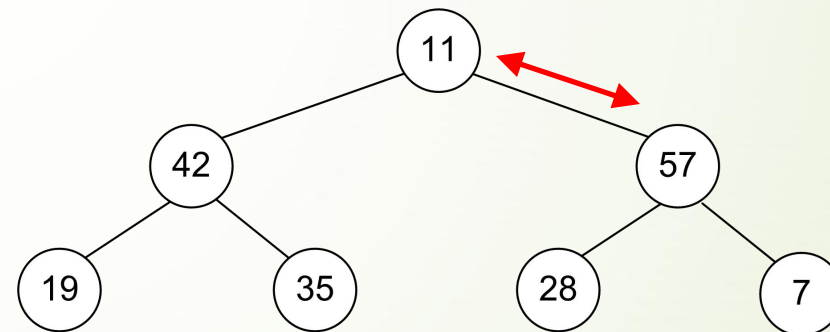
p



Najpre se element u korenu stabla zameni sa poslednjim elementom

1	2	3	4	5	6	7	8
11	42	57	19	35	28	7	<b>69</b>

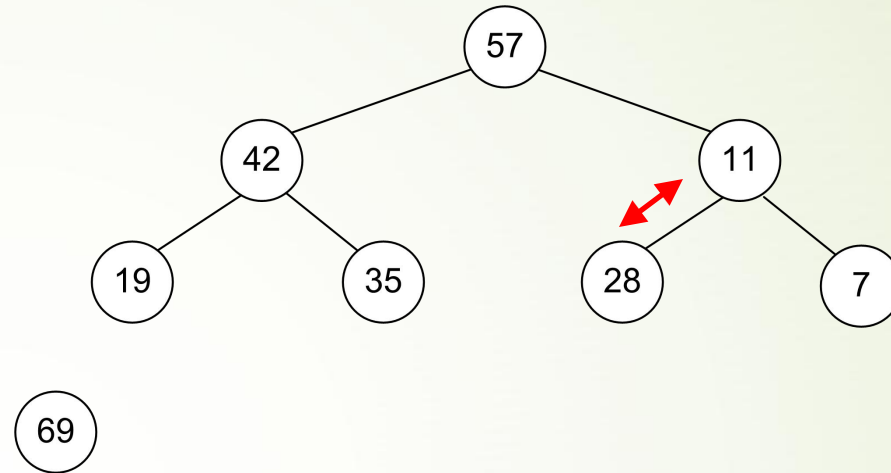
p



# Zadatak 1 - Rešenje

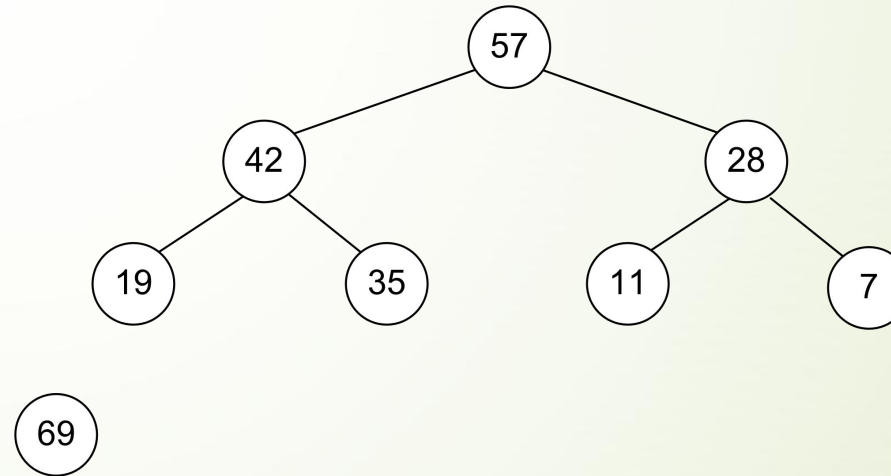
1	2	3	4	5	6	7	8
57	42	11	19	35	28	7	69

p



1	2	3	4	5	6	7	8
57	42	28	19	35	11	7	69

p

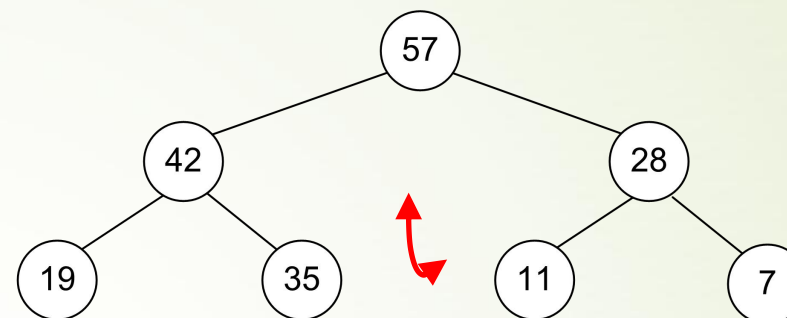




# Zadatak 1 - Rešenje

1	2	3	4	5	6	7	8
57	42	28	19	35	11	7	<b>69</b>

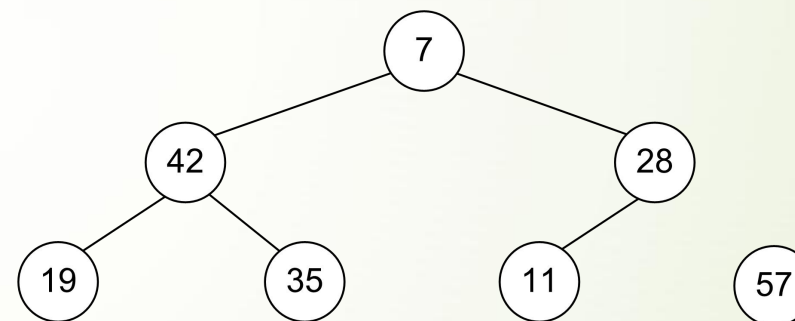
p



69

1	2	3	4	5	6	7	8
7	42	28	19	35	11	<b>57</b>	<b>69</b>

p



69

# Zadatak 1 - Rešenje

1	2	3	4	5	6	7	8
42	35	28	19	7	11	57	69

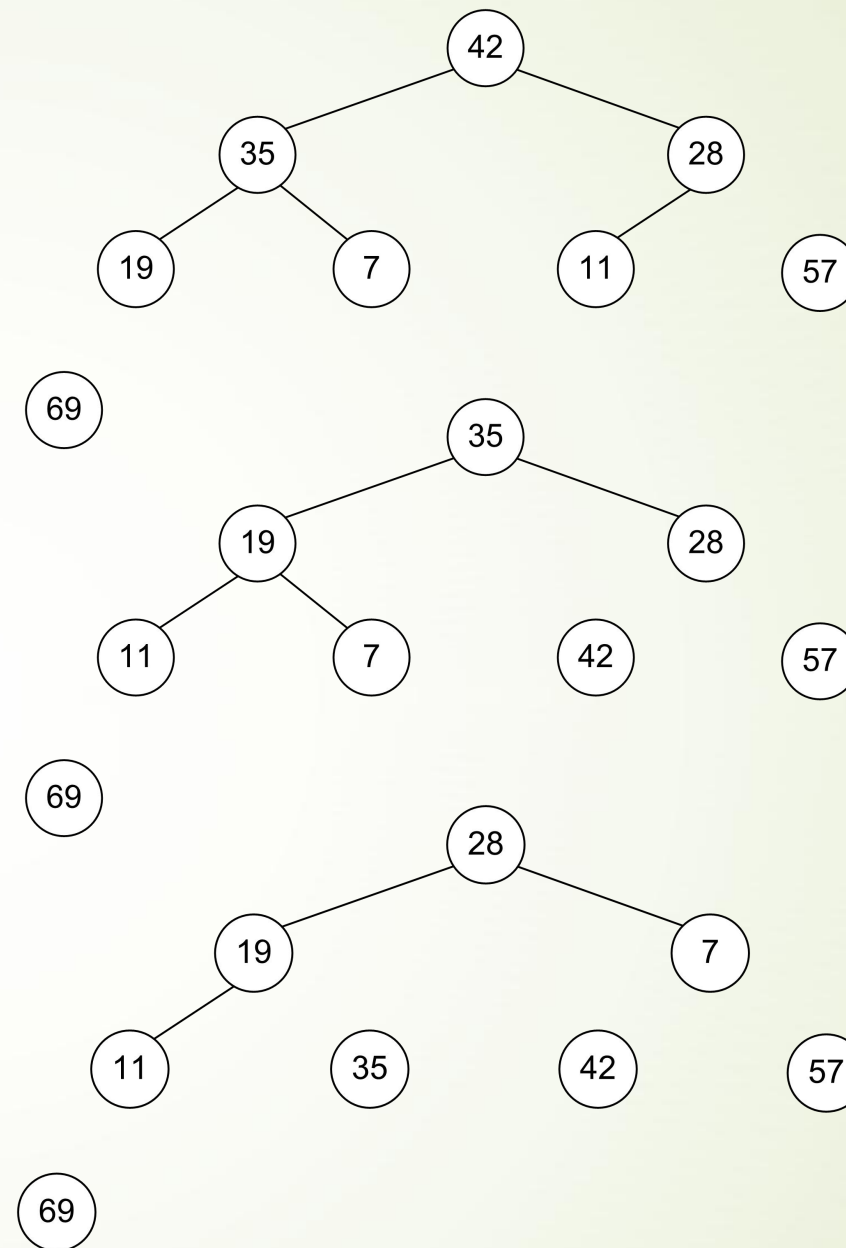
p

1	2	3	4	5	6	7	8
35	19	28	11	7	42	57	69

p

1	2	3	4	5	6	7	8
28	19	7	11	35	42	57	69

p





## Zadatak 2

Demonstrirati algoritam za direktno umetanje (*insertion sort*)  
na primeru sortiranja neuređenog niza  
3, 10, 4, 6, 8, 9, 7, 2, 1, 5

**INSERTION-SORT(*a*)**

**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**

## Zadatak 2 – Rešenje

$i=2$   
 $k=10$

3	10	4	6	8	9	7	2	1	5
j	i								

$i=3$   
 $k=4$

3	10	→ 4	6	8	9	7	2	1	5
j	i								

3	10	10	6	8	9	7	2	1	5
j		i							

3	4	10	6	8	9	7	2	1	5
j		i							

$i=4$   
 $k=6$

3	4	10	→ 6	8	9	7	2	1	5
		j	i						

$i=5$   
 $k=8$

3	4	6	10	→ 8	9	7	2	1	5
			j	i					

## Zadatak 2 - Rešenje

i=6  
k=9

3	4	6	8	10	9	7	2	1	5
				j	i				

i=7  
k=7

3	4	6	8	9	10	7	2	1	5
				j	i				

3	4	6	8	9	10	10	2	1	5
				j	i				

3	4	6	8	9	9	10	2	1	5
				j	i				

3	4	6	8	8	9	10	2	1	5
				j	i				

3	4	6	7	8	9	10	2	1	5
				j	i				



## Zadatak 2 - Rešenje

$i=8$   
 $k=2$

$j=0$

3	4	6	7	8	9	10	2	1	5
---	---	---	---	---	---	----	---	---	---

$i$

$i=9$   
 $k=1$

2	3	4	6	7	8	9	10	1	5
---	---	---	---	---	---	---	----	---	---

$i$

$i=10$   
 $k=5$

1	2	3	4	6	7	8	9	10	5
---	---	---	---	---	---	---	---	----	---

$i$

1	2	3	4	5	6	7	8	9	10
---	---	---	---	---	---	---	---	---	----

Performanse: najbolje  $O(n)$ , najgore  $O(n^2)$ , prosečno  $O(n^2)$ .  
Dobre performanse za prilično uređene nizove.



## Zadatak 3

Opisati sortiranje primenom metoda umetanja  
sa smanjenjem inkrementa (*shell sort*).

Objasniti na čemu se zasniva efikasnost ovog metoda, kako se bira sekvenca  
inkremenata  
i kolika je složenost metoda.

Ilustrovati rad algoritma pri sortiranju niza  
19, 61, 42, 31, 7, 95, 77 i 25  
u tri iteracije sa efikasnim izborom inkrementa.

## Zadatak 3 – Rešenje

```
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  $(k \geq 1)$  and  $(y < a[k])$  do  
       $a[k + inc] = a[k]$   
       $k = k - inc$   
    end_while  
     $a[k + inc] = y$   
  end_for  
end_for
```

*Shell sort* pokušava da popravi nedostatke algoritma *insertion sort*:

1. *insertion sort* najbolje radi kada je niz (gotovo) uređen
2. loše performanse duguje višestrukom pomeranju podataka za jedno mesto (korak)
3. **ideja**: napraviti više prolaza, najpre sa većom vrednošću koraka a zatim sve manjom dok se niz ne svede na situaciju koja je pogodna za *insertion sort*

## Zadatak 3 - Rešenje

Performanse zavise i od sekvence inkremenata

Generalno: sekvenca  $h_1, h_2, h_3, \dots, h_t$  proizvoljna,  
ali mora da ispuni uslove  $h_{i+1} < h_i$ ,  $h_t = 1$

Predložena optimalna sekvenca, empirijski utvrđena:

1, 4, 10, 23, 57, 132, 301, 701

(Marcin Ciura, Best Increments for the Average Case of Shellsort, 13th  
International

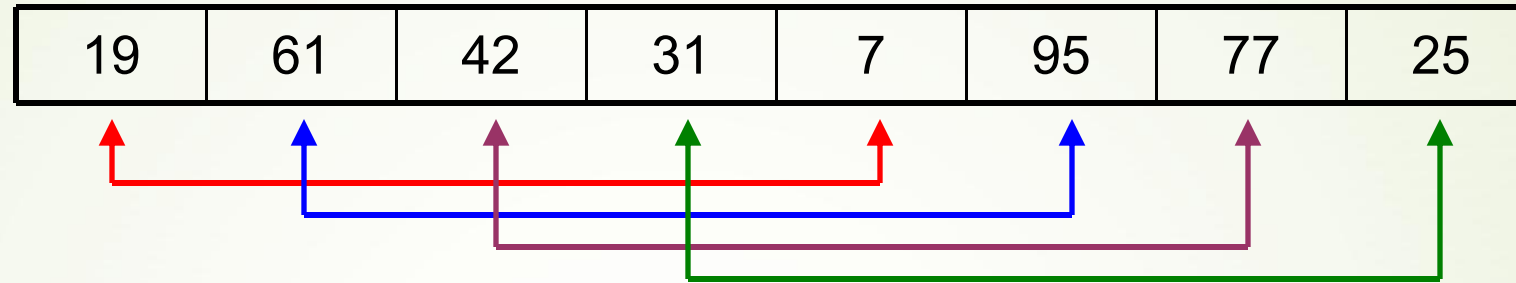
Symposium on Fundamentals of Computation Theory, 2001)

Performanse: najgore  $O(n^2)$ , prosečno  $O(n^{1.3})$ .

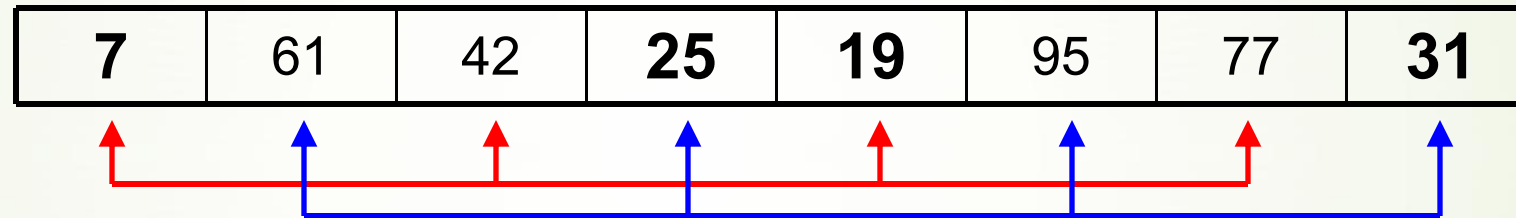
Još uvek nije dokazano da li može  $O(n \log n)$ .

## Zadatak 3 – Rešenje

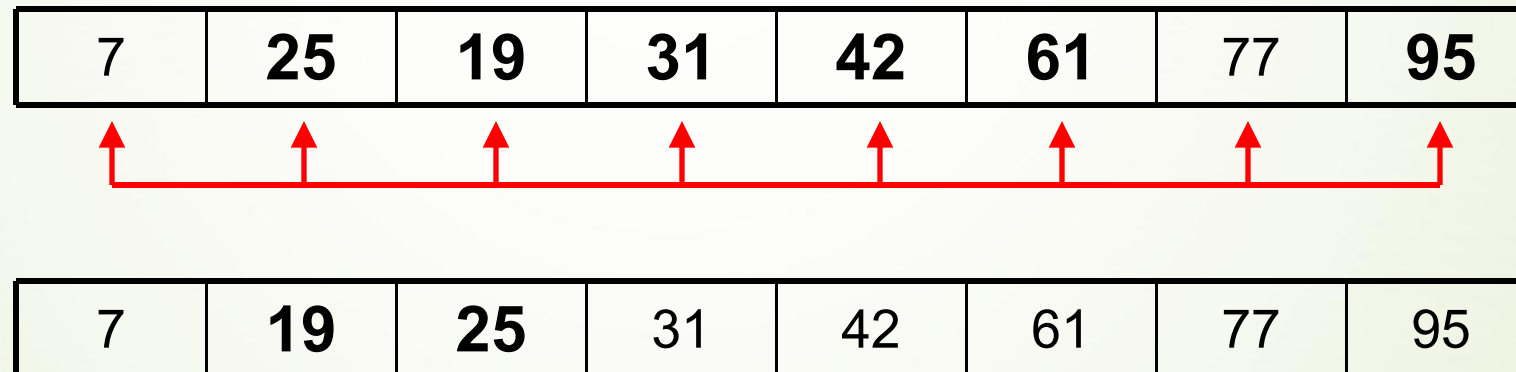
$h_1 = 4$



$h_2 = 2$



$h_3 = 1$





## Zadatak 4

Opisati sortiranje primenom metoda umetanja  
sa smanjenjem inkrementa.

Objasniti na čemu se zasniva efikasnost ovog metoda, kako se bira sekvenca  
inkremenata i kolika je složenost metoda.

Ilustrovati rad algoritma pri sortiranju niza

38, 70, 63, 94, 7, 82, 24, 11, 45, 53

za sledeće vrednosti inkremenata:

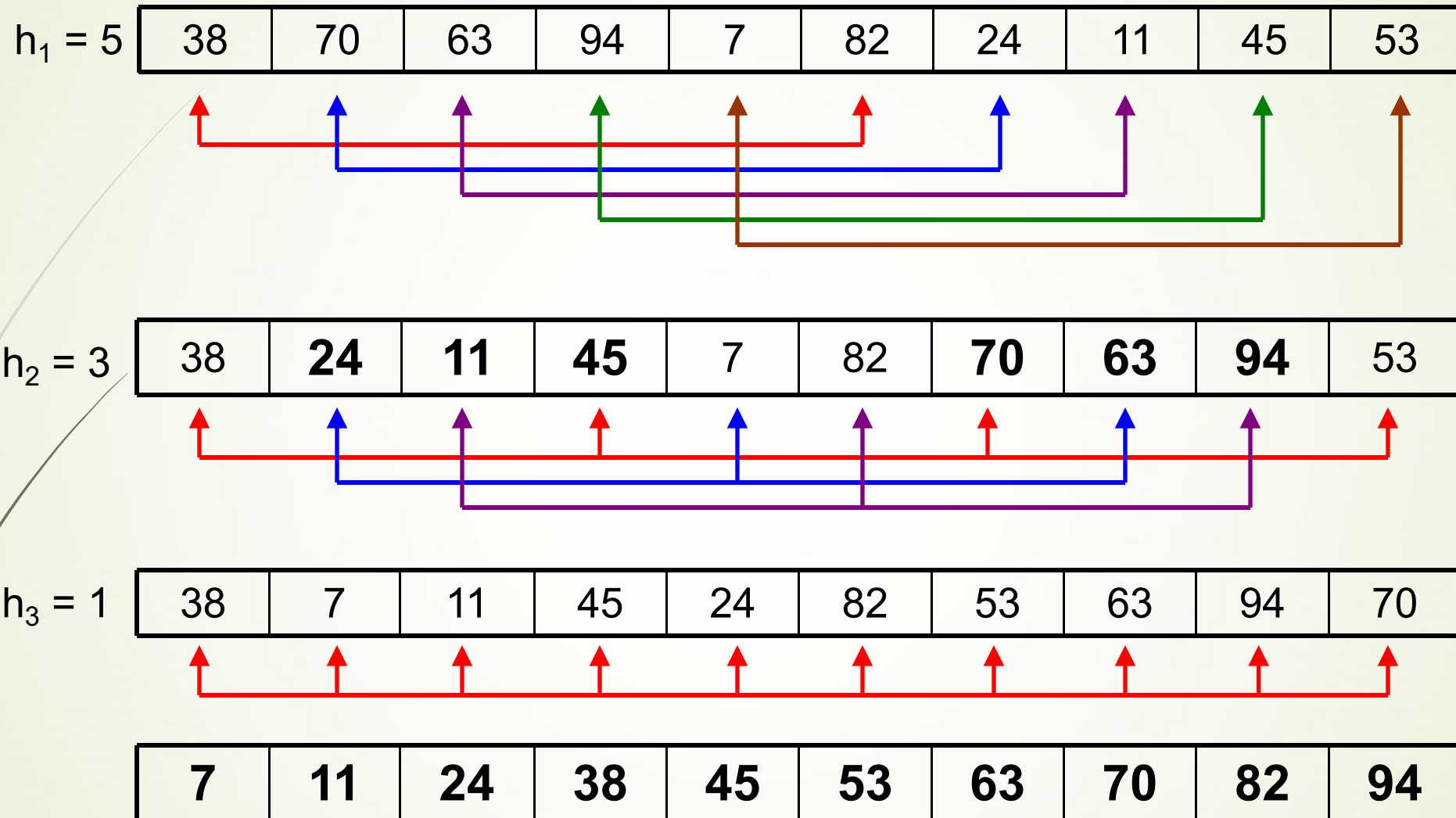
$$h_1=5$$

$$h_2=3$$

$$h_3=1$$



## Zadatak 4 - Rešenje



## Zadatak 5

Ilustrovati rad algoritma direktne selekcije (*selection sort*) pri sortiranju niza: 13, 10, 4, 6, 8, 9, 7, 2, 1, 5

```
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
```

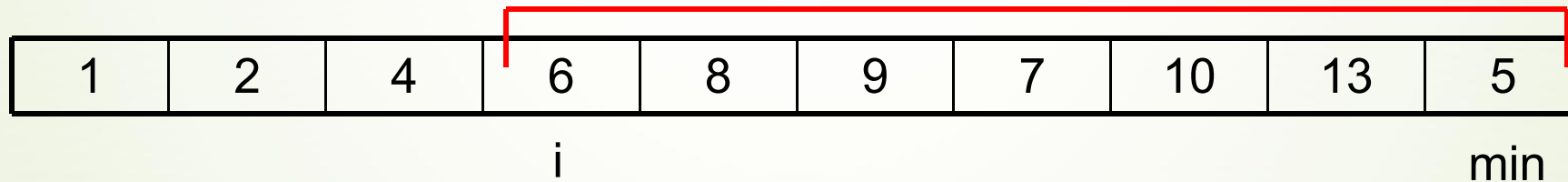
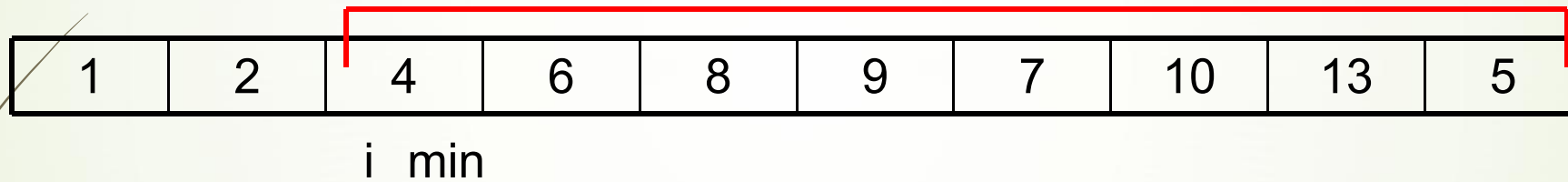
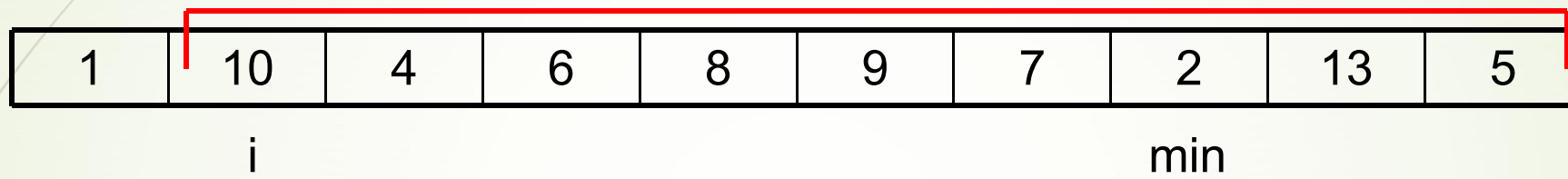
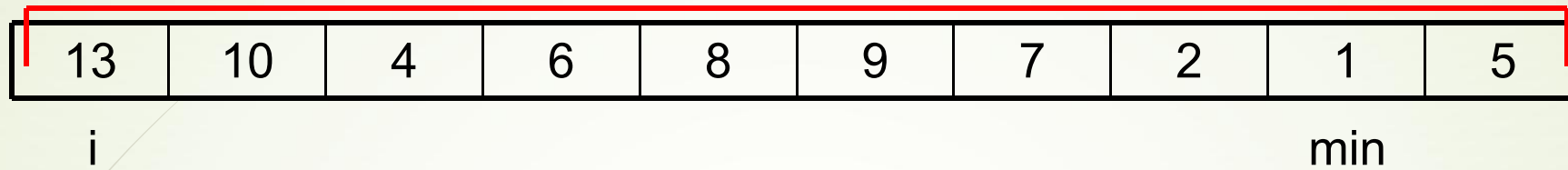
U svakom ciklusu *i*

- algoritam pronalazi element najmanje vrednosti u neobrađenom delu niza
- menja mesta uočenom minimalnom elementu sa elementom *i*

Performanse:

- najgori slučaj  $O(n^2)$
- pretraga ne zavisi od uređenosti niza, pa je prosečna i najbolja takođe  $O(n^2)$
- za razliku od INSERTION-SORT-a, algoritam zahteva da ceo niz bude u dostupan pre početka izvršavanja

## Zadatak 5 - Rešenje



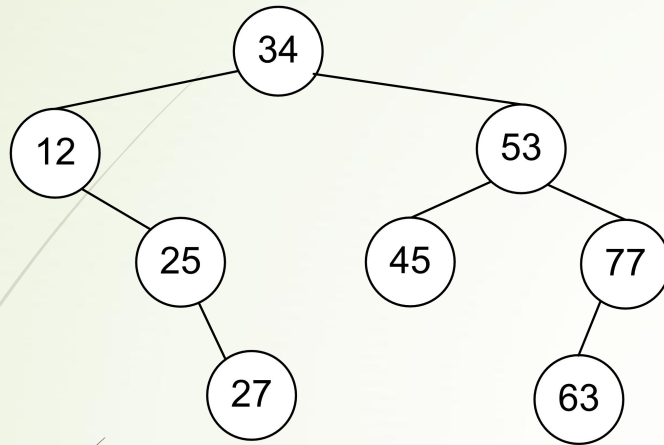
## Zadatak 6

Ilustrovati postupak sortiranja korišćenjem stabla binarnog pretraživanja na primeru sortiranja niza ključeva 34, 53, 45, 77, 63, 12, 25 i 27.

Kako se rešava problem istih ključeva?

Koja je složenost ovog postupka?

## Zadatak 6 – Rešenje



Ključevi se umeću u stablo redom u kojem su dati u ulaznom nizu.

*INORDER* obilazak stabla daje ključeve u neopadajućem poretku.


Problem istih ključeva se rešava:

- umetanjem ponovljenog ključa u desno podstablo (zašto?)
- ulančavanjem ključeva u datom čvoru

**Složenost:** najgora  $O(n^2)$ , najbolja  $O(n \log n)$ , prosečna  $O(n \log n)$

**Loše osobine:**

1. potrebna dodatna memorija
2. loš za pretežno monotone nizove



## Zadatak 7

- × Objasniti postupak i realizaciju algoritma particijskog sortiranja (*quicksort*).
- × Demonstrirati algoritam na primeru sortiranja neuređenog niza 64, 81, 24, 42, 90, 30, 9 i 95.  
Za pivot izabrati prvi element u particiji.
- × Izvesti performanse u najboljem i najgorem slučaju. Kako se može izbeći najgori slučaj?



## Zadatak 7 - Rešenje

- × Quicksort se bazira na strategiji "podeli i osvoji" (*divide and conquer*)
  - × složen problem se razlaže na nekoliko jednostavnijih potproblema koji su po prirodi isti kao i složen (osnovni problem)
  - × novi potproblemi se dalje razlažu na još jednostavnije potprobleme
  - × proces razlaganja se nastavlja sve dok rezultujućí potproblem nije trivijalan za rešavanje
- × Generalna ideja:
  - × podeliti osnovni niz na dva podniza razdvojenih elementom koji se garantovano nalazi na svojoj konačnoj poziciji (poziciji nakon sortiranja)
  - × primeniti ovaj postupak na dobijene podnizove

## Zadatak 7 – Rešenje

```
QUICKSORT(a, low, high)  
if ( $low \geq high$ ) then return  
 $j = \text{PARTITION}(a, low, high)$   
 $\text{QUICKSORT}(a, low, j - 1)$   
 $\text{QUICKSORT}(a, j + 1, high)$ 
```

```
PARTITION(a, down, up)  
 $i = down$   
 $j = up$   
 $pivot = a[down]$   
while ( $i < j$ ) do  
  while ( $(a[i] \leq pivot)$  and ( $i < j$ )) do  $i = i + 1$  end_while  
  while ( $a[j] > pivot$ ) do  $j = j - 1$  end_while  
  if ( $i < j$ ) then  $a[i] \leftrightarrow a[j]$  end_if  
end_while  
 $a[down] = a[j]$   
 $a[j] = pivot$   
return  $j$ 
```

- × Quicksort se najjednostavnije formuliše u svom rekurzivnom obliku
- × Funkcija PARTITION preuređuje osnovni niz i vraća indeks pozicije koja razdvaja dva podniza
  - × PIVOT je element koji razdvaja podnizove
  - × nakon preuređivanja, svi elementi pre pivota moraju biti manji ili jednaki pivotu, svi nakon pivota veći ili jednaki
- ~~× izbor pivota utiče na performanse algoritma~~

## Zadatak 7 – Rešenje

U postavci je rečeno da se za pivot bira prvi element particije.  
Na početku, ceo niz je jedna particija.

<u>64</u>	81	24	42	90	30	9	95
-----------	----	----	----	----	----	---	----

i

j

<u>64</u>	81	24	42	90	30	9	95
-----------	----	----	----	----	----	---	----

i

j

<u>64</u>	81	24	42	90	30	9	95
-----------	----	----	----	----	----	---	----

i



j

<u>64</u>	<b>9</b>	24	42	90	30	<b>81</b>	95
-----------	----------	----	----	----	----	-----------	----

i

j

## Zadatak 7 – Rešenje

<u>64</u>	<b>9</b>	24	42	90	30	<b>81</b>	95
-----------	----------	----	----	----	----	-----------	----

i ↔ j

<u>64</u>	9	24	42	<b>30</b>	<b>90</b>	81	95
-----------	---	----	----	-----------	-----------	----	----

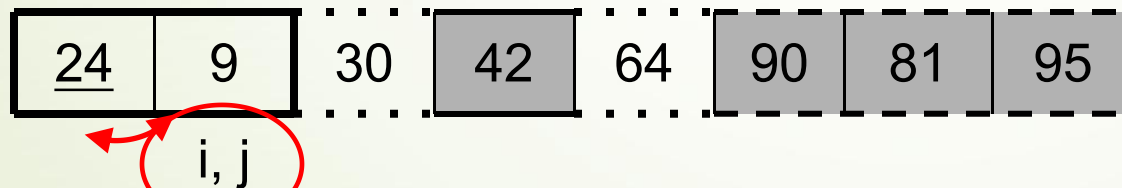
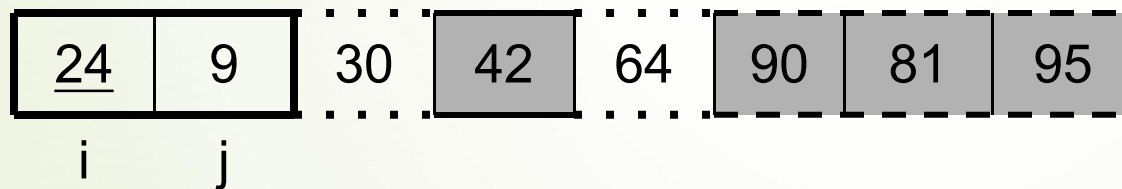
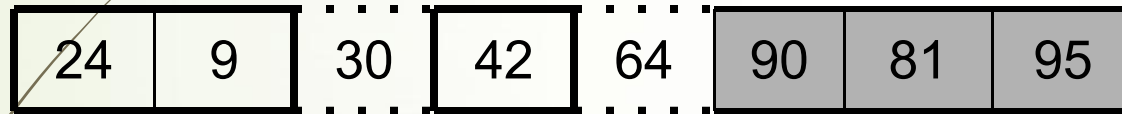
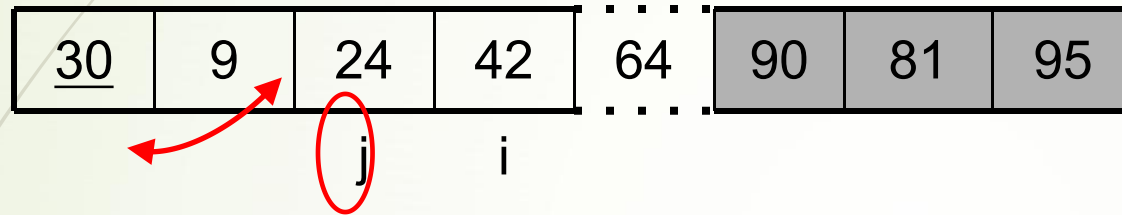
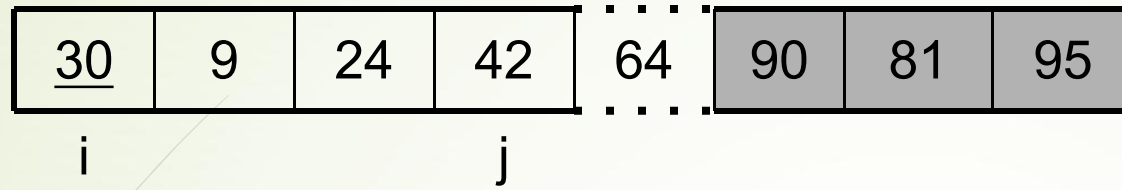
i j

<u>64</u>	9	24	42	30	90	81	95
-----------	---	----	----	----	----	----	----

j i

30	9	24	42	64	90	81	95
----	---	----	----	----	----	----	----

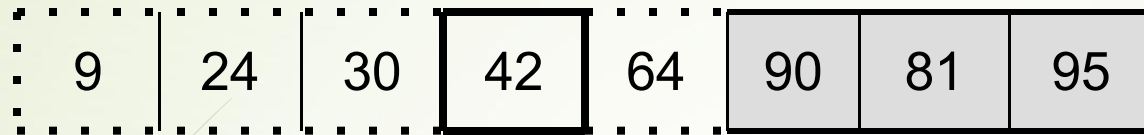
## Zadatak 7 - Rešenje



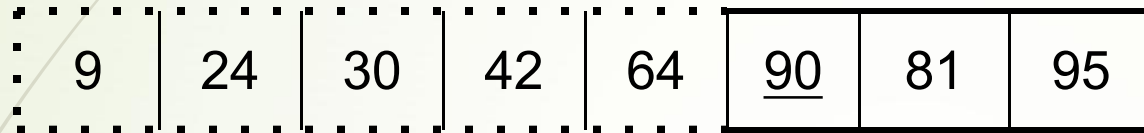
Binarni hip; Algoritmi sortiranja

18.05.2020.

## Zadatak 7 – Rešenje

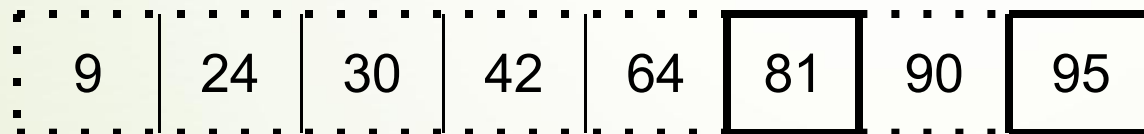
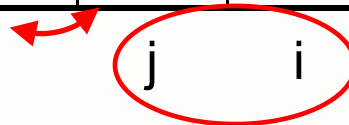
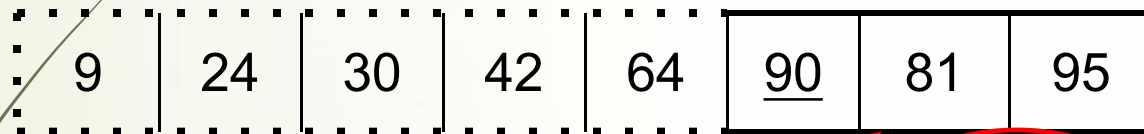


i,j



i

j





# Test

1. Šta je binarni hip?
2. Po čemu se razlikuju maks-hip i min-hip?
3. Kako se hip koristi za sortiranje?
4. Kako se stablo binarnog pretraživanja koristi za sortiranje?
5. Opisati kako funkcioniše shell sort?
6. Šta je pivot i kod kog sorta se koristi?
7. Napisati kod u C-u za insertion sort?
8. Sortirati niz 6, 17, 8, 34, 1, 12, 9 korišćenjem selection sorta.
9. Sortirati niz 6, 17, 8, 34, 1, 12, 9 korišćenjem quick sorta.
10. Sortirati niz 6, 17, 8, 34, 1, 12, 9 korišćenjem heap sorta.



# Test

- × Test poslati do 25.05.2020. u 14h na mejl [apl.jaskovic@np.ac.rs](mailto:apl.jaskovic@np.ac.rs) prema uputstvima sa sajta univerziteta





Hvala na pažnji!