Luku 7

Lajittelu

7.3 Lisäys kohdan pikalajittelun loppuun

Valinta (mediaani)

Valinnassa (selection) tehtävänä etsiä kokoelman *k*:nneksi suurin (tai pienin) alkio. Helposti se saadaan selville lajittelemalla kokoelma ja valitsemalla *k*:s alkio. Prioriteettijonosta on myös hyötyä, erityisesti jos *k* on pieni (kts. harjoitukset prioriteettijonoon liittyen). Tärkeä erikoistapaus valinnasta on keskimmäinen alkio eli mediaani.

Pikalajittelun jakoalgoritmia voidaan hyödyntää keskimäärin nopean valinnan toteuttamiseksi. Olkoon valittava alkio *k*:nneksi pienin. Jakoalgoritmihan sijoittaa yhden alkion oikealle paikalleen (olkoon se indeksi *j*), ja muut alkiot oikealle puolelle ko. jakoalkioon nähden. Näinollen, jos $k = j$ valittava alkio on löytynyt, jos $k < j$, valittava alkio löytyy vasemmasta $(1..j-1)$ osataulukosta, jos taas $k = j$, valittava alkio löytyy oikeasta $(j+1..n)$ osataulukosta. Hakua jatketaan rekursiivisesti kunnes oikea kohta löytyy.

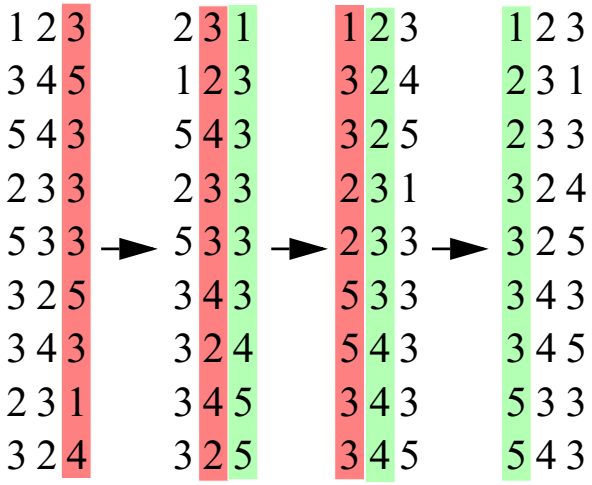
Aikavaativuuden analyysi sujuu kuten pikalajittelussakin, mutta nyt kahden rekursiivisen kutsun sijaan tehdään vain yksi. Jos jako osuu aina puoleenväliin, niin

$$T_{best}(n) = \begin{cases} O(n) + T\left(\frac{n}{2}\right), \text{ kun } n > 1 \\ O(1), \text{ kun } n \leq 1 \end{cases} \Rightarrow \tag{7-1}$$

$$T_{best}(n) = n + \frac{n}{2} + \frac{n}{4} + \ldots + 1$$

$$T_{best}(n) = O(n)$$

Pahimman tapauksen aikavaativuus (ja syötteet joissa se esiintyy) taas on identtinen pikalajittelun kanssa, eli $O(n^2)$. Tämän valinta-algoritmin käyttäminen sellaisenaan on siis melkoinen riski. Mikäli jakoalkioksi valitaan esimerkikisi kolmen valitun alkion mediaani, riski pienenee oleellisesti. Näytekoon kasvattaminen (esimerkiksi kokoon $O(n/\log n)$, joka voidaan lajitella lineaarisessa ajassa) pienentää riskiä edelleen, mutta ei poista sitä. Sensijaan valitsemalla näyte näytteiden älykkäästi, voidaan pikavalinta-algoritmi saada luotettavasti toimimaan lineaarisessa ajassa. "Älykäs" näyte on hakea mediaani alijoukkojen mediaanien joukosta. Tarkempi kuvaus esim. Weiss:n kirjasssa.



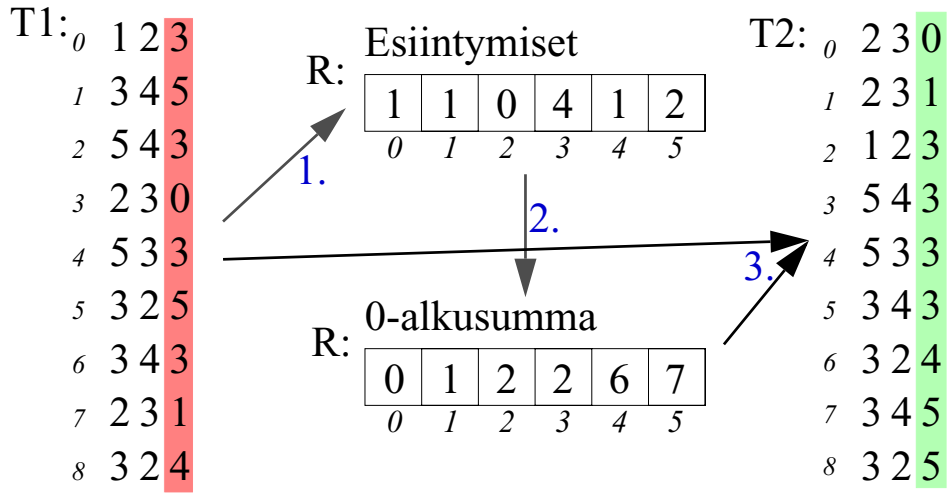
Kuva 7-1: Vaiheittainen lajittelu.

7.6 Lisäys kohdan kaukalolajittelu loppuun

Kantalukulajittelu

Kaukalolajittelussa listojen muodostaminen ja yhdistäminen vie suhteellisen paljon aikaa. Seuraavassa esitettävässä kantalukulajittelussa (radix sort) käytetään toista taulukkoa aputaulukkona ja alkiot viedään suoraan oikealle paikalleen aputaulukkoon. Kuten kaukalolajittettelussakin, lajittelu tehdään avaimenosa kerrallaan joten kunkin lajitteluvaiheen on oltava vakaa (kuva 7-1). Kun kukin vaihe kantalukulajittelua luo senhetkisestä syötetaulukosta uuden taulukon, kannattaa vuorotella alkuperäistä taulukkoa ja yhtä aputaulukkoa. Tämä on erityisen kätevää jos vaiheita on parillinen määrä. Tällöin lopullinen tulos tulee suoraan tallennetuksi syötetaulukkoon.

Kuten kaukalolajittelussakin, varaamme oman kaukalon kullekin mahdolliselle avaimenosan arvolle. Nyt emme kuitenkaan sijoita arvoja kaukaloon, vaan laskentalajittelun tapaan ainoastaan laskemme kuinka monta ko. avainta lajiteltavassa syötteessä esiintyy.



Kuva 7-2: Kantalukulajittelu histogrammilla.

Laskettuamme kunkin avaimenosan esiintymien määrät, voimme laskea kunkin avaimen sijainnin (tämän vaiheen) tulostaulukossa. Tämä suoritetaan laskemalla esinntymistaulukosta 0-alkusumma (0, a_1, a_1+a_2, a_1+a_2+a_3, …). Alkusummat osoittavat suoraan mihin kunkin osaavaimen mukainen ensimmäinen alkio kuuluu tulostaulukossa. Käytännössä siis käymme syötetaulukon uudestaan läpi ja kunkin alkio kohdalla sijoitamme sen tulostaulukkoon alkusummataulukon osoittamaan paikkaan. Kuva 7-2 esittää kunkin taulukon roolin kantalukulajittelussa. Seuraava saman osaavaimen alkio kuuluu seuraavaan indeksiin jne. Tämä toteutetaan kätevästi kasvattamalla alkusummataulukon arvoja yhdellä aina kun olemme alkio sijoittaneet tulostaulukkoon. Jos käyttämämme taulukot ovat 1-alkuisia, teemme alkusummataulukon arvojen kasvatuksen ennen sijoittamista tulostaulukkoon. Toinen läpikäynti on siis (1-alkuisille taulukoille):

```
for i := 1 to N do begin                                    (7-2)
        inc(R[avain(T1[i], k)]);
        T2[R[avain(T1[i], k)]] := T[i];
end;
```

missä k on kaukalolajittelun kierros (monesko osa-avain on kyseessä) ja *avain*() ottaa koko avaimesta k:nnen osa-avaimen.

Kantalukulajittelu on ehkä nopein lajittelualgoritmi reaalimaailman syötteille jos lajiteltavaa on paljon eivätkä avaimet ole kovin pitkiä. Jollei avain ole jotain erityistä tyyppiä (kuten esimerkissämme numeroita), käytetään osaavaimena avaimen bittiesityksen bittejä lohko kerrallaan. Esimerkiksi 16 bittiä kerrallaan (jolloin kaukaloita on 65536 kappaletta). AIkavaativuus on tällöin $O\left(\frac{m}{r}(n+2^r)\right)$, missä r on avaimen pituus bitteinä. Kaukalojaittelun nopeus käytännössä jotuu paljolti sen yksinkertaisuudesta, sekä siitä, että syötettä käydään läpi lähinnä peräkkäisjärjestyksessä (joskin alkusummataulukkoa ja tulostaulukkoa osoitetaan myös satunnaisjärjestyksessä).