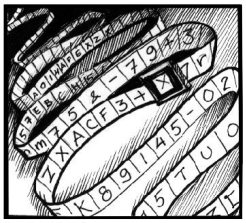


Mitä tekemistä logaritmeilla on tietokoneiden kanssa?

Pekka Kilpeläinen
Kuopion yliopisto
Tietojenkäsittelytieteen ja sovelletun matematiikan laitos

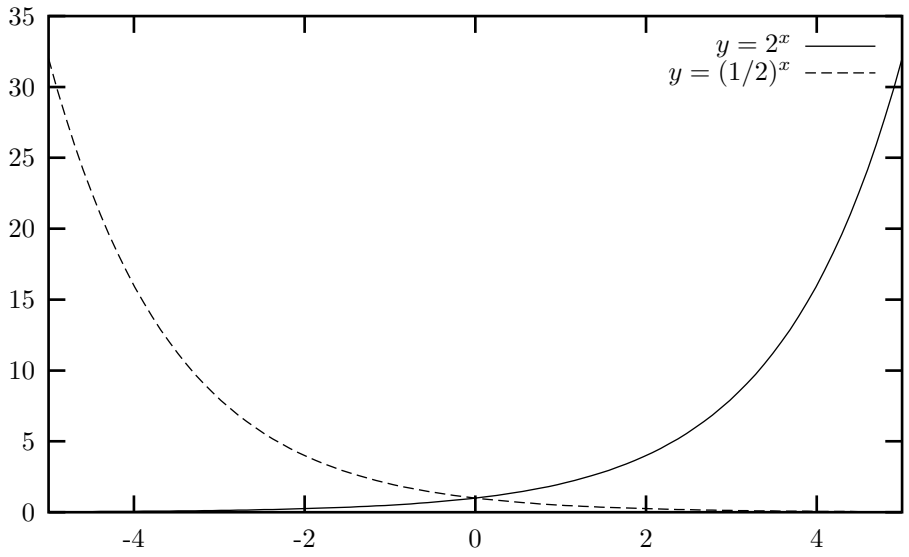
Eräs opiskelija kysyi pitämälläni Algoritmien suunnittelun ja analysoinnin luennolla: ”Mitä tekemistä logaritmeilla on tietokoneiden kanssa?” Arvelen hänen kysymyksensä heijastavan sitä monien opiskelijoiden opintoja haittaavaa käsitystä, että matematiikka olisi tietojenkäsittelyn kannalta hyödytöntä. Asia on kuitenkin päinvastoin: matematiikka on tietojenkäsittelyilmiöiden kunnolliselle ymmärtämiselle hyödyllistä ja osin jopa välttämätöntä. Koska lisäksi juuri logaritmifunktio on tietojenkäsittelyn kannalta varsin keskeinen, pyrin valaisemaan kysyttyä asiaa muutamalla yksinkertaisella esimerkillä tiedon esittämiseen tarvittavasta tilasta ja tiedon käsittelemiseen tarvittavasta työmäärästä.

Mitäs ne logaritmit olivatkaan?

Eksponenttifunktio $f(x) = b^x$ on määritelty kaikilla reaaliluvuilla x ja jokaisella kantalukuna toimivalla positiivisella reaaliluvulla b. Tapaus $b = 1$ on melko mielenkiinnoton, koska $1^x = 1$, mutta muulloin b-kantainen eksponentti on injektiivinen ja kaikki positiiviset reaaliarvot saava funktio. (Katso kuva 1.) Positiivisille reaaliluvuille x määritelty logaritmifunktio $\log_b x$ on tällöin b-kantaisen eksponentin käänteisfunktio, eli $\log_b x$ on se yksikäsitteinen luku y, jolla $b^y = x$. Toisin sanoen $b^{\log_b x} = x$, eli $\log_b x$ on se potenssi, johon kantaluku b on korotettava tuloksen x saamiseksi.

