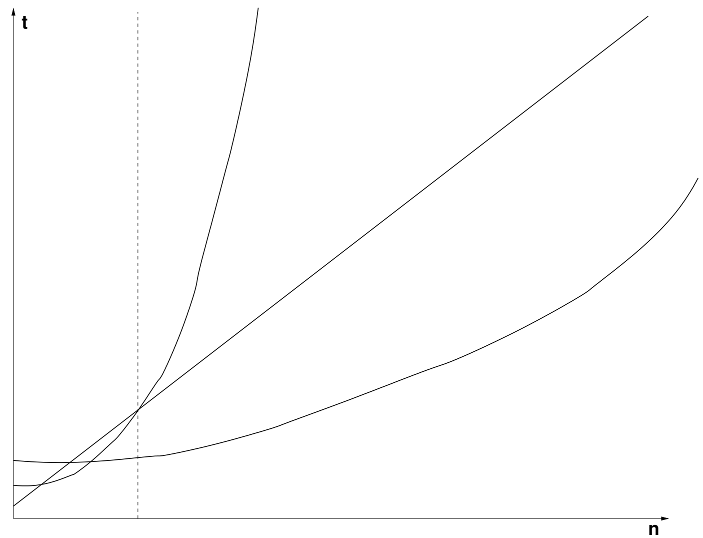
4 Tehokkuus ja algoritmien suunnittelu

Tässä luvussa pohditaan tehokkuuden käsitettä ja esitellään kurssilla käytetty kertaluokkanotaatio, jolla kuvataan algoritmin *asymptoottista* käyttäytymistä eli tapaa, jolla algoritmin resurssien kulutus muuttuu syötekoon kasvaessa.



4.1 Kertaluokat

Algoritmin analysoinnilla tarkoitetaan sen kuluttamien resurssien määrän arvioimista

Tyypillisesti analysoidaan syötekoon kasvun vaikutusta algoritmin resurssien kulutukseen

Useimmiten meitä kiinnostaa algoritmin ajankäytön kasvu syötteen koon kasvaessa

- Voimme siis tarkastella ajankäyttöä irrallaan toteutusympäristöstä
- Itse asiassa voimme kuvata periaatteessa minkä tahansa peräkkäisiä operaatioita sisältävän toiminnan ajankulutusta

- **Algoritmin ajankäyttö**:
  Algoritmin suorittamien "askelten" suorituskertojen määrä
- **Askel**:
  syötekoosta riippumattoman operaation viemä aika.
- Emme välitä siitä, kuinka monta kertaa jokin operaatio suoritetaan kunhan se tehdään vain vakiomäärä kertoja.
- Tutkimme kuinka monta kertaa algoritmin suorituksen aikana kukin rivi suoritetaan ja laskemme nämä määrät yhteen.

– Yksinkertaistamme vielä tulosta poistamalla mahdolliset vakiokertoimet ja alemman asteen termit.

$\Rightarrow$ Näin voidaan tehdä, koska syötekoon kasvaessa riittävästi alemman asteen termit käyvät merkityksettömiksi korkeimman asteen termin rinnalla.

$\Rightarrow$ Menetelmä ei luonnollisestikaan anna luotettavia tuloksia pienillä syöteaineistoilla, mutta niillä ohjelmat ovat tyypillisesti riittävän tehokkaita joka tapauksessa.

– Kutsumme näin saatua tulosta algoritmin ajan kulutuksen kertaluokaksi, jota merkitään kreikkalaisella kirjaimella Θ (äännetään "theeta").

$$f(n) = 23n^2 + 2n + 15 \Rightarrow f \in \Theta(n^2)$$

$$f(n) = \frac{1}{2}n \lg n + n \Rightarrow f \in \Theta(n \lg n)$$

Esimerkki 1: taulukon alkioiden summaus

```
1  for i := 1 to A.length do
2      summa := summa + A[i]
```

- jos taulukon A pituus (syötekoko) on n, rivi 1 suoritetaan n + 1 kertaa
- rivi 2 suoritetaan n kertaa
- ajankulutus kasvaa siis n:n kasvaessa seuraavalla tavalla:

n	aika = 2n + 1
1	3
10	21
100	201
1000	2001
10000	20001

$\Rightarrow$ n:n arvo hallitsee ajankulutusta

– suoritamme edellä sovitut yksinkertaistukset: poistamme vakiokertoimen ja alemman asteen termin:

$$f(n) = 2n + 1 \Rightarrow n$$

$\Rightarrow$ saamme tulokseksi $\Theta(n)$

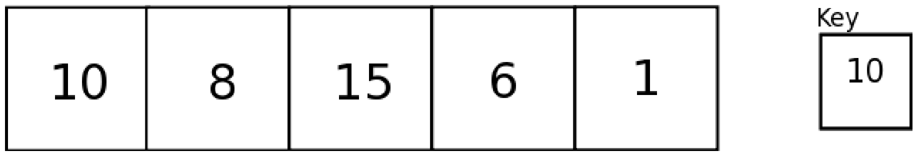
$\Rightarrow$ kulutettu aika riippuu *lineaarisesti* syötteen koosta

Esimerkki 2: alkion etsintä järjestämättömästä taulukosta

```
for i := 1 to A.length do
    if A[i] = key then
        return i
```

- tässä tapauksessa suoritusaika riippuu syöteaineiston koon lisäksi sen koostumuksesta eli siitä, mistä kohtaa taulukkoa haluttu alkio löytyy
- On tutkittava erikseen:
  paras,
  huonoin ja
  keskimääräinen tapaus

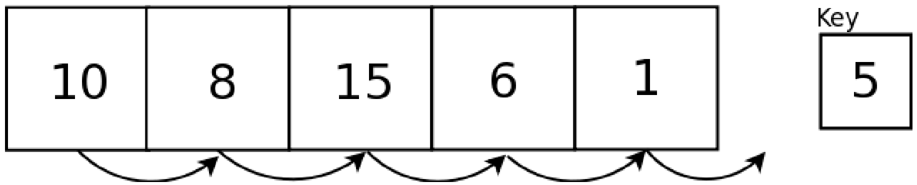
– paras tapaus:



Kuva 6: Etsintä: paras tapaus, löytyy ensimmäisestä alkiosta

$\Rightarrow$ alkio löytyy *vakioajassa* eli ajankulutus on $\Theta(1)$

– huonoin tapaus



Kuva 7: Etsintä: huonoin tapaus, löytyy viimeisestä tai ei ollenkaan

rivi 1 suoritetaan n + 1 kertaa ja rivi 2 n kertaa
$\Rightarrow$ suoritusaika on *lineaarinen* eli $\Theta(n)$.

- keskimääräinen tapaus:
  täytyy tehdä jonkinlainen oletus tyypillisestä eli keskimääräisestä aineistosta:
  * alkio on taulukossa todennäköisyydellä p $(0 \leq p \leq 1)$
  * ensimmäinen haettu alkio löytyy taulukon jokaisesta kohdasta samalla todennäköisyydellä
- voimme laskea suoraan todennäköisyyslaskennan avulla, kuinka monta vertailua keskimäärin joudutaan tekemään

- todennäköisyys sille, että alkio ei löydy taulukosta on 1 - p
  $\Rightarrow$ joudutaan tekemään n vertailua (huonoin tapaus)
- todennäköisyys sille, että alkio löytyy kohdasta i, on p/n
  $\Rightarrow$ joudutaan tekemään i vertailua
- odotusarvoinen tarvittavien vertailujen määrä saadaan siis seuraavasti:

$$[1 \cdot \frac{p}{n} + 2 \cdot \frac{p}{n} + \cdots + i \cdot \frac{p}{n} \cdots + n \cdot \frac{p}{n}] + n \cdot (1 - p)$$

– oletamme, että alkio varmasti löytyy taulukosta eli $p = 1$, saamme tulokseksi $(n+1)/2$ eli $\Theta(n)$
$\Rightarrow$ koska myös tapaus, jossa alkio ei löydy taulukosta, on ajankäytöltään lineaarinen, voimme olla varsin luottavaisia sen suhteen, että keskimääräinen ajankäyttö on kertaluokassa $\Theta(n)$

– kannattaa kuitenkin muistaa, että läheskään aina kaikki syötteet eivät ole yhtä todennäköisiä
$\Rightarrow$ jokaista tapausta on syytä tutkia erikseen

Esimerkki 3: kahden taulukon yhteisen alkion etsintä

1 **for** i := 1 **to** $A.length$ **do**
2   **for** j := 1 **to** $B.length$ **do**
3   **if** $A[i] = B[j]$ **then**
4   **return** $A[i]$

- rivi 1 suoritetaan 1 .. (n + 1) kertaa
- rivi 2 suoritetaan 1 .. ($n \cdot$ (n + 1)) kertaa
- rivi 3 suoritetaan 1 .. ($n \cdot n$) kertaa
- rivi 4 suoritetaan korkeintaan kerran

- nopeimmillaan algoritmi on siis silloin kun molempien taulukoiden ensimmäinen alkio on sama

  $\Rightarrow$ parhaan tapauksen ajoaika on $\Theta(1)$
- pahimmassa tapauksessa taulukoissa ei ole ainuttakaan yhteistä alkiota tai ainoastaan viimeiset alkiot ovat samat

  $\Rightarrow$ tällöin suoritusajaksi tulee *neliöllinen* eli $2n^2 + 2n + 1 = \Theta(n^2)$
- keskimäärin voidaan olettaa, että molempia taulukoita joudutaan käymään läpi noin puoleen väliin

  $\Rightarrow$ tällöin suoritusajaksi tulee $\Theta(n^2)$
  (tai $\Theta(nm)$ mikäli taulukot ovat eri mittaisia)

Palataan INSERTION-SORTiin. Sen ajankäyttö:

```
INSERTION-SORT( A )                              (syöte saadaan taulukossa A)
1  for j := 2 to A.length do                     (siirretään osien välistä rajaa)
2      key := A[ j ]                             (otetaan alkuosan uusi alkio käsittelyyn)
3      i := j − 1
4      while i > 0 and A[ i ] > key do           (etsitään uudelle alkiolle oikea paikka)
5          A[ i + 1 ] := A[ i ]                  (raivataan uudelle alkiolle tilaa)
6          i := i − 1
7      A[ i + 1 ] := key                         (asetetaan uusi alki o oikealle paikalleen)
```

– rivi 1 suoritetaan n kertaa

– rivit 2 ja 3 suoritetaan n - 1 kertaa

– rivi 4 suoritetaan vähintään n - 1, enintään (2 + 3 + 4 + $\cdots$ + n - 2) kertaa

– rivit 5 ja 6 suoritetaan vähintään 0, enintään (1 + 2 + 3 + 4 + $\cdots$ + n - 3) kertaa

- parhaassa tapauksessa, kun taulukko on valmiiksi järjestyksessä, koko algoritmi siis kuluttaa vähintään $\Theta(n)$ aikaa
- huonoimmassa tapauksessa, kun taulukko on käänteisessä järjestyksessä, aikaa taas kuluu $\Theta(n^2)$
- keskimääräisen tapauksen selvittäminen on jälleen vaikeampaa:
  * oletamme, että satunnaisessa järjestyksessä olevassa taulukossa olevista elementtipareista puolet ovat keskenään epäjärjestyksessä.

    $\Rightarrow$ vertailuja joudutaan tekemään puolet vähemmän kuin pahimmassa tapauksessa, jossa kaikki elementtiparit ovat keskenään väärässä järjestyksessä

    $\Rightarrow$ keskimääräinen ajankulutus on pahimman tapauksen ajankäyttö jaettuna kahdella: ((n - 1)n)/ 4 = $\Theta(n^2)$