

Esimerkki Lomitusjärjestäminen

```
merge-sort( $A, p, q$ ):  
var  $k$       % paikallinen muuttuja, vakiotila  
1. if  $p < q$  then  
2.      $r := \lfloor (p + q)/2 \rfloor$   
3.     merge-sort( $A, p, r$ )  
4.     merge-sort( $A, r + 1, q$ )  
5.     merge( $A, p, r, q$ )  
end if
```

Olkoon $S(n)$ kutsun merge-sort(A, p, q) tilavaativuus kun $p - q + 1 = n$.

Oletetaan merge toteutetuksi vakioyötilassa (ei-triviaalia mutta mahdollista).

Saadaan

$$\begin{aligned} S(n) &= \max \{ S(\lceil n/2 \rceil), S(\lfloor n/2 \rfloor) \} + \Theta(1) \\ &= S(\lceil n/2 \rceil) + \Theta(1) \end{aligned}$$

joten

$$S(n) = \Theta(\log n).$$

3. Algoritmien suunnittelutekniikoita

Tutustutaan tärkeimpiin algoritmien suunnitteluperiaatteisiin ja tarkastellaan esimerkkien valossa niiden soveltamista.

Tämän luvun jälkeen opiskelija osaa

1. laatia ja analysoida osittamiseen ja taulukointiin perustuvia algoritmeja,
2. soveltaa heuristisia ratkaisumenetelmiä (ahneus, paikallinen etsintä) ja tarkastella, milloin nämä johtavat optimaaliseen lopputulokseen sekä
3. ratkaista kombinatorisia ongelmia täydellisellä etsinnällä (peruutus, karsinta).

Esimerkkeinä tarkastelluista keskeisistä verkko-ongelmista (lyhimmät polut, virittävät puut) tulisi muistaa niiden ratkaisualgoritmit ja analyysiperiaatteet.

3.1 Osittaminen (divide and conquer)

Esimerkki Lomitusjärjestäminen edellä

Periaate yleisemmin

```
ratkaise( $S$ ):  
  if  $|S|$  on "pieni"  
  then käytä jotain triviaalialgoritmia  
  else  
    hajota( $S; S_1, \dots, S_m$ )  
    ratkaise( $S_1$ )  
    ratkaise( $S_2$ )  
    ...  
    ratkaise( $S_m$ )  
    yhdistä( $S_1, \dots, S_m; S$ )  
  end if
```

Yleensä hajotuksessa yritetään saada osat S_i mahdollisimman samankokoisiksi. Jos lisäksi $|S| = \sum_i |S_i|$ (eli $|S_i| = |S|/m$), saadaan aikavaativuudelle $T(n)$ yhtälö

$$\begin{aligned} T(n) &= \Theta(1) && \text{kun } |S| \text{ "pieni"} \\ T(n) &= mT(n/m) + f(n) && \text{muuten} \end{aligned}$$

missä f on hajottamisen ja yhdistämisen aikavaativuus.

Jos jako on kovin epätasainen, aikavaativuus yleensä kasvaa:

$$\begin{aligned} \text{lisäysjärjestäminen} &: n - 1 \quad \text{plus} \quad 1 \\ \text{lomitusjärjestäminen} &: \lceil n/2 \rceil \quad \text{plus} \quad \lceil n/2 \rceil \end{aligned}$$

Esimerkki Seuraava funktiokutsu $\text{minmax}(S)$ lukujoukolla $S = \{s_1, \dots, s_n\}$ palauttaa parin $(\min S, \max S)$:

```
minmax(S)
  if |S| = 2
  then return (min { s1, s2 }, max { s1, s2 })
  else
    r := ⌊n/2⌋
    S1 := { s1, ..., sr }
    S2 := { sr+1, ..., sn }
    (a1, b1) := minmax(S1)
    (a2, b2) := minmax(S2)
    return (min { a1, a2 }, max { b1, b2 })
  end if
```

Vertailujen lukumäärälle $T(|S|)$ saadaan

$$T(2) = 1$$

$$T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + 2 \quad \text{kun } n > 2$$

eli (kun $n = 2^k$ jollain k)

$$T(n) = \frac{3}{2}n - 2.$$

Triviaaliratkaisu vaatii $(n - 1) + (n - 2) = 2n - 3$ vertailua.

Opetus: päällekkäistä työtä kannattaa välttää.

Mediaaniongelma ja sen yleistys

(order statistics)

Annettu: joukko S , $|S| = n$, ja luku $k \leq n$. Oletetaan, että joukon alkioden välillä on lineaarinen järjestysrelaatio \leq . Yleisemmin voidaan sallia, että S on monijoukko (multiset) eli sama alkio saa esiintyä useita kertoja; joukko-operaatiot yleistyvät ilmeisellä tavalla.

Määrättävä: joukon k :nneksi pienin alkio, ts. b_k kun $S = \{b_1, \dots, b_n\}$ missä $b_1 \leq \dots \leq b_n$.

(Eryityisesti **mediaani:** $k = \lceil n/2 \rceil$.)

Ilmeisiä ratkaisuja:

- etsitään pienin, toiseksi pienin, \dots , k . pienin; aikavaativuus $\Theta(kn) = O(n^2)$
- järjestetään koko joukko; aikavaativuus $\Theta(n \log n)$

Seuraavassa esitetään kaksi edistyneempää ratkaisua:

- yksinkertainen **satunnaisalgoritmi** jonka aikavaativuus on $O(n)$ **keskimäärin**
- hieman monimutkaisempi deterministinen algoritmi jonka aikavaativuus on $O(n)$ pahimmassa tapauksessa

Ratkaisut perustuvat osittamiseen, ongelmana on löytää riittävän tasainen jako.

Satunnaisalgoritmi valintaongelmaan

```
select-r( $S, k$ )
  if  $S = \{a\}$ 
  then return  $a$     % vain yksi alkio
  else
     $a := \text{random}(S)$  % valitaan satunnainen alkio
     $S_1 := \{x \in S \mid x < a\}$ 
     $S_2 := \{x \in S \mid x = a\}$ 
     $S_3 := \{x \in S \mid x > a\}$ 
    if  $k \leq |S_1|$  then return select-r( $S_1, k$ )
    else if  $k \leq |S_1| + |S_2|$  then return  $a$ 
    else return select-r( $S_3, k - |S_1| - |S_2|$ )
  end if
```

Algoritmin perusidea: Satunnaisen alkion a pitäisi tyypillisesti osua joukon keskivaiheille, joten keskimäärin saadaan

$$T(n) \approx T(n/2) + O(n)$$

eli $T(n) = O(n)$. Tarkemmin todetaan että kullakin $1 \leq r \leq n$ saadaan kukin jako $(|S_1|, |S_3|) = (r - 1, n - r)$ todennäköisyydellä $1/n$ (olettaen että $|S_2| = 1$) joten

$$T(n) \leq \frac{1}{n} \sum_{r=1}^n T(\max\{r - 1, r - k\}) + O(n)$$

(yksityiskohdat HT).

Huom. odotusarvot ovat yli algoritmin sisäisten satunnaisvalintojen, joten tämä **ei** ole keskimääräisen tapauksen analyysia aiemmin esitetystä mielestä.

Deterministinen algoritmi valintaongelmaan (Blum, Floyd, Pratt, Rivest, Tarjan 1983)

```
select( $S, k$ )
  if  $|S| \leq 50$  then
    järjestä  $S$  ja palauta  $k$ . alkio
  else
    olkoon  $n = |S|$  ja  $r = \lceil n/5 \rceil$ 
    jaa  $S$  viisialkioisiin osiin  $P_1, \dots, P_r$ 
      (osa  $P_r$  jää vajaaksi jos ei mene tasan)
    for  $i := 1$  to  $r$  do  $\Theta(n)$ 
       $m_i := \text{select}(P_i, 3)$ 
     $M := \{m_1, \dots, m_r\}$ 
     $a := \text{select}(M, \lceil M/2 \rceil)$   $T(n/5)$ 
     $S_1 := \{x \in S \mid x < a\}$ 
     $S_2 := \{x \in S \mid x = a\}$ 
     $S_3 := \{x \in S \mid x > a\}$ 
    if  $k \leq |S_1|$  then return  $\Theta(|S_1|)$ 
      select( $S_1, k$ )
    else if  $k \leq |S_1| + |S_2|$  then return  $a$ 
    else return  $\Theta(|S_3|)$ 
      select( $S_3, k - |S_1| - |S_2|$ )
    end if
  end if
end if
```

Jakoalkion a valintaa lukuunottamatta toimii kuten edellä. **Kysymys:** joukkojen S_1 ja S_3 kokorajat?

Aikavaatimukselle $T(n)$ saadaan

$$T(n) \leq T(n/5) + \max\{T(|S_1|), T(|S_3|)\} + \Theta(n).$$

Kysymys: takaako jakoalkion valintamenettely, että $|S_1|$ ja $|S_3|$ ovat riittävän pieniä?

Jos $m_i \leq a$, niin joukossa P_i on ainakin 3 alkiota p joille $p \leq a$. Tällaisia m_i on ainakin $\lceil r/2 \rceil$, joten

$$|S_1| + |S_2| \geq 3\lceil r/2 \rceil \geq \lceil n/4 \rceil.$$

Vastaavasti jos $m_i \geq a$, niin joukossa P_i on ainakin 3 alkiota p joille $p \geq a$, joten

$$|S_3| + |S_2| \geq 3\lceil r/2 \rceil \geq \lceil n/4 \rceil.$$

Siis kaiken kaikkiaan

$$|S_1| \leq \left\lfloor \frac{3}{4}n \right\rfloor \quad \text{ja} \quad |S_3| \leq \left\lfloor \frac{3}{4}n \right\rfloor$$

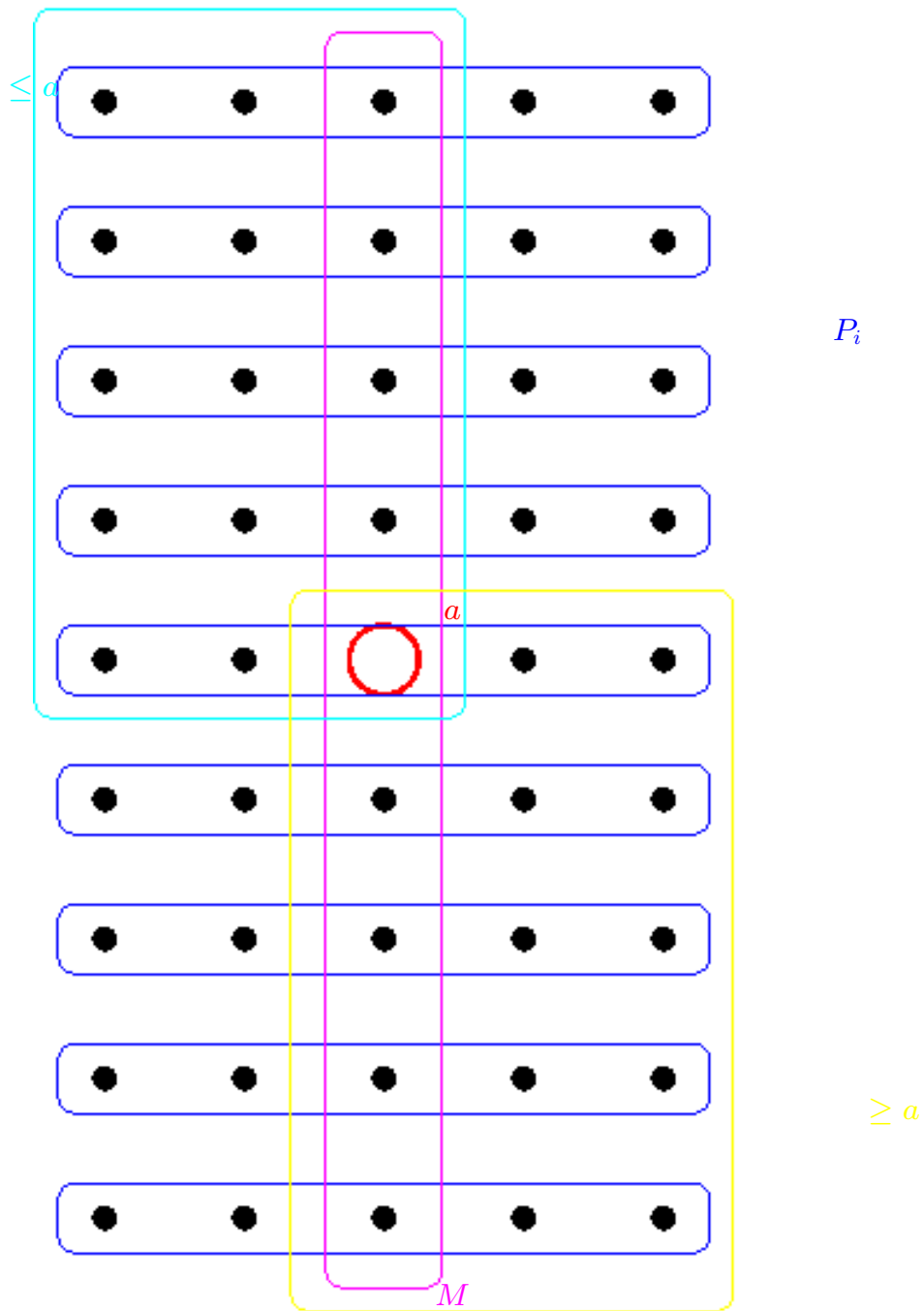
joten saadaan

$$T(n) \leq T(\lceil n/5 \rceil) + T(\lfloor 3n/4 \rfloor) + \Theta(n)$$

mistä voidaan osoittaa $T(n) = O(n)$ (oleellisesti koska $n/5 + 3n/4 < n$).

Tarkemmin voidaan osoittaa vertailujen lukumääräksi $18n + O(1)$. Tämä voidaan parantaa arvoon $14\frac{1}{3}n + O(1)$ tarkastelemalla 9-alkioisia osia.

Opetus: osatapauksiin jakaminen voi olla ei-triviaalia.



Joukon S alkioista ainakin $|S|/4$ on korkeintaan jakoalkion a suuruisia (vasen ylänurkka) ja ainakin $|S|/4$ vähintään jakoalkion a suuruisia (oikea alanurkka).

Kertolaskualgoritmeja

Tarkastellaan kahden n -bittisen luvun X_1 ja X_2 kertolaskua. Oletetaan yksinkertaisuuden vuoksi $n = 2^k$.

"Peruskoulualgoritmi" laskee tulon ajassa $\Theta(n^2)$. Onko parempaa? Huom. yhteenlasku menee ajassa $\Theta(n)$.

Osittamisratkaisua varten olkoon A_i luku joka saadaan ottamalla luvun X_i esityksestä $n/2$ eniten merkitsevää bittiä ja B_i ottamalla $n/2$ vähiten merkitsevää. Siis

$$X_i = A_i 2^{n/2} + B_i, \quad i = 1, 2.$$

Osittava ratkaisu 1 (peruskoulualgoritmin yleistys):

$$\begin{aligned} X_1 X_2 &= (A_1 2^{n/2} + B_1)(A_2 2^{n/2} + B_2) \\ &= A_1 A_2 2^n + (A_1 B_2 + A_2 B_1) 2^{n/2} + B_1 B_2. \end{aligned}$$

Muotoa $Z2^k$ olevat kertolaskut menevät sivuttaissiirtoina ajassa $\Theta(k)$. Siis n -bittinen kertolasku palautuu neljään $n/2$ -bittiseen:

$$T(n) = 4T(n/2) + \Theta(n), \quad n = 2^k \geq 2.$$

Kertaluokkaan ei tule parannusta:

$$T(n) = \Theta(n^{\log_2 4}) = \Theta(n^2).$$

Osittava ratkaisu 2: järjestetään termit uudelleen.

$$\begin{aligned} X_1 X_2 &= (A_1 2^{n/2} + B_1)(A_2 2^{n/2} + B_2) \\ &= \underline{A_1 A_2 2^n} + \underline{B_1 B_2} \\ &\quad + ((A_1 - B_1)(B_2 - A_2) + \underline{A_1 A_2} + \underline{B_1 B_2}) 2^{n/2}. \end{aligned}$$

Nyt kertolaskuja on vain kolme:

$$T(n) = 3T(n/2) + \Theta(n), \quad n = 2^k \geq 2,$$

joten

$$T(n) = \Theta(n^{\log_2 3}) = \Theta(n^{1,59\dots}).$$

Opetus: osaratkaisujen kokoaminen voi olla epätriviaalia.

Huom.

- vakiotekijöiden takia peruskoulualgoritmi voi olla parempi vielä noin 500-bittisillä luvuilla
- pitkien lukujen kertolaskua tarvitaan esim. kryptografiassa
- asympotoottisesti tehokkain tunnettu menetelmä **Schönhage-Strassen -algoritmi** toimii ajassa $O(n(\log n)(\log \log n))$
- avoin kysymys: onko $O(n)$ mahdollista?

Matriisitulo $C = AB$ kahdelle $n \times n$ matriisille A ja B ;
taas oletetaan $n = 2^k$

Suoraan kaavasta $c_{ij} = \sum_{k=1}^n a_{ik}b_{kj}$ saadaan
aikavaativuus $\Theta(n^3)$.

