5 Kertaluokkamerkinnät

Tässä luvussa käsitellään asymptoottisessa analyysissa käytettyjä matemaattisia merkintätapoja

Määritellään tarkemmin Θ, sekä kaksi muuta saman sukuista merkintää O ja Ω.

5.1 Asymptoottinen aika-analyysi

Edellisessä luvussa yksinkertaistimme suoritusajan lauseketta melkoisesti:

- jätimme jäljelle ainoastaan eniten merkitsevän termin
- poistimme sen edestä vakiokertoimen

$\Rightarrow$ kuvastavat algoritmin käyttäytymistä, kun syötekoko kasvaa kohti ääretöntä

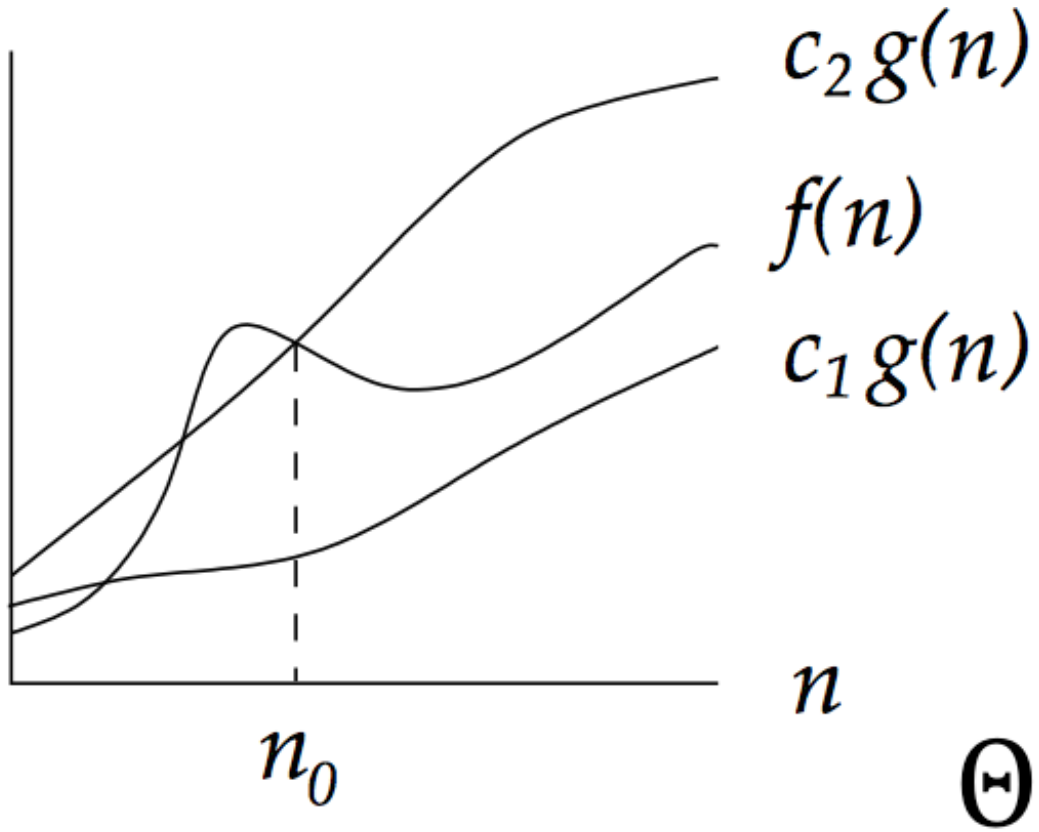
- siis kuvaavat *asymptoottista* suorituskykyä

$\Rightarrow$ antavat käyttökelpoista tietoa **vain jotain rajaa suuremmilla syötteillä**

- todettiin, että usein raja on varsin alhaalla

  $\Rightarrow$ Θ- yms. merkintöjen mukaan nopein on myös käytännössä nopein, paitsi aivan pienillä syötteillä

Θ-merkintä



Kuva 8: Θ-merkintä

eli matemaattisesti

- olkoon $g(n)$ funktio luvuilta luvuille

  $\Theta(g(n))$ *on niiden funktioiden* $f(n)$ *joukko, joille on olemassa positiiviset vakiot* c_1, c_2 *ja* n_0 *siten, että aina kun* $n \geq n_0$, *niin*

$$0 \leq c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$$

  - $\Theta(g(n))$ on *joukko* funktioita
    $\Rightarrow$ tulisi kirjoittaa esim. $f(n) \in \Theta(g(n)) \Rightarrow$ ohjelmistotieteessä vakiintunut käytäntö kuitenkin on käyttää = -merkintää

Funktion $f(n)$ kuuluvuuden kertaluokkaan $\Theta(g(n))$, voi siis todistaa etsimällä jotkin arvot vakioille c_1, c_2 ja n_0 ja osoittamalla, että funktion arvo pysyy n:n arvoilla n_0:sta alkaen arvojen $c_1 g(n)$ ja $c_2 g(n)$ välillä (eli suurempana tai yhtäsuurena kuin $c_1 g(n)$ ja pienempänä tai yhtäsuurena, kuin $c_2 g(n)$).

Esimerkki: $3n^2 + 5n - 20 = \Theta(n^2)$

- valitaan $c_1 = 3$, $c_2 = 4$ ja $n_0 = 4$
- $0 \leq 3n^2 \leq 3n^2 + 5n - 20 \leq 4n^2$ kun $n \geq 4$, koska silloin $0 \leq 5n - 20 \leq n^2$
- yhtä hyvin olisi voitu valita $c_1 = 2$, $c_2 = 6$ ja $n_0 = 7$ tai $c_1 = 0{,}000\,1$, $c_2 = 1\,000$ ja $n_0 = 1\,000$
- tärkeää on vain, että voidaan valita *jotkut* positiiviset, ehdot täyttävät c_1, c_2 ja n_0

Tärkeä tulos: jos $a_k > 0$, niin

$$a_k n^k + a_{k-1} n^{k-1} + \cdots + a_2 n^2 + a_1 n + a_0 = \Theta(n^k)$$

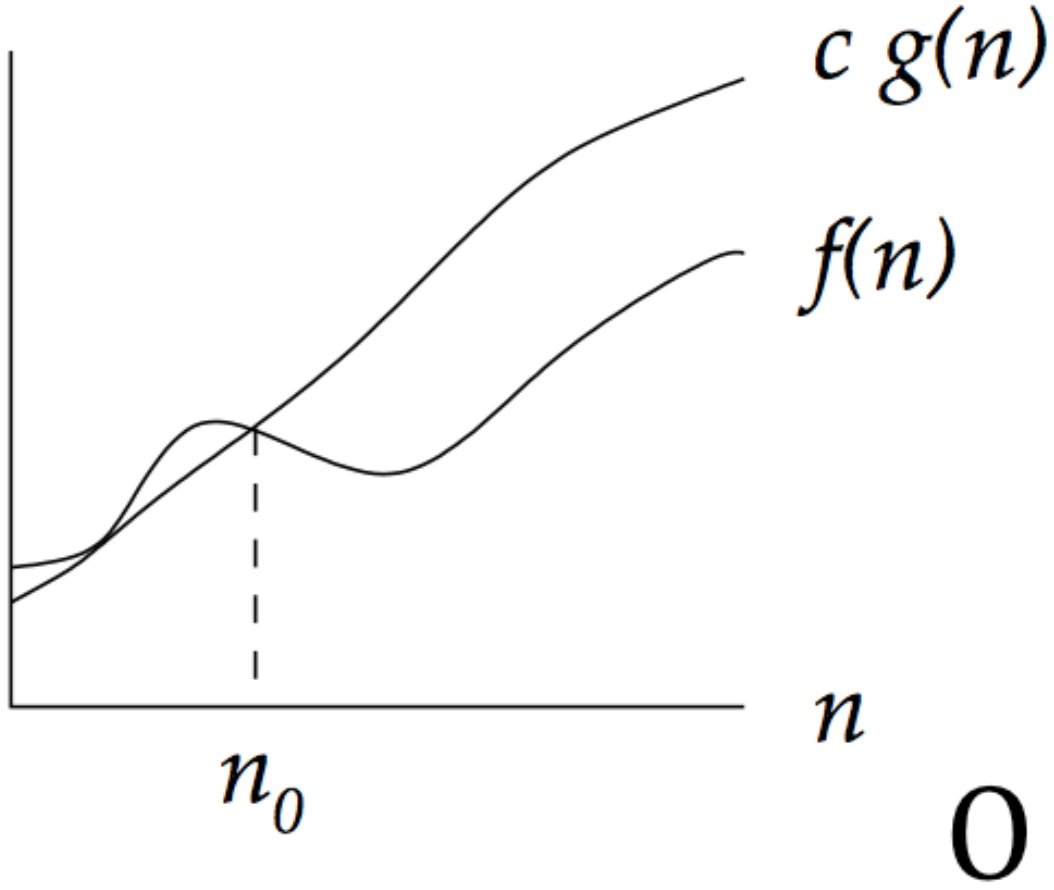
- toisin sanoen, jos polynomin eniten merkitsevän termin kerroin on positiivinen, Θ-merkintä sallii kaikkien muiden termien sekä e.m. kertoimen abstrahoinnin pois

Vakiofunktiolle pätee $c = \Theta(n^0) = \Theta(1)$

- $\Theta(1)$ ei kerro, minkä muuttujan suhteen funktioita tarkastellaan

  $\Rightarrow$ sitä saa käyttää vain kun muuttuja on asiayhteyden vuoksi selvä $\Rightarrow$ yleensä algoritmien tapauksessa on

O-merkintä



Kuva 9: O-merkintä

O-merkintä on muuten samanlainen kuin Θ-merkintä, mutta se rajaa funktion ainoastaan ylhäältä.

