

4. Joukkojen käsittely

Tämän luvun jälkeen opiskelija

- osaa soveltaa lomittuvien kasojen operaatioita
- tuntee lomittuvien kasojen toteutuksen binomi- ja Fibonacci-kasoina sekä näiden totetutusten analyysiperiaatteet
- osaa soveltaa erillisten joukkojen yhdisteitä
- tuntee erillisten joukkojen yhdisteen puutoteutuksen ja sen analyysiperiaatteet

Joukkotietorakenteista yleensä

- kurssilla Tietorakenteet on käsitelty etsintäpuita ja hajaustusta (hashing), jotka ovat tärkeitä yleisiä joukkotietorakenteita
- kurssilla Tietorakenteet on myös esitetty prioriteettijonon toteutus "tavallisena" kasana (jota kutsutaan myös binäärikasaksi tai (binääri)keoksi)
- tässä luvussa täydennetään näitä tietoja joillain sellaisilla tietorakenteilla, joita tarvitaan esim. aiemmin esitettyjen verkkoalgoritmien (Dijkstra, Kruskal) tehokkaaseen toteutukseen
- tarkastelussa korostuu tasoitettu analyysi

4.1 Binomikeot

Binomikeoilla toteutetaan tehokkaasti seuraavat *lomitettavien kasojen* (mergeable heaps) operaatiot:

Make-Heap : luo ja palauttaa tyhjän keon

Insert(H, x) : lisää kasaan H alkion x

Minimum(H) : palauttaa keon H pienimmän alkion

Extract-Min(H) : poistaa keon H pienimmän alkion

Union(H_1, H_2) : palauttaa keon jossa on kasojen H_1 ja H_2 alkiot; alkuperäiset H_1 ja H_2 tuhoutuvat

Myös seuraavat operaatiot ovat nopeita:

Decrease-Key(H, x, k) : muuttaa alkion x avaimeksi k , josta oletetaan että se on pienempi kuin avaimen alkuperäinen arvo

Delete(H, x) : poistaa alkion x keosta H

Decrease-Key ja Delete itse asiassa tarvitsevat argumentikseen *osoittimen* alkion x sijaintiin keossa; sivuutetaan tällaiset detaljit jatkossa.

Verrataan operaatioiden suoritusajkoja eri tietorakenteilla:

	binaarikasa	binomikasa	Fib.-kasa
Make-Heap	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$
Insert	$\Theta(\log n)$	$O(\log n)$	$\Theta(1)$
Minimum	$\Theta(1)$	$O(\log n)$	$\Theta(1)$
Extract-Min	$\Theta(\log n)$	$\Theta(\log n)$	$O(\log n)$
Union	$\Theta(n)$	$O(\log n)$	$\Theta(1)$
Decrease-Key	$\Theta(\log n)$	$\Theta(\log n)$	$\Theta(1)$
Delete	$\Theta(\log n)$	$\Theta(\log n)$	$O(\log n)$

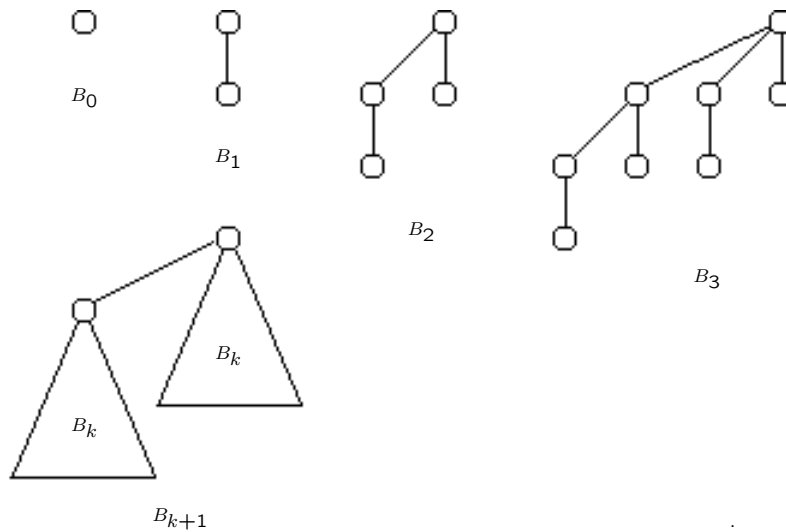
Binaarikeolle ja binomikeolle aikavaativuudet pätevät pahimmassa tapauksessa, Fibonacci-keolle *tasoitetusti*.

Siis binaarikasa ei tue Union-operaatiota, ja Fibonacci-kasa on erityisen tehokas operaatioissa jotka eivät poista alkioita.

Binomipuut

Binomipuu B_k on järjestetty puu (s.o. kunkin solmun lapset ovat järjestyksessä vasemmalta oikealle), joka määritellään rekursiivisesti:

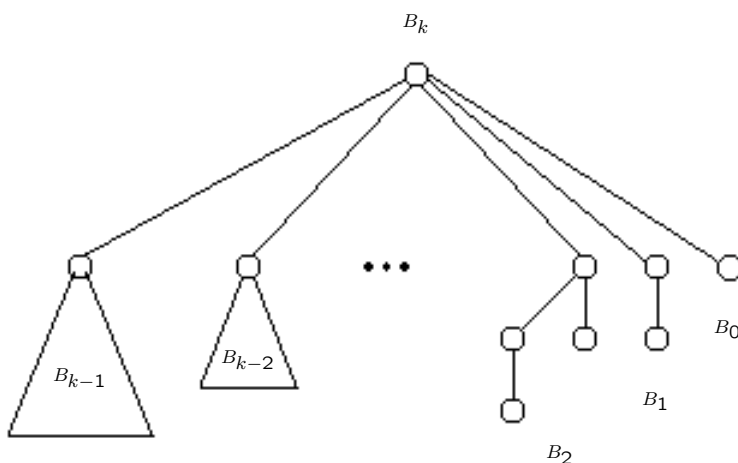
1. B_0 on yksisolmuinen puu ja
2. B_{k+1} saadaan lisäämällä puun B_k juurelle vasemmanpuolimmaisiksi alipuuksi toinen B_k .



Binomipuiden perusominaisuuksia

Seuraavat ominaisuudet nähdään helposti induktiolla:

1. Puussa B_k on 2^k solmua.
2. Puun B_k korkeus on k .
3. Puussa B_k on tasan $\binom{k}{i}$ solmua syvyydellä i ,
 $i = 1, \dots, k$.
4. Puun B_k juuren aste (eli lapsien lukumäärä) on k ,
ja muiden puun B_k solmujen aste on pienempi kuin
 k .
5. Puun B_k juuren lapset ovat vasemmalta luetellen
puiden $B_{k-1}, B_{k-2}, \dots, B_0$ juuria.



Binomikasa on kokoelma binomipuita, joka toteuttaa seuraavat kaksi ehtoa:

1. Kukin **yksittäinen** binomipuu on **kasajärjestyksessä** eli minkään solmun mihinkään lapseen liittyvä avain ei ole suurempi kuin solmun itsensä avain.
2. Kullakin k kokoelmassa on **korkeintaan yksi** muotoa B_k oleva puu.

Ehdon 1 nojalla kunkin puun pienin alkio on juuressa, mutta emme tiedä minkä puun juuressa on koko joukon pienin alkio.

Tarkastellaan ehtoa 2, kun binomikasassa H on kaikkiaan on n alkioita. Olkoon $k = \lfloor \log n \rfloor + 1$ ja luvun n binääriesitys $b_k b_{k-1} \dots b_0$:

$$n = \sum_{i=0}^k b_i 2^i.$$

Koska B_i sisältää 2^i solmua, nähdään että H sisältää puun B_i joss $b_i = 1$.

Nähdään myös, että

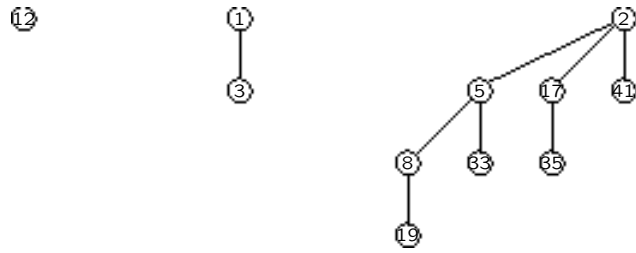
- H sisältää korkeintaan $\lfloor \log n \rfloor + 1$ puuta ja
- jokaisen solmun asteluku binomikasassa H on korkeintaan $\log n$.

Käytämme binomikasoille tavanomaista talletusrakennetta:

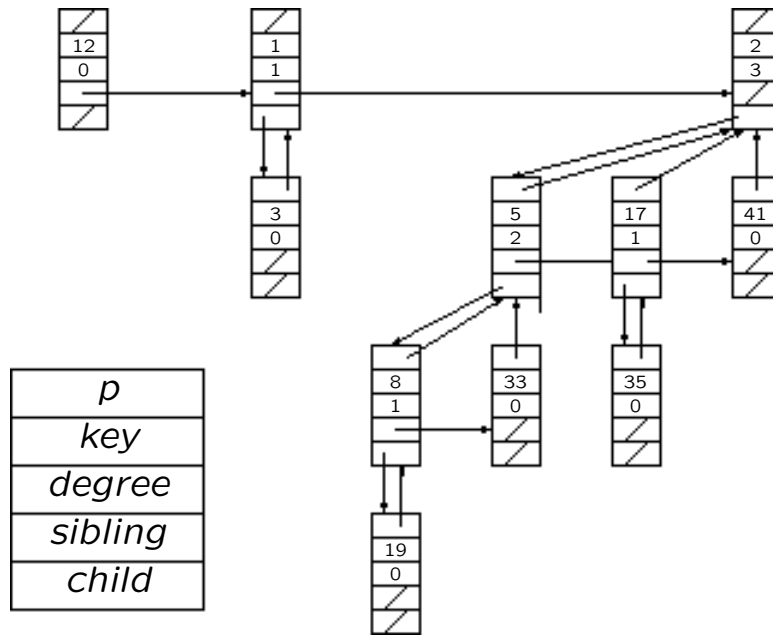
- Kustakin solmusta on linkki vasemmanpuolimmaiseen lapseen (*child*), oikealla puolella olevaan sisarukseen (*sibling*) ja vanhempaan (*p*).
- Kuhunkin solmuun talletetaan myös sen asteluku (*degree*).
- Lisäksi puiden juuret on linkitetty listaan järjestyksessä pienemmistä suurimpaan.
- Binomikasa esitetään osoittimella juurilistan alkuun.

Jätämme hankalimman operaation Union viimeiseksi ja käsittelemme ensin helpot operaatiot.

Erotukseksi jatkossa esiteltävään Fibonacci-kasatoteutukseen käytämme nimiä Binomial-Heap-Union jne.



Binomikasa



Binomikasan linkitetty talletusrakenne

Make-Binomial-Heap: tarvitsee vain alustaa tyhjä juurilista; selvästi $O(1)$.

Binomial-Heap-Minimum: kasaominaisuuden perusteella riittää tarkastaa kaikki juuret, joita on $O(\log n)$.

Binomial-Heap-Insert(H, x): Muodostetaan ensin alkioista x yhden alkion binomikasa H' ja suoritetaan sitten Binomial-Heap-Union(H, H'). Aikavaativuus on selvästi $O(\log n)$, kunhan osoitetaan miten Binomial-Heap-Union tehdään tässä ajassa.

Binomial-Heap-Extract-Min(H):

1. Etsi juurilistan pienin alkio x
2. Muodosta solmun x lapsista (oikealta vasemmalle) lista H'
3. $H := \text{Binomial-Heap-Union}(H, H')$

Aikavaativuus taas selvästi $O(\log n)$ jos Union menee tässä ajassa.

Binomial-Heap-Decrease-Key(H, x, k): Asetetaan ensin $\text{key}[x] := k$; oletuksen mukaan tässä $\text{key}[x]$ ei ainakaan kasva. Jos kasaominaisuus edelleen pätee, operaatio on valmis. Jos kasaominaisuus meni rikki, tämä voi johtua vain siitä, että $k < \text{key}[p[x]]$. Vaihdetaan solmujen x ja $p[x]$ avaimet (ja mahdollinen muu data) ja jatketaan solmusta $p[x]$ kasan korjaamista. Aikavaativuus on $O(\log n)$, koska tämä on yläraja puun syvyydelle.

Havainnollistetaan binomikekojen yhdistämistä vertauksella binäärilukujen yhteenlaskuun.

Lasketaan yhteen luvut $11 = 1 + 2 + 8 = 1011_2$ ja $14 = 2 + 4 + 8 = 1110_2$. Lomitetaan ensin binääriesityksiin perustuvat summat:

$$11 + 14 = 1 + 2 + 8 + 2 + 4 + 8 = 1 + 2 + 2 + 4 + 8 + 8.$$

Tästä pitäisi saada binääriesitys, eli kukin kakkosen potenssi saisi esiintyä vain kerran.

Ruvetaan poistamaan summasta duplikaatteja vasemmalta alkaen. Termi $1 = 2^0$ esiintyy vain kerran, joten sille ei tarvitse tehdä mitään.

Termi $2 = 2^1$ esiintyy kaksi kertaa, joten sovelletaan sääntöä $2 + 2 = 4$:

$$11 + 14 = 1 + 2 + 2 + 4 + 8 + 8 = 1 + 4 + 4 + 8 + 8.$$

Nyt tuli "muistibitti", ja uudessa summassa puolestaan 4 esiintyy kaksi kertaa. Korjataan tilanne:

$$11 + 14 = 1 + 4 + 4 + 8 + 8 = 1 + 8 + 8 + 8.$$

Nyt "muistibitin" takia 8 esiintyy kolme kertaa. Summataan niistä kaksi jälkimmäistä:

$$11 + 14 = 1 + 8 + 8 + 8 = 1 + 8 + 16.$$

On siis saatu $11 + 14 = 16 + 8 + 1 = 11001_2 = 25$.

Idean sovellus binomikasoihin:

- n -alkioinen binomikasa vastaa luvun n binääriesitystä
- B_i on mukana kasassa joss luvussa n bitti i on 1.
- kahden B_i -binomimipuun yhdistäminen yhdeksi B_{i+1} -puuksi vastaa termien yhdistämistä
 $2^i + 2^i = 2^{i+1}$

Tarvitsemme seuraavaa apuproseduuria, joka linkittää puun y puun z juuren vasemmaksi lapseksi. Siis jos y ja z ovat B_i -puiden juuria, solmusta y tulee yhdistetyn B_{k+1} -puun juuri.

Binomial-Link(x, y):

$p[y] := z$

$sibling[y] := child[z]$

$child[z] := y$

$degree[z] := degree[z] + 1$

Toinen tarvittava aliohjelma on **Binomial-Heap-Merge**, joka lomittaa kaksi juurilistaa ja palauttaa osoittimen lomitetun listan alkuun. Lomitetussa listassa on samat binomipuut kuin alkuperäisissä listoissa, ja järjestys on pienimmästä puusta suurimpaan. Siis jos kummassakin alkuperäisessä kasassa on puu B_k , nämä puut ovat lomitetussa listassa peräkkäin; niiden keskinäisellä järjestyksellä ei ole väliä. Lomituksen toteutus jätetään harjoitustehtäväksi.

Kasojen H_1 ja H_2 yhdistäminen tapahtuu nyt seuraavilla kahdella askelella:

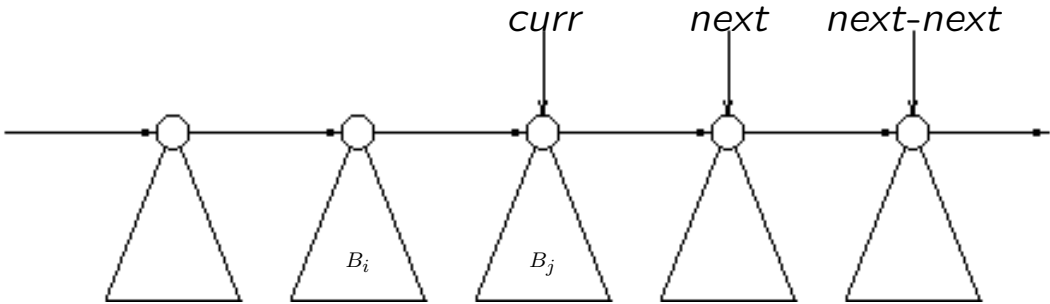
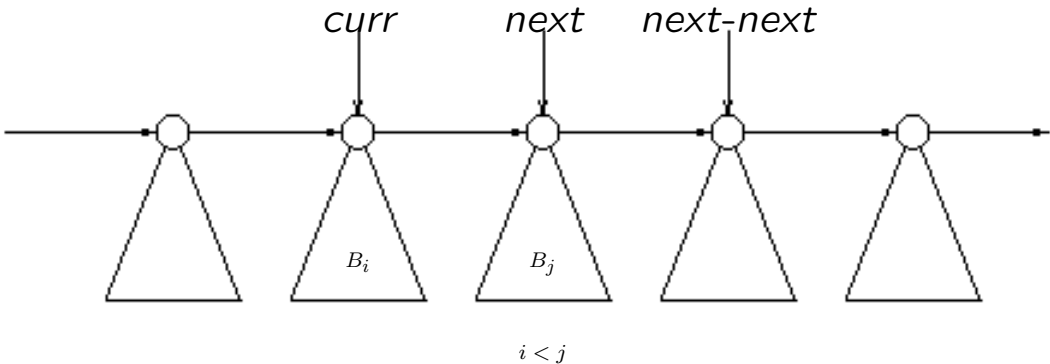
1. Lomita listat; tapahtuu siis kutsulla
 $H := \text{Binomial-Heap-Merge}(H_1, H_2)$.

2. Siisti lista: lomitetusta listasta pitää poistaa tapaukset, joissa esiintyy kaksi puuta B_k .

Vaiheen 2 toteuttamiseksi käydään lista H läpi puu kerrallaan. Osoittimet **curr**, **next** ja **next-next** osoittavat kolmea peräkkäistä listan puuta. (Sivuutetaan tässä vaiheessa listan päissä tulevat erikoistapaukset.) Tarkastelu jakaantuu kolmeen tapaukseen näiden puiden kokojen mukaan.

Tapaus 1: $degree[curr] < degree[next]$.

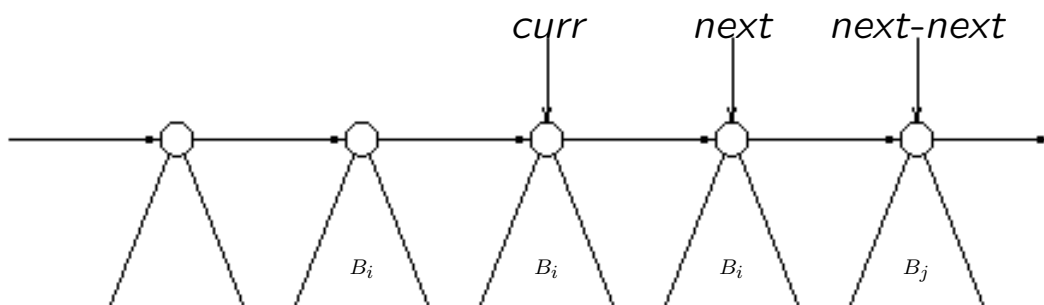
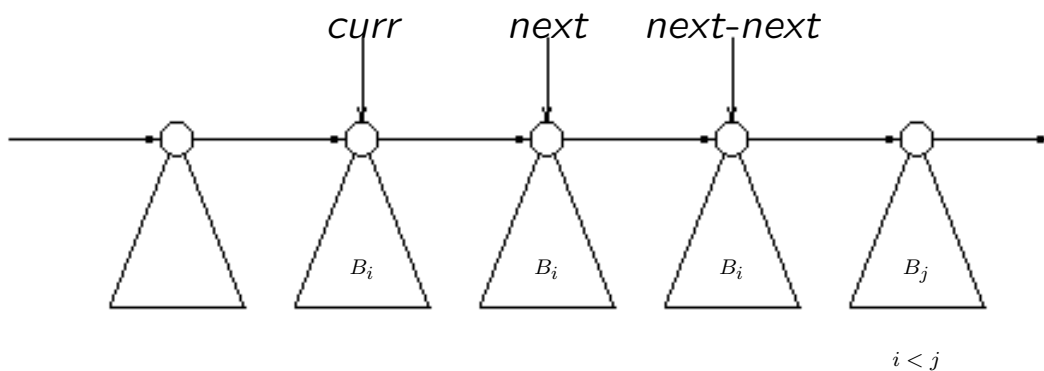
Ei tarvitse tehdä yhdistämisiä tällä kohtaa. Siirrytään listassa eteenpäin.



Tapaus 2: $degree[curr] = degree[next] = degree[next]$.

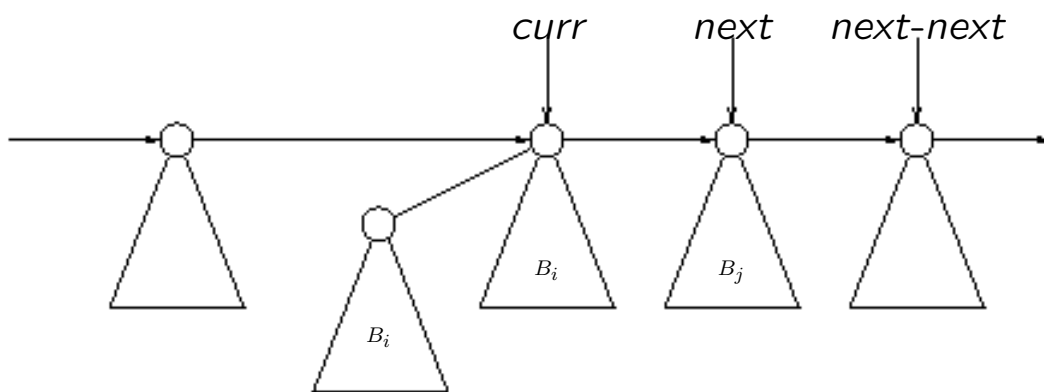
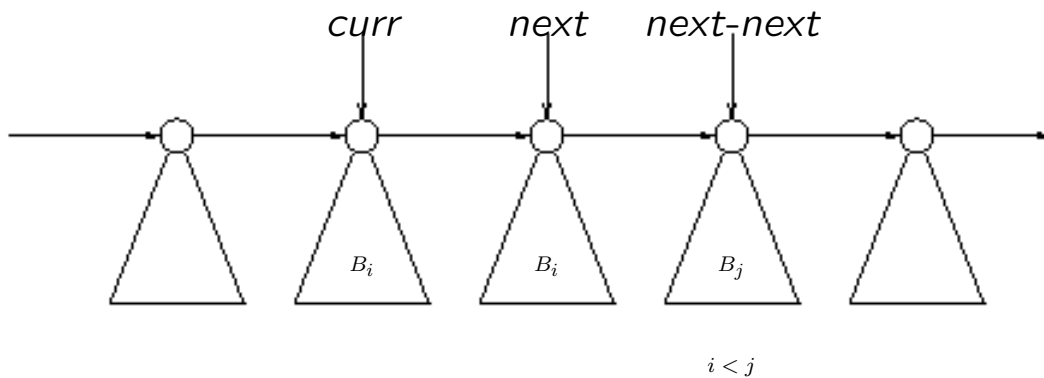
Ei tarvitse tehdä yhdistämisiä tällä kohtaa.
Yhdistäminen jää seuraavalle askelelle. (Huom. B_i voi esiintyä korkeintaan kolmesti: kerran kummastakin alkuperäisestä kasasta ja kerran juuri tehdystä yhdistämisestä.)

Siirrytään listassa eteenpäin.



Tapaus 3: $degree[curr] = degree[next] < degree[next]$.

Nyt yhdistetään *curr* ja *next*. Juureksi tulee kasaehdon säilyttämiseksi se, jolla on pienempi avain. Huomaa, että tapauksessa $degree[next-next] = degree[curr] + 1$ voidaan edelleen joutua yhdistämään tämä uusi puu seuraavaan.



Esitetään koko toimenpide pseudokoodina (seuraava sivu).

Linkkien päivittämiseksi muistetaan vielä solmu *prev* josta löytyy osoitin *curr*-solmuun, ja otetaan erikseen huomioon listan alku.

Algoritmin oikeellisuus: Selvästi yhdisteeseen tulee mukaan tasan ne alkiot, jotka ovat jommassa kummassa alkuperäisistä kasoista. (Sovellukset ovat yleensä sellaisia, että kasan alkioilla on "olioidentiteetti", joten saman avaimen mahdollisia useita esiintymiä ei pidä karsia.)

Kun H_1 ja H_2 toteuttavat kasaominaisuuden, selvästi myös niistä lomittamalla saatu ensimmäinen versio listasta H toteuttaa sen. Yhdistämiset tehdään aina niin, että kasaominaisuus pysyy voimassa. Kun koko lista H on käyty läpi, millään kahdella juurella ei ole samaa astelukua eli mikään B_i ei esiinny kuin korkeintaan kerran. Siis lopuksi H on binomikasa.

Aikavaativuus: Selvästi aikavaativuus on lineaarinen lomitettun listan H pituuden suhteen. Tämä taas on sama kuin binomikasojen H_1 ja H_2 juurilistojen yhteenlaskettu pituus, joka on korkeintaan $2 \log n$ missä n on alkioden yhteismäärä.

```

Binomial-Heap-Union( $H_1, H_2$ ):
   $H :=$  Binomial-Heap-Merge( $H_1, H_2$ )
  if  $H$  tyhjä then return  $H$ 
   $prev :=$  Nil
   $curr := H$ 
   $next :=$  sibling[ $curr$ ]
  while  $next \neq$  Nil do
    if degree[ $curr$ ] = degree[ $next$ ]
      and (degree[ $next$ ] < degree[ $next-next$ ]
        or  $next-next =$  Nil)
    then % Yhdistetään
      if key[ $curr$ ]  $\leq$  key[ $next$ ] then
        sibling[ $curr$ ] :=  $next-next$ 
        Binomial-Link( $next, curr$ )
      else
        if  $prev =$  Nil then % Listan alku
           $H := next$ 
        else
          sibling[ $prev$ ] :=  $next$ 
        end if
        Binomial-Link( $curr, next$ )
         $curr := next$ 
      end if
    end if
     $next := next-next$ 
     $next-next :=$  sibling[ $next-next$ ]
  end while
return  $H$ 

```