Osittavaa ratkaisua varten jaetaan A ja B neljään
 $(n/2) \times (n/2)$ -alimatriisiin:

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \quad B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}.$$

Tunnetusti nyt

$$C = AB = \begin{pmatrix} A_{11}B_{11} + A_{12}B_{21} & A_{11}B_{12} + A_{12}B_{22} \\ A_{21}B_{11} + A_{22}B_{21} & A_{21}B_{12} + A_{22}B_{22} \end{pmatrix}.$$

Alimatriisien kertolaskuja on 8, joten ottamalla
huomioon ajan $\Theta(n^2)$ vievät yhteenlaskut saadaan

$$T(n) = 8T(n/2) + \Theta(n^2)$$

joten parannusta ei synny:

$$T(n) = \Theta(n^{\log_2 8}) = \Theta(n^3).$$

Strassenin algoritmi: lasketaan

$$AB = \begin{pmatrix} M_2 + M_3 & M_1 + M_2 + M_5 + M_6 \\ M_1 + M_2 + M_4 - M_7 & M_1 + M_2 + M_4 + M_5 \end{pmatrix}$$

missä

$$M_1 = (A_{21} + A_{22} - A_{11})(B_{22} - B_{12} + B_{11})$$

$$M_2 = A_{11}B_{11}$$

$$M_3 = A_{12}B_{21}$$

$$M_4 = (A_{11} - A_{21})(B_{22} - B_{12})$$

$$M_5 = (A_{21} + A_{22})(B_{21} - B_{11})$$

$$M_6 = (A_{12} - A_{21} + A_{11} - A_{22})B_{22}$$

$$M_7 = A_{22}(B_{11} + B_{22} - B_{12} - B_{21})$$

Nyt tarvitaan vain 7 kertolaskua, joten

$$T(n) = 7T(n/2) + \Theta(n^2)$$

mistä saadaan

$$T(n) = \Theta(n^{\log_2 7}) = \Theta(n^{2,807} \dots).$$

Huomattavaa:

- Strassenin menetelmä voi tiheillä matriiseilla olla käytännöllinen jo kun $n \approx 45$ (harvoille matriiseille on omat algoritminsa)
- asympotoottisesti tehokkain tunnettu algoritmi vaatii ajan $\Theta(n^{2,376\dots})$
- paras tunnettu alaraja on triviaali $\Omega(n^2)$
- algoritmi yleistyy muillekin kuin reaalmatriiseille

3.2 Taulukointi (dynamic programming)

Esimerkki binomikertoimien $\binom{n}{k}$ laskeminen palautuskaavasta

$$\binom{n}{0} = \binom{n}{n} = 1$$
$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}, \quad 0 < k < n$$

Naiivilla rekursiivisella algoritmilla

```
b(n, k)
  if k = 0 or k = n
    then return 1
    else return b(n - 1, k - 1) + b(n - 1, k)
  end if
```

aikavaatimus on ilmeisesti $\Theta(\binom{n}{k})$, joka kasvaa eksponentiaalisesti esim. jos $k = n/2$. (Jos k on vakio niin $\binom{n}{k} = \Theta(n^k)$.)

Algoritmi on tehoton, koska sama asia lasketaan moneen kertaan:

$$\begin{aligned} b(8, 4) &= b(7, 3) + b(7, 4) \\ &= b(6, 2) + b(6, 3) + b(6, 3) + b(6, 4) \\ &= b(5, 1) + b(5, 2) + b(5, 2) + b(5, 3) \\ &\quad + b(5, 2) + b(5, 3) + b(5, 3) + b(5, 4). \end{aligned}$$

Tehokkaampaa on tietysti pitää kirjaa jo saavutetuista tuloksista.

Ensimmäinen versio on ns. [implisiittinen taulukointi](#):

- varataan taulukko $C[0 \dots n, 0 \dots k]$ ja alustetaan sen kaikki alkiot nolllaksi
- varsinainen laskenta tehdään edelleen rekursiivisesti:

```
 $b(m, j)$   
  if  $C[m, j] \neq 0$   
  then return  $C[m, j]$   
  else  
    if  $j = 0$  or  $j = m$   
    then  $C[m, j] := 1$   
    else  $C[m, j] := b(m - 1, j - 1) + b(m - 1, j)$   
    end if  
    return  $C[m, j]$   
  end if
```

Yleensä, kuten tässäkin, pelkkä taulukko kuitenkin riittää.

Laskut voidaan helposti järjestää ilman rekursiotakin niin, että tarvittavat arvot ovat valmiina silloin kuin pitääkin.

```
 $b(n, k)$   
  for  $m := 0$  to  $n$  do  
     $C[m, 0] := 1$   
     $C[m, m] := 1$   
    for  $j := 1$  to  $m - 1$  do  
       $C[m, j] := C[m - 1, j - 1] + C[m - 1, j]$   
    end for  
  end for  
  return  $C[n, k]$ 
```

Tämän naiivin toteutuksen aika- ja tilavaativuus ovat $\Theta(m^2)$. Algoritmia voidaan vielä tehostaa (HT) huomaamalla että

- arvoja $C[m, j]$ missä $j > k$ ei tarvita ja
- taulukosta riittää tallettaa viimeisin rivi.