$\Rightarrow$ *asymptoottinen yläraja*

Määritelmä:

> *$O(g(n))$ on niiden funktioiden $f(n)$ joukko, joille on olemassa positiiviset vakiot c ja n_0 siten, että aina kun $n \geq n_0$, niin*
>
> $$0 \leq f(n) \leq c \cdot g(n)$$

- pätee: jos $f(n) = \Theta(g(n))$, niin $f(n) = O(g(n))$
- päinvastainen ei aina päde: $n^2 = O(n^3)$, mutta $n^2 \neq \Theta(n^3)$
- tärkeä tulos: jos $k \leq m$, niin $n^k = O(n^m)$
- jos **hitaimman** tapauksen suoritusaika on $O(g(n))$, niin **jokaisen** tapauksen suoritusaika on $O(g(n))$

Usein algoritmin hitaimman (ja samalla jokaisen) tapauksen suoritusajalle saadaan pelkällä vilkaisulla jokin O-merkinnällä ilmoitettava yläraja.

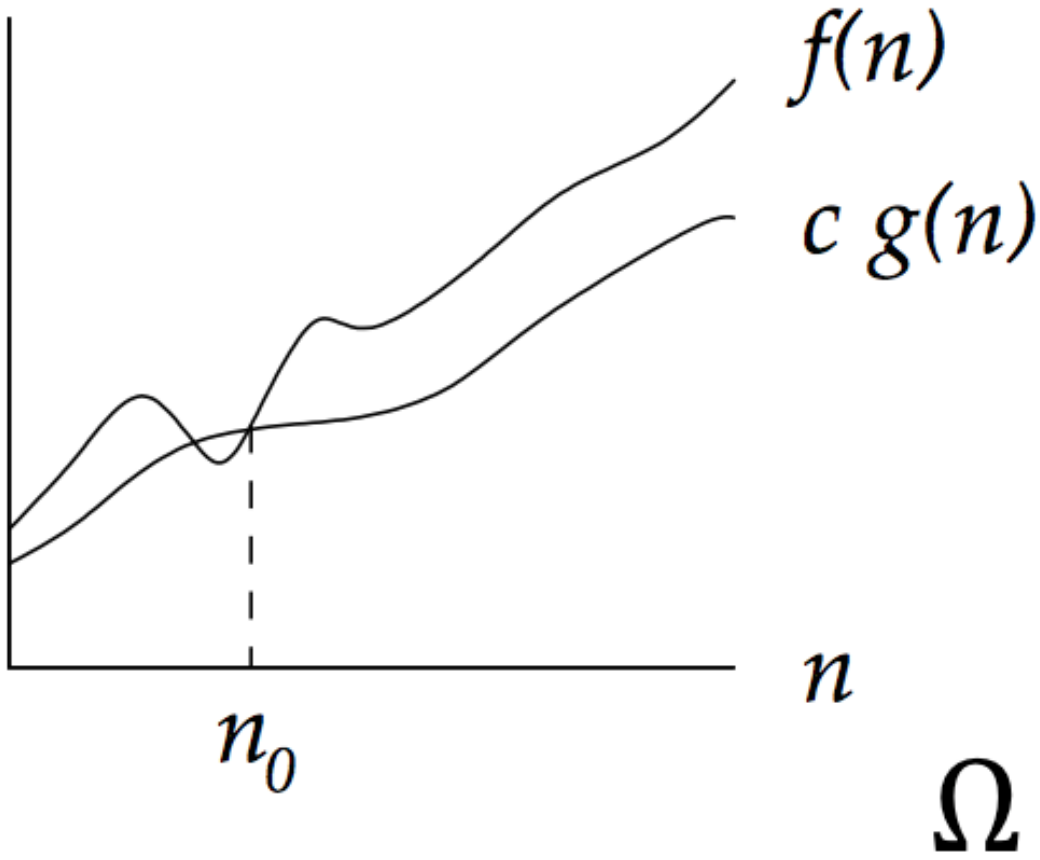
Usein vain yläraja onkin kiinnostava
$\Rightarrow$ O-merkinnällä on suuri käytännön merkitys

Esimerkki: INSERTION-SORT

rivi	kerta-aika
for j := 2 **to** $A.length$ **do**	$O(n)$
key := $A[j]$	$\cdot \quad O(1)$
i := $j-1$	$\cdot \quad O(1)$
while $i > 0$ and $A[i] > key$ **do**	$\cdot \quad O(n)$
$A[i+1]$:= $A[i]$	$\cdot \quad \cdot \quad O(1)$
i := $i-1$	$\cdot \quad \cdot \quad O(1)$
$A[i+1]$:= key	$\cdot \quad O(1)$

Jolloin pahimmalle tapaukselle saadaan suoritusaika
$O(n) \cdot O(n) \cdot O(1) = O(n^2)$

Ω-merkintä (äännetään “iso oomega”)



Kuva 10: Ω-merkintä

Ω-merkintä on muuten täysin samanlainen kuin Θ-merkintä, mutta se rajaa funktion vain alhaalta.

$\Rightarrow$ *asymptoottinen alaraja*

määritelmä:

> $\Omega(g(n))$ *on niiden funktioiden* $f(n)$ *joukko, joille on olemassa positiiviset vakiot* c *ja* n_0 *siten, että aina kun* $n \geq n_0$*, niin*
>
> $$0 \leq c \cdot g(n) \leq f(n)$$

- määritelmistä seuraa tärkeä tulos:
  $f(n) = \Theta(g(n))$ *jos ja vain jos* $f(n) = O(g(n))$ *ja* $f(n) = \Omega(g(n))$.
- jos **nopeimman** tapauksen suoritusaika on $\Omega(g(n))$, niin **jokaisen** tapauksen suoritusaika on $\Omega(g(n))$

Käytännön hyötyä Ω-merkinnästä on lähinnä tilanteissa, joissa jonkin ratkaisuvaihtoehdon parhaan tapauksenkin tehokkuus on epätyydyttävä, jolloin ratkaisu voidaan hylätä välittömästi.

Merkintöjen keskinäiset suhteet

$f(n) = \Omega(g(n))$ ja $f(n) = O(g(n)) \iff f(n) = \Theta(g(n))$

Kertaluokkamerkinnöillä on samankaltaisia ominaisuuksia kuin lukujen vertailuilla:

$$\begin{aligned} f(n) &= O(g(n)) \quad a \leq b \\ f(n) &= \Theta(g(n)) \quad a = b \\ f(n) &= \Omega(g(n)) \quad a \geq b \end{aligned}$$

Eli, jos $f(n)$:n korkeimman asteen termi, josta on poistettu vakiokerroin $\leq g(n)$: n vastaava, $f(n) = O(g(n))$ jne.

Yksi merkittävä ero: reaaliluvuille pätee aina tasan yksi kaavoista $a < b$, $a = b$ ja $a > b$, mutta vastaava ei päde kertaluokkamerkinnöille.

$\Rightarrow$ Kaikkia funktioita ei pysty mielekkäällä tavalla vertaamaan toisiinsa kertaluokkamerkintöjen avulla (esim. n ja $n^{1+\sin n}$).

Hieman yksinkertaisten:

- Jos algoritmi on $\Omega(g(n))$, sen resurssien kulutus on ainakin kertaluokassa $g(n)$.
  - vrt. kirja maksaa ainakin noin kympin.
- Jos algoritmi on $O(g(n))$, sen resurssien kulutus on korkeintaan kertaluokassa $g(n)$.
  - vrt. kirja maksaa korkeintaan noin kympin.
- Jos algoritmi on $\Theta(g(n))$, sen resurssien kulutus on aina kertaluokassa $g(n)$.
  - vrt. kirja maksaa suunnilleen kympin.

Kaikkien algoritmien kaikkien tapausten suoritusajalle ei välttämättä voida antaa mitään ratkaisua Θ-notaatiolla.

Esimerkkinä Insertion-Sort:

- paras tapaus on $\Omega(n)$, mutta ei $\Omega(n^2)$
- pahin tapaus on $O(n^2)$, mutta ei $O(n)$
  $\Rightarrow$ kaikille tapauksille yhteistä Θ-arvoa ei voida määrittää

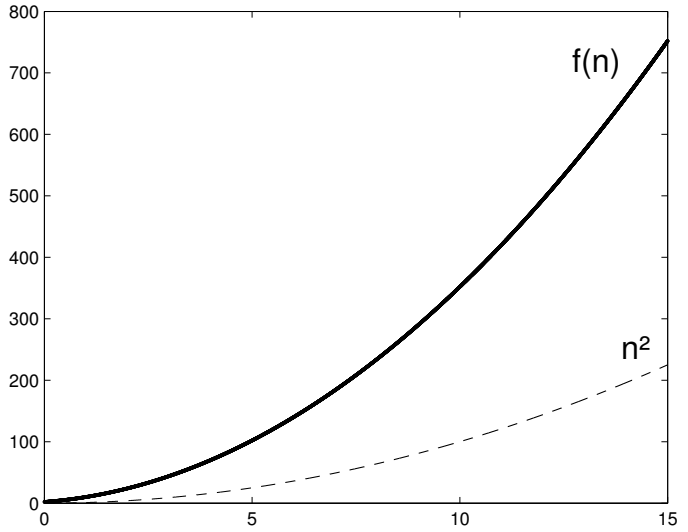
Esimerkki

Otetaan funktio $f(n) = 3n^2 + 5n + 2$.

Suoritetaan sille aiemmin sovitut yksinkertaistukset:

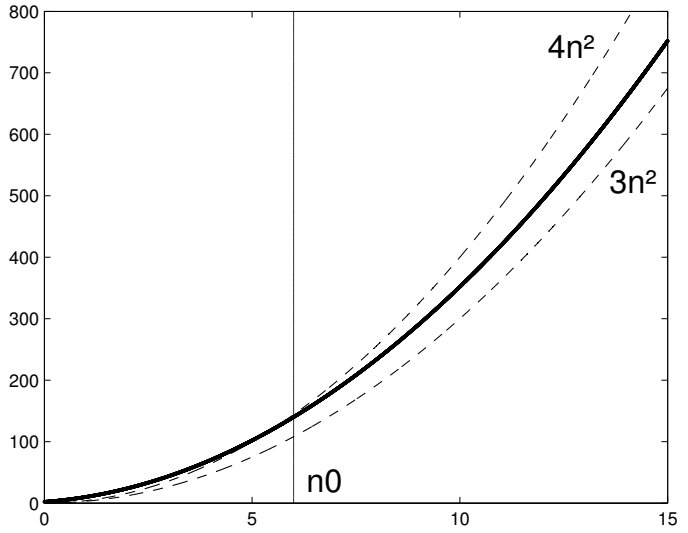
- alemman asteen termit pois
- vakiokertoimet pois

$\Rightarrow f(n) = \Theta(n^2)$



Vakuuttuaksemme asiasta etsimme kertoimet c_1 ja c_2:

$3n^2 \leq 3n^2 + 5n + 2 \leq 4n^2$, kun $n \geq 6$
$\Rightarrow c_1 = 3$, $c_2 = 4$ ja $n_0 = 6$ toimivat
$\Rightarrow f(n) = O(n^2)$ ja $\Omega(n^2)$
$\Rightarrow f(n) = \Theta(n^2)$

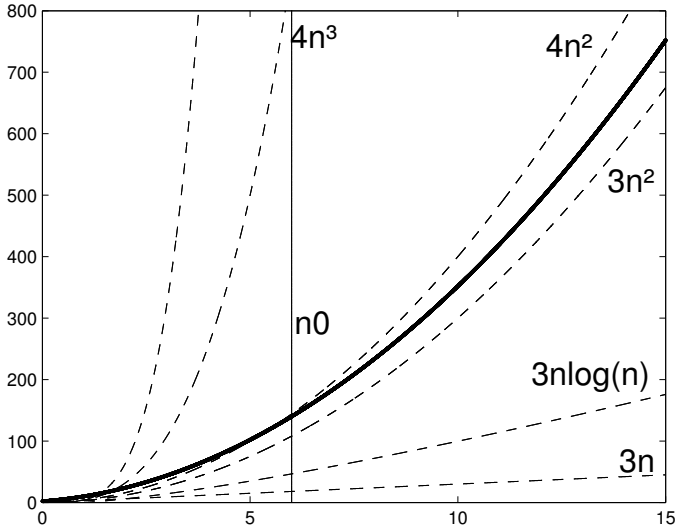


Selvästi kerroin $c_2 = 4$ toimii myös kun $g(n) = n^3$, sillä kun $n \geq 6, n^3 > n^2$
$\Rightarrow f(n) = O(n^3)$

- sama pätee kun $g(n) = n^4$...

Ja alapuolella kerroin $c_1 = 3$ toimii myös kun $g(n) = n \lg n$, sillä kun $n \geq 6, n^2 > n \lg n$
$\Rightarrow f(n) = \Omega(n \lg n)$

- sama pätee kun $g(n) = n$ tai $g(n) = \lg n$



5.2 Suorituskykykäsitteitä

Tähän mennessä algoritmien suorituskykyä on arvioitu lähinnä suoritusajan näkökulmasta. Muitakin vaihtoehtoja kuitenkin on:

- Voidaan mitata myös esimerkiksi muistinkulutusta tai kaistanleveyttä.

Lisäksi käytännössä tulee ottaa huomioon ainakin seuraavat seikat:

- Millä mittayksiköllä resurssien kulutusta mitataan?
- Miten syötekoko määritellään?
- Mitataanko huonoimman, parhaan vai keskimääräisen tapauksen resurssien kulutusta?
- Millaiset syötekoot tulevat kysymykseen?
- Riittääkö kertaluokkamerkintöjen tarkkuus vai tarvitaanko tarkempaa tietoa?

Ajoajan mittayksiköt

Valittaessa ajoajan mittayksikköä "askelta" pyritään yleensä mahdollisimman koneriippumattomaan ratkaisuun:

- Todelliset aikayksiköt kuten sekunti eivät siis kelpaa.
- Vakiokertoimet käyvät merkityksettömiksi.
  $\Rightarrow$ Jäljelle jää kertaluokkamerkintöjen tarkkuustaso.

$\Rightarrow$ askeleeksi voidaan katsoa mikä tahansa enintään vakioajan vievä operaatio.

- Vakioaikaiseksi tulkitaan mikä tahansa operaatio, jonka ajankulutus on riippumaton syötekoosta.
- Tällöin on olemassa jokin syötteen koosta riippumaton aikamäärä, jota operaation kesto ei milloinkaan ylitä.
- Yksittäisiä askeleita ovat esimerkiksi yksittäisen muuttujan sijoitus, **if**-lauseen ehdon testaus etc.
- Askeleen rajauksen kanssa ei tarvitse olla kovin tarkka, koska $\Theta(1) + \Theta(1) = \Theta(1)$.

Muistin käytön mittayksiköt

Tarkkoja yksiköitä ovat lähes aina bitti, tavu (8 bittiä) ja sana (jos sen pituus tunnetaan).

Eri tyyppien muistin käyttö on usein tunnettu, joskin se vaihtelee vähän eri tietokoneiden ja kielten välillä.

- kokonaisluku on yleensä 16 tai 32 bittiä
- merkki on yleensä 1 tavu = 8 bittiä
- osoitin on yleensä 4 tavua = 32 bittiä
- taulukko $A[1 \ldots n]$ on usein $n \cdot$ <alkion koko>

$\Rightarrow$ Tarkka muistin käytön arvioiminen on usein mahdollista, joskin huolellisuutta vaativaa.

Kertaluokkamerkinnät ovat käteviä silloin, kun tarkka tavujen laskeminen ei ole vaivan arvoista.

Jos algoritmi säilyttää yhtäaikaa koko syöteaineiston, niin arvioinnissa kannattaa erottaa syöteaineiston kuluttama muisti muusta muistin tarpeesta.

- $\Theta(1)$ lisämuistin tarve vs. $\Theta(n)$ lisämuistin tarve
- Kannattaa kuitenkin huomata, että esimerkiksi merkkijonon etsintä syötetiedostosta ei talleta yhtäaikaa koko syöteaineistoa, vaan selaa sen läpi.

6 Pikalajittelu ja satunnaistaminen

MERGE-SORTilla osaongelmiin jako oli helppoa ja ratkaisujen yhdistämisessä nähtiin paljon työtä.

Tutustutaan seuraavaksi keskimäärin erittäin nopeaan järjestämisalgoritmiin QUICKSORT, jossa työ tehdään jakovaiheessa.

QUICKSORTin kautta kohdataan uusi suunnitteluperiaate: *satunnaistaminen*.

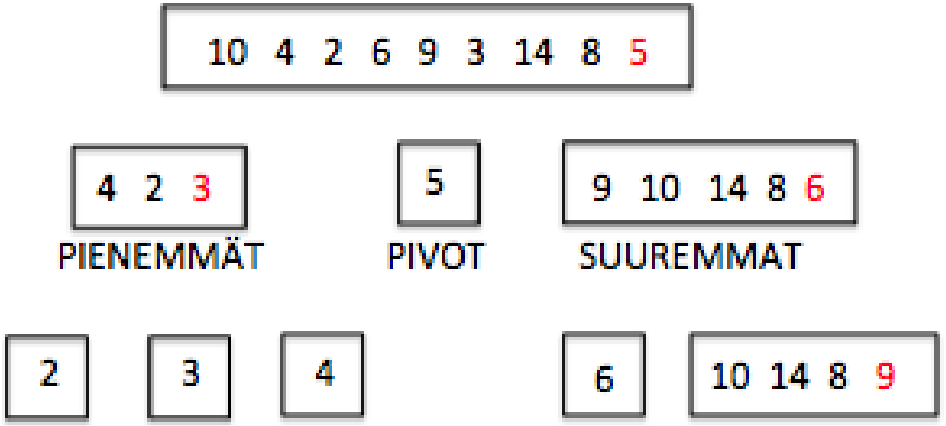
6.1 QUICKSORT

Ongelman jakaminen pienemmiksi osaongelmiksi

- Valitaan jokin taulukon alkioista jakoalkioksi eli *pivot-alkioksi*.
- Muutetaan taulukon alkioiden järjestystä siten, että kaikki jakoalkiota pienemmät tai yhtäsuuret alkiot ovat taulukossa ennen jakoalkiota ja suuremmat alkiot sijaitsevat jakoalkion jälkeen.
- Jatketaan alku ja loppuosien jakamista pienemmiksi, kunnes ollaan päästy 0:n tai 1:n kokoisiin osataulukoihin.

Kertaus: QUICKSORT-algoritmi

QUICKSORT($A, left, right$)
1 **if** $left < right$ **then** *(triviaalitapaukselle ei tehdä mitään)*
2 p := PARTITION($A, left, right$) *(muuten jaetaan alkuosaan ja loppuosaan)*
3 QUICKSORT($A, left, p-1$) *(järjestetään jakoalkiota pienemmät)*
4 QUICKSORT($A, p+1, right$) *(järjestetään jakoalkiota suuremmat)*



Kuva 11: Jako pienempiin ja suurempiin

Pienet osaongelmat:

- 0:n tai 1:n kokoiset osataulukot ovat valmiiksi järjestyksessä.

Järjestyksessä olevien osataulukoiden yhdistäminen:

- Kun alkuosa ja loppuosa on järjestetty on koko (osa)taulukko automaattisesti järjestyksessä.
  - kaikki alkuosan alkiothan ovat loppuosan alkioita pienempiä, kuten pitääkin

Ositus- eli *partitiointialgoritmi* jakaa taulukon paikallaan vaaditulla tavalla.

PARTITION($A, left, right$)

```
1  p := A[ right ]                    (otetaan pivotiksi viimeinen alkio)
2  i := left − 1                      (merkitään i:llä pienten puolen loppua)
3  for j := left to right − 1 do      (käydään läpi toiseksi viimeiseen alkioon asti)
4      if A[ j ] ≤ p                  (jos A[ j] kuuluu pienten puolelle...)
5          i := i + 1                 (... kasvatetaan pienten puolta...)
6          exchange A[ i ] ↔ A[ j ]   (... ja siirretään A[j] sinne)
7  exchange A[ i + 1 ] ↔ A[ right ]   (sijoitetaan pivot pienten ja isojen puolten väliin)
8  return i + 1                       (palautetaan pivot-alkion sijainti)
```

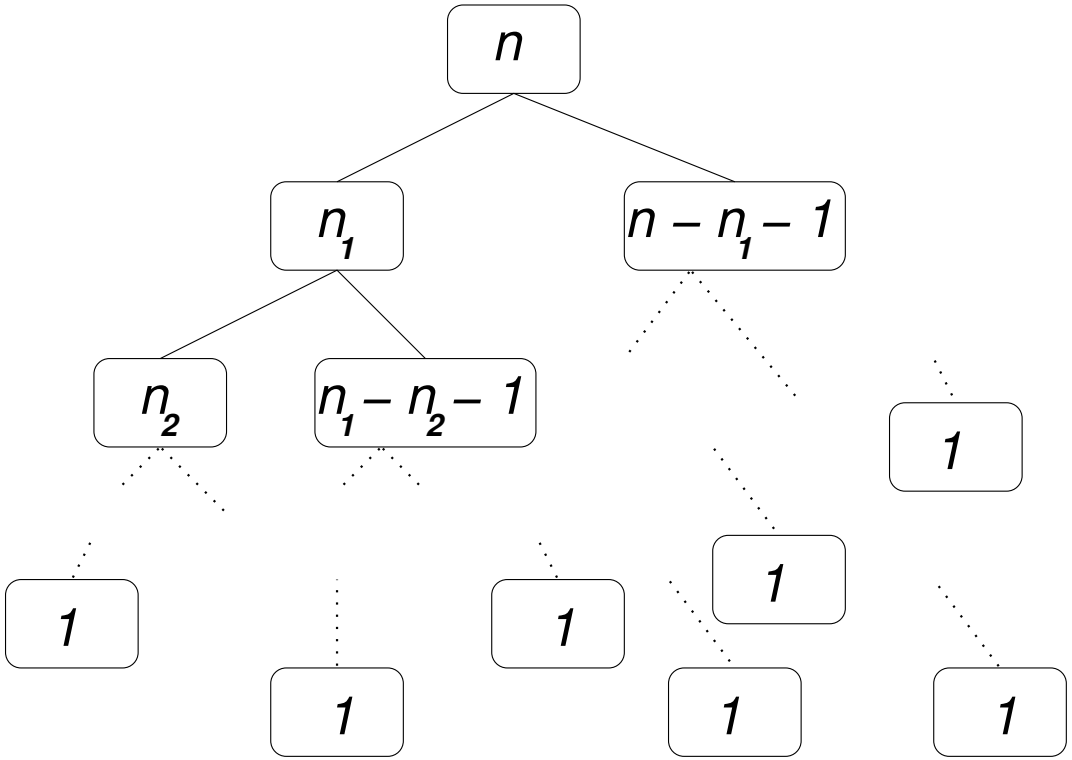
Kuinka nopeasti PARTITION toimii?

- **For**-silmukka tekee n - 1 kierrosta, kun n on $right$ - $left$
- Kaikki muut operaatiot ovat vakioaikaisia.

$\Rightarrow$ Sen suoritusaika on $\Theta(n)$.

Kuten MERGE-SORTilla QUICKSORTin analyysi ei ole yhtä suoraviivainen rekursion vuoksi

- Koska PARTITIONIA ja rekursiivista kutsua lukuunottamatta kaikki QUICKSORTIN operaatiot ovat vakioaikaisia, keskitymme tarkastelemaan PARTITIONIN instanssien käyttämää aikaa.



- Kokonaisaika on edellisen kuvan esittämän puun solmujen aikojen summa.
- 1 kokoiselle taulukolle suoritus on vakioaikaista.
- Muille suoritus on lineaarista osataulukon kokoon nähden.
  $\Rightarrow$ Kokonaisaika on siis Θ(solmujen numeroiden summa).

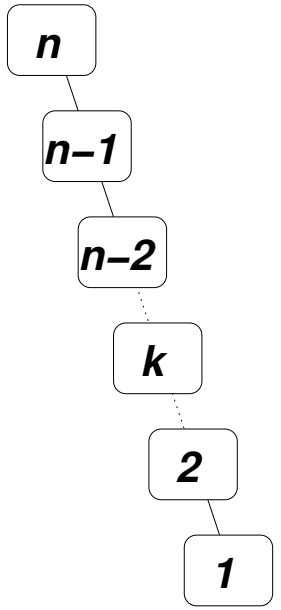
Pahimman tapauksen suoritusaika

- Solmun numero on aina pienempi kuin isäsolmun numero, koska jakoalkio on jo oikealla paikallaan, eikä se kuulu kumpaankaan järjestettävään osataulukkoon

  $\Rightarrow$ puussa voi siis olla kerroksia enintään n kappaletta

- pahin tapaus realisoituu, kun jakoalkioksi valitaan aina pienin tai suurin alkio
  - näin tapahtuu esimerkiksi valmiiksi järjestetyllä taulukolla
- solmujen numeroiden summa on n + n - 1 + $\cdots$ + 2 + 1

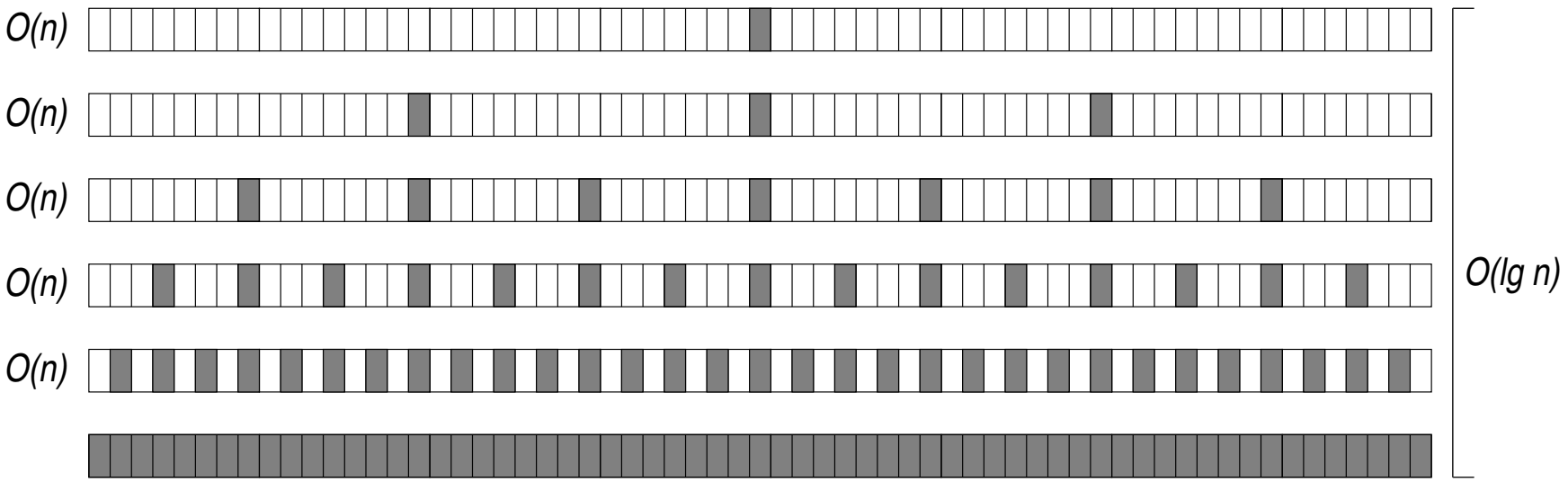
$\Rightarrow$ QUICKSORTIN suoritusaika on $O(n^2)$



Paras tapaus realisoituu kun taulukko jakautuu aina tasan.

- Alla oleva kuva näyttää kuinka osataulukoiden koot pienenevät.
  - harmaat ruudut ovat jo oikealla paikallaan olevia alkioita
- Jokaisella tasolla tehtävä työ on kertaluokassa $\Theta(n)$.
  - tästä saamme pessimistisen arvion suorituspuun korkeudelle parhaassa tapauksessa $\Rightarrow O(\lg n)$

  $\Rightarrow$ Parhaan tapauksen suoritusajan yläraja on $O(n \lg n)$.



QUICKSORTIN kohdalla parhaan ja huonoimman tapauksen suoritusajat eroavat toisistaan selvästi.

- Olisi mielenkiintoista tietää keskimääräinen suoritusaika.
- Sen analysoiminen menee tämän kurssin tavoitteiden ulkopuolelle, mutta on osoitettu, että jos aineisto on tasajakautunutta, keskimääräinen suoritusaika on $\Theta(n \lg n)$.
- Keskimääräinen suoritusaika on siis varsin hyvä.

QUICKSORTIIN liittyy kuitenkin se kiusallinen seikka, että sen pahimman tapauksen suoritus on hidasta ja sen esiintyminen on käytännössä varsin todennäköistä.

- On helppoa kuvitella tilanteita, joissa aineisto on jo järjestyksessä tai melkein järjestyksessä.

$\Rightarrow$ Tarvitaan keino, jolla pahimman tapauksen systemaattisen esiintymisen riskiä saadaan pienennettyä.

Satunnaistaminen on osoittautunut tässä varsin tehokkaaksi.

QUICKSORTIN etuja ja haittoja

Etuja:

- järjestää taulukon keskimäärin erittäin tehokkaasti
  - ajoaika on keskimäärin $\Theta(n \lg n)$
  - vakiokerroin on pieni
- tarvitsee vain vakiomäärän lisämuistia
- sopii hyvin virtuaalimuistiympäristöön
- käyttää tehokkaasti välimuistia

Haittoja:

- ajoaika on hitaimmillaan $\Theta(n^2)$
- hitaimman tapauksen tuottava syöte on kiusallisen tavallinen ilman satunnaistamista
- rekursiivisuus
  $\Rightarrow$ tarvitsee lisämuistia pinolle
- epävakaus