Logaritmi on sikäli mukava operaattori, että se muuttaa argumenttinaan olevan lausekkeen laskutoimituksia helpommiksi: kertolaskusta tulee yhteenlasku ($\log_b(xy) = \log_b x + \log_b y$), jakolaskusta tulee vähennyslasku ($\log_b \frac{x}{y} = \log_b x - \log_b y$), ja potenssiinkorotus muuttuu kertolaskuksi ($\log_b x^y = y \log_b x$).

Logaritmin kantaluku voi siis olla melkein mikä tahansa. Matematiikassa tarkastellaan useimmin *luonnollista logaritmia* $\ln x = \log_e x$, jonka kantaluku on ns. Neperin luku $e \approx 2{,}718$. Luonnontieteissä usein luonteva logaritmin kantaluku on 10, kun taas tietojenkäsittelyssä kaksikantainen logaritmi on usein kätevin. Logaritmin kantaluvulla ei itse asiassa ole kovin suurta väliä: Logaritmifunktiot ovat kasvavia kaikilla positiivisilla kantaluvuilla ja poikkeavat tällöin toisistaan vain vakiokertoimella, joka on toisen logaritmin arvo toisen kantaluvusta:



Kuva 1: Eksponenttifunktioiden kuvaajia.

$\log_a x = \frac{1}{\log_b a} \log_b x$. Tämä tarkoittaa esimerkiksi sitä, että kaksikantaisen logaritmin $\log_2 x$ arvo on luonnollisen logaritmin $\ln x$ arvoon verrattuna $1/\ln 2$- eli likimain $1/0{,}693 = 1{,}44$-kertainen. (Katso kuva 2.)

Tietojenkäsittelyn kannalta tärkeä logaritmifunktion ominaisuus on sen hidas kasvuvauhti, jonka voi havaita kuvasta 2. Ominaisuutta voi perustella myös logaritmifunktion derivaatalla. Tunnetusti $D \ln x = \frac{1}{x}$. Derivaatan arvohan annetussa pisteessä vastaa funktion kuvaajalle kyseiseen pisteeseen piirretyn tangentin kulmakerrointa. Suurilla muuttujan x arvoilla logaritmifunktion tangentin kulmakerroin $1/x$ lähestyy nollaa, eli kuten kuvasta 2 nähdään, logaritmifunktion kasvu alkaa muistuttaa vakiofunktion (olematonta) kasvua. Näemme jatkossa esimerkkejä siitä, että käytännössä esiintyvien lukujen logaritmit ovat usein varsin pieniä. Tästä huolimatta on syytä muistaa, että argumentin kasvaessa myös logaritmifunktion arvo kasvaa rajoittamattoman suureksi.

Konkretisoidaan vielä logaritmin kasvuvauhtia muutamalla esimerkillä kaksikantaisesta logaritmista. Eräs perustelu sen hitaalle kasvuvauhdille on edellä mainittu logaritmin kertolaskun yhteenlaskuksi muuttava käyttäytyminen: $\log_2 2x = \log_2 2 + \log_2 x = 1 + \log_2 x$. Argumentin kaksinkertaistaminen kasvattaa kaksikantaisen logaritmin arvoa siis vain ykkösellä. Konkreettisina esimerkkeinä todetaan vaikka, että luvun 1000 kaksikantainen logaritmi on hieman alle 10, sillä $2^{10} = 1024$.[1] Edelleen on helppo päätellä, että myös miljoonan ja miljardin logaritmit ovat vielä suhteellisen pieniä: $\log_2 1\,000\,000 = \log_2 1000^2 = 2 \log_2 1000 \approx 2 \cdot 10 = 20$, ja $\log_2 10^9 = \log_2(1000 \cdot 10^6) = \log_2 1000 + \log_2 1\,000\,000 \approx 10 + 20 = 30$.

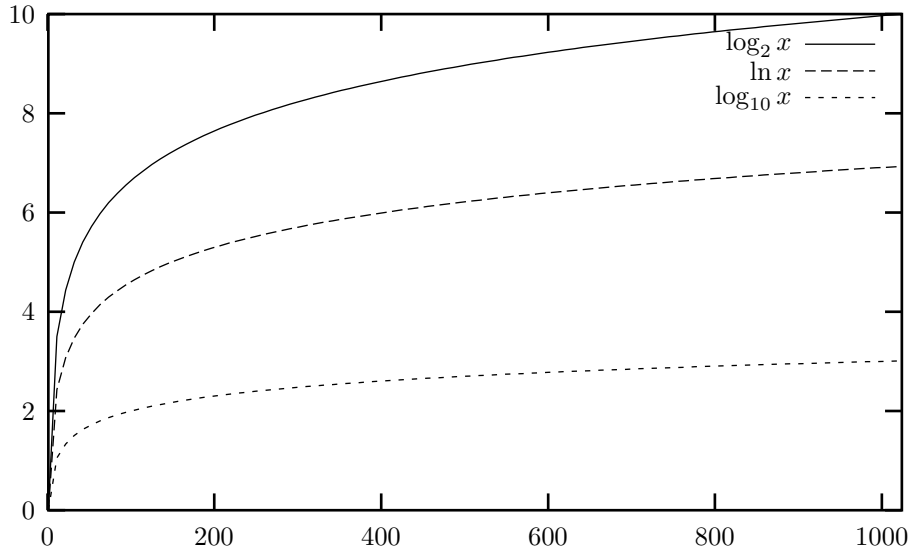
Logaritmi, algoritmi, biorytmi ...?

Algoritmi kuulostaa lähes samalta kuin ”logaritmi”: sanat saadaan toisistaan siirtämällä kaksi kirjainta uuteen paikkaan. Mitä sitten algoritmit ovat, ja onko niillä ja logaritmeilla jotain järkevääkin yhteyttä?

Algoritmeilla tarkoitetaan tietojenkäsittelyongelmien täsmällisiä ratkaisumenetelmiä. Jokaisen tietokoneohjelman ytimenä on jonkinlainen algoritmi. Algoritmitutkimus on tietojenkäsittelytieteen keskeinen ala, jonka käytännöllisenä tavoitteena on kehittää tietojenkäsittelyongelmille hyödyllisiä ratkaisualgoritmeja. Tyypillinen tarkastelun kohde on esimerkiksi se, miten tietoa järjestetään tai etsitään tehokkaalla tavalla eri tilanteissa.

Tietokoneohjelmissa tai -laitteissa käyttöön otettavien algoritmien pitäisi olla ”hyviä”. Algoritmin tulee tietenkin toimia oikein eli suorittaa virheettömästi sitä tehtävää, johon se on kehitetty. Mitä tämän lisäksi pidetään hyvänä voi vaihdella tilanteesta toiseen, mutta yleensä tavoitellaan jossain mielessä tehokkaita ratkaisuja. Tavallisimpia

[1] Kaksijärjestelmän keskeisyydestä tietokoneissa johtuu, että yleensä tuhatkertaisuutta tarkoittava etuliite ”kilo” tarkoittaa tietotekniikassa juuri arvoa $2^{10} = 1024$.



Kuva 2: Logaritmifunktioiden kuvaajia.

algoritmien tehokkuusmittareita ovat tiedon käsittelyyn tarvittu *aika* ja toisaalta tiedon esittämiseen tarvittu *muistitila*. Tehokkaimpia ovat algoritmit, jotka toimivat nopeimmin ja vaativat vähiten muistitilaa.

Yleensä algoritmit ovat ratkaisuja periaatteessa saman ongelman hieman erilaisille ja erikokoisille tapauksille. Ajatellaan esimerkkinä vaikka jonkin henkilön etsimistä puhelinluettelosta. Nimeä voi etsiä täsmälleen samalla tavalla vaikkapa Helsingin, Heinolan tai Hauhon puhelinluettelosta – yleinen menetelmä toimii kaikissa tapauksissa, vaikka luetteloiden sisällöt ovat aivan erilaiset. Toisaalta nimen etsiminen isommasta luettelosta voi arvatenkin olla työläämpää kuin sen paikantaminen pienemmästä joukosta nimiä. Tämän takia algoritmien tehokkuutta ei ilmoiteta kiinteinä absoluuttisina arvoina.

Algoritmianalyysissä pyritään matemaattisiin lausekkeisiin, jotka kuvaavat algoritmin tekemää työmäärää suhteessa käsiteltävän *tapauksen kokoon*. Puhelinluetteloesimerkissä luonteva ongelman tapauksen kokoa kuvaava parametri n voisi olla puhelinluettelossa mainittujen nimien lukumäärä. Yksinkertainen (mutta typerä) tapa etsiä annetun henkilön puhelinnumeroa olisi lukea luetteloa alusta alkaen nimi kerrallaan kunnes nimi löytyy tai luettelo loppuu. Pahimmillaan tällaisessa *peräkkäishaussa* tutkitaan kaikki luettelon n nimeä. Kyseisen algoritmin *aikavaativuuden* sanotaan olevan *lineaarinen* (suhteessa nimien lukumäärään n). Luettelon piteneminen kaksinkertaiseksi vaatii peräkkäishaussa pahimmillaan kaksinkertaisen etsintätyön.

Palataan puhelinluetteloetsintään ja sen tehokkaampaan suorittamiseen *logaritmisella* määrällä suoritusaskeleita hetken kuluttua. Tarkastellaan ensin kokonaislukujen esityksen pituutta, sillä kyseisestä tarkastelusta on hyötyä myös etsintäongelman työläyden arvioinnissa.

Kuinka pitkä on budjetin loppusumma?

Kokonaisluvut ovat keskeisimpiä informaation esittämisen välineitä. Niillä voi laskea lukumääriä tai nimetä mielivaltaisia asioita tyyliin ”ensimmäinen”, ”toinen”, jne. Tutustumme nyt kokonaislukujen pituuden ja niiden logaritmien läheiseen yhteyteen.

Tutulla kymmenjärjestelmällä voimme esittää mielivaltaisia kokonaislukuja, vaikka meillä on käytössämme ainoastaan merkit $0, 1, 2, \ldots, 9$. Tämä perustuu käyttämäämme positionaaliseen luvunesitykseen, jossa numerot edustavat sijaintinsa mukaan lukujärjestelmän kantaluvun eri potensseja: vähiten merkitsevät eli oikeanpuoleiset numerot ovat ykkösiä, seuraavat kymppejä, kolmannet satoja jne. Näin esimerkiksi valtion vuoden 2001 budjetin tulojen kokonaismäärä 209 172 310 000 mk tarkoittaa arvoa $0 + 0 \cdot 10 + \ldots + 1 \cdot 10^4 + 3 \cdot 10^5 + \ldots + 2 \cdot 10^{11}$ mk.

Kymmenjärjestelmän kantaluku lienee peräisin ihmislajin sormien lukumäärästä. Täsmälleen samaa ideaa voi kuitenkin käyttää myös muilla kantaluvuilla. Jokaisella ykköstä suuremmalla kokonaisluvulla b voidaan nimit-

täin määritellä *b-kantainen lukuesitys* $(d_{m-1}d_{m-2}\ldots d_1d_0)_b$, missä kukin d_0, d_1, …, d_{m-1} on jokin numeroista $0, 1, \ldots, b-1$. Tällainen luvunesitys tarkoittaa kokonaislukua $d_0 + d_1 \cdot b^1 + \ldots + d_{m-2} \cdot b^{m-2} + d_{m-1} \cdot b^{m-1}$. Erityisesti tietokoneita on käytännöllistä rakentaa siten, että niiden elektroniikka operoi kymmenen sijasta vain kahdella toisistaan erottuvalla tilalla. Siksi tietokoneet käyttävät *binääristä* luvunesitystä, jonka kantaluku b on 2 ja jossa käytettävät numerot ovat *bittejä* 0 ja 1.

Paljonko tilaa kokonaisluvun n esittäminen vaatii? Tarkastellaan kokonaisluvun n esitystä b-järjestelmän lukuna $(d_{m-1}d_{m-2}\ldots d_1d_0)_b$. Mitä voidaan sanoa tämän esityksen pituudesta m? Käytetään apuna merkintätapaa, jossa osoitamme jokaisen numeron sijainnin alaindeksinä $0, \ldots, m-1$. Esimerkiksi budjetin loppusumma on tällä esityksellä $(2_{11}0_{10}9_91_87_72_63_51_40_30_20_10_0)_{10}$, ja $(1_{m-1}0_{m-2}\ldots 0_0)_2$ tarkoittaa m-numeroista binäärilukua, jonka merkitsevin numero on ykkönen ja muut nollia.

Jos luku $n = (d_{m-1}d_{m-2}\ldots d_1d_0)_b > 0$ on aidosti m-numeroinen, niin sen merkitsevin numero d_{m-1} on vähintään ykkönen. Siten

$$(1_{m-1}0_{m-2}\ldots 0_0)_b \leq n \ .$$

Ylläolevan epäyhtälön vasemman puolen arvo on b^{m-1}. Soveltamalla epäyhtälöön b-kantaista logaritmia näemme, että $m-1 \leq \log_b n$, eli esityksen pituus m on enintään $\log_b n+1$. Toisaalta jokainen luvun n numeroista $d_{m-1}, \ldots, d_0$ on enintään $b-1$, joten näemme seuraavaa:

$$\begin{aligned}(d_{m-1}d_{m-2}\ldots d_1d_0)_b &\leq ((b-1)_{m-1}(b-1)_{m-2}\ldots(b-1)_1(b-1)_0)_b \\ &= (1_m0_{m-1}\ldots 0_10_0)_b - 1 \\ &= b^m - 1 < b^m \ .\end{aligned}$$

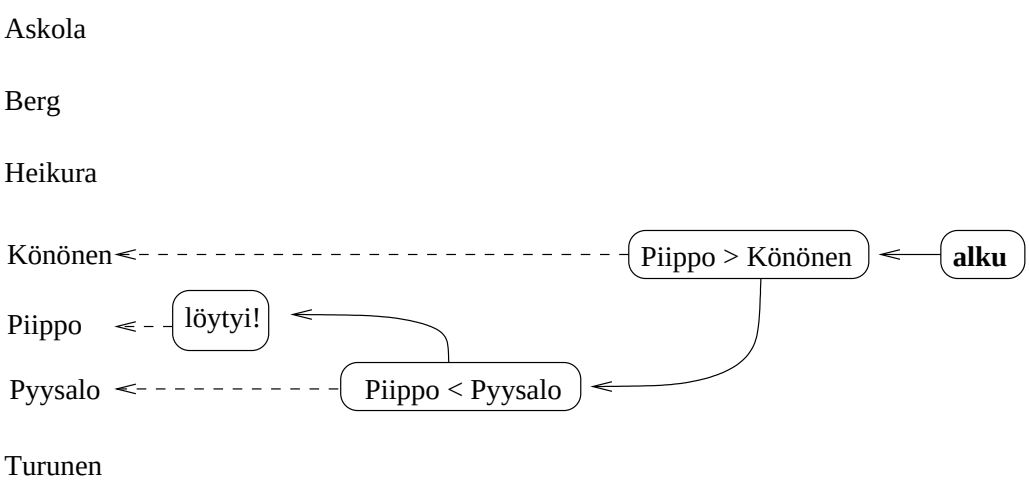
Soveltamalla logaritmia tämän epäyhtälöketjun ensimmäiseen ja viimeiseen jäseneen näemme että $\log_b n < m$. Näiden arvioiden mukaan kokonaisluku m on siis suurempi kuin $\log_b n$ ja enintään $\log_b n+1$. Tämä luku voidaan ilmaista yksinkertaisessa muodossa käyttämällä desimaaliluvun x katkaisevalle alaspäinpyöristykselle merkintää $\lfloor x \rfloor$. (Esimerkiksi $\lfloor 2{,}0 \rfloor = \lfloor 2{,}5 \rfloor = \lfloor 2{,}99 \rfloor = 2$.) Koska desimaaliosan katkaiseminen pienentää lukua alle ykkösellä, on voimassa $\log_b n - 1 < \lfloor \log_b n \rfloor \leq \log_b n$. Lisäämällä tämän epäyhtälön osapuoliin ykkönen nähdään edellisen nojalla, että $m = \lfloor \log_b n \rfloor + 1$. Olemme siis osoittaneet, että kokonaisluvun n esitys b-kantaisessa järjