Tampereen yliopisto
Tampere University

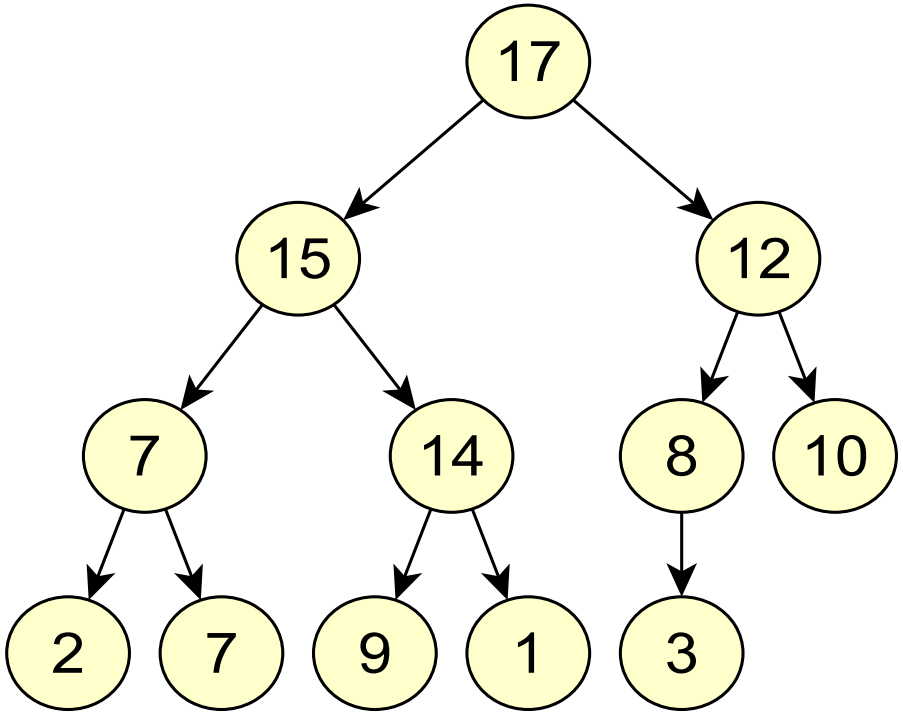
Keko taulukkona

COMP.CS.300 Tietorakenteet ja algoritmit 1

Matti Rintala (matti.rintala@tuni.fi)

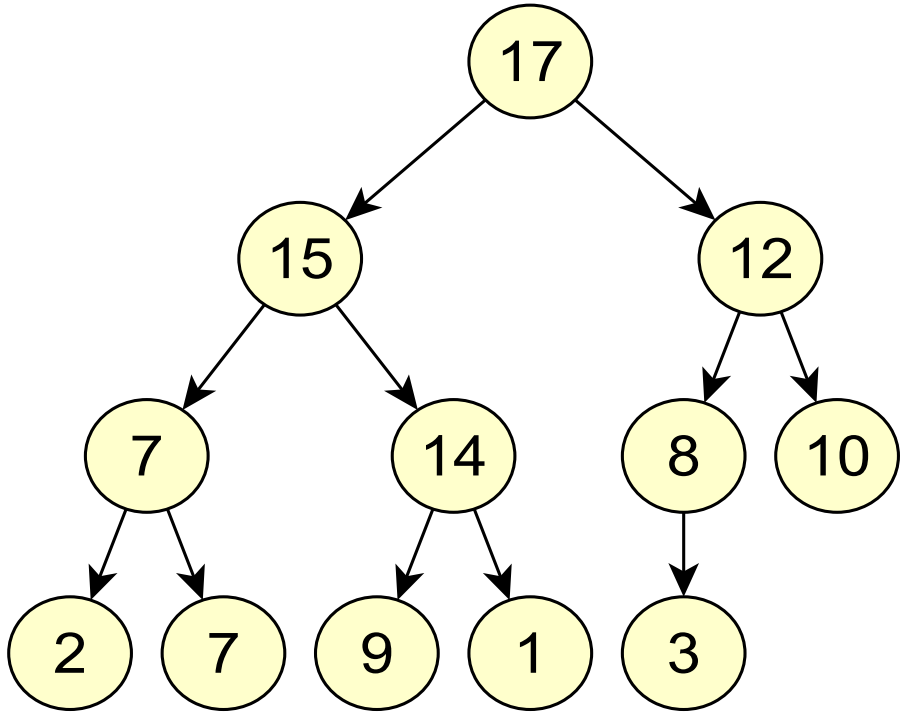
Kertaus : keko

- Binääripuu
- Solmun arvo ≥ lapsien arvot
- Täydellisesti tasapainotettu
  - Haarojen pituusero korkeintaan 1
  - (Pisimmät haarat vasemmalla)



Keko taulukkona

- Säästää muistia, on tehokkaampi
- Solmun yksilöi indeksi *i*
- Keon juuri: A[1]
- Solmun *i* lapset
  Left-Child(*i*):
  return 2**i*
  Right-Child(*i*):
  return 2**i* + *1*
- Solmun vanhempi
  Parent(*i*):
  return ⌊*i* / 2⌋ (pyöristys alas)
- (*A.heapsize* pitää kirjaa, "mihin keko loppuu")



1	2	3	4	5	6	7	8	9	10	11	12
17	15	12	7	14	8	10	2	7	9	1	3

Kekoon lisääminen (Heap Insert)

Heap-Insert(*A*, *value*):

1 Insert-Last(*A, value*)
2 A.heapsize := *A.heapsize* + 1
3 i := *A.heapsize*
4 while *i* > 1 **and** *A*[*i*] > A[Parent(*i*)*]*
5 *A*[*i*] ⇄ A[Parent(*i*)*]*
6 *i* := Parent(*i*)

Idea: Laitetaan uusi alkio keon “pohjalle” ja liu’utetaan sitä ylöspäin, kunnes se on arvonsa kannalta mahdollisessa paikassa.

Keon korjaaminen (Heapify)

- Oletus: keko ok, paitsi ehkä solmu *i*

Heapify(*A*, *i*):

```
 1 repeat
 2       orig_i := i
 3       lc := Left-Child(i)
 4       rc := Right-Child(i)
 5       if lc < A.heapsize and A[lc] > A[i] then
 6             i := lc
 7       if rc < A.heapsize and A[rc] > A[i] then
 8             i := rc
 9       if i ≠ orig_i then
10             A[i] ⇄ A[orig_i]
11 until i = orig_i
```

Keon luominen (Build Heap)

- Oletus: Puun alkiot missä tahansa järjestyksessä

Build-Heap(*A*):

1 A.heapsize := *A.length* (alustetaan koko)
2 for *i* **in** ⌊*A.heapsize* / 2⌋ **downto** 1
3 Heapify(*A*, *i)*

Suurimman poistaminen (Heap Extract Max)

- (Tämä jättää *A*:n loppuun ylimääräisen alkion, jonka voi poistaa)

Heap-Extract-Max(*A*):

1 max := *A*[1]
2 last_node := *A.heapsize*
3 A[1] := *A*[*last_node*]
4 A.heapsize := *A.heapsize* **-** 1
5 Heapify(*A*, 1*)*
6 return *max*

Tampereen yliopisto
Tampere University

Suunnitteluperiaate Muunna-ja-hallitse, kekolajittelu

COMP.CS.300 Tietorakenteet ja algoritmit 1

Matti Rintala (matti.rintala@tuni.fi)

Muunna-ja-hallitse (Transform and conquer)

- Strategia
  - Muunnetaan ongelma *toiseen muotoon*
  - Ratkaistaan tämä *helpommin/tehokkaammin*
  - Ratkaisu *muutetaan* alkup. ongelman ratkaisuksi
- Esimerkkejä muunnoksista:
  - *Yksinkertaistaminen*: "kätevämpi" instanssi samasta ongelmasta
  - *Esitystavan muutos*: saman instanssin toinen esitystapa
  - *Ongelman muunnos*: muutetaan ongelmaksi, jolle on jo valmis algoritmi

Kekolajittelu (heap sort)

- Ongelma: Miten lajitella taulukko?
- Muunnetaan taulukko keoksi
- (Tehostetaan kekoa toteuttamalla taulukkona)
- Poimitaan alkiot keosta suuruusjärjestyksessä

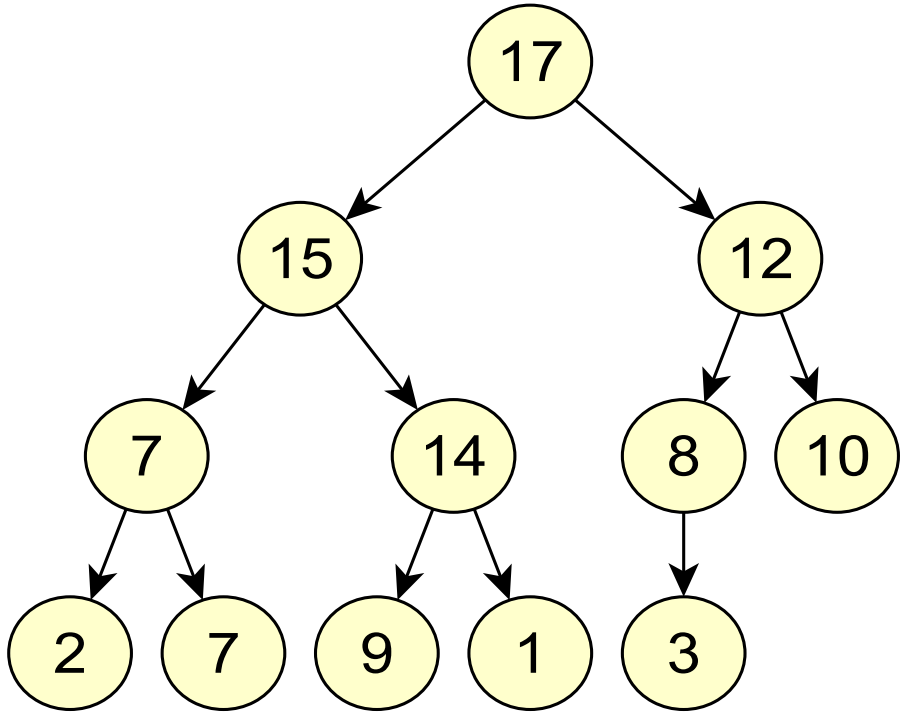
Heap-Sort(*A*):

```
1 Build-Heap(A)
2 while A.heapsize > 1 do
3     A[1] ⇄ A[A.heapsize]
4     A.heapsize := A.heapsize - 1
5     Heapify(A, 1)
```

Kekolajittelu (Heap Sort)

Heap-Sort(*A*):

```
1 Build-Heap(A)
2 while A.heapsize > 1 do
3     A[1] ⇄ A[A.heapsize]
4     A.heapsize := A.heapsize - 1
5     Heapify(A, 1)
```



1	2	3	4	5	6	7	8	9	10	11	12
17	15	12	7	14	8	10	2	7	9	1	3

Tehokkuuksien vertailua

Tapaus	Insertion sort	Quick sort	Merge sort	Heap sort
Paras	n	n log n	n log n	
Keskimäärin	n^2	n log n	n log n	
Huonoin	n^2	n^2	n log n	

Lisämuistitarve
Θ(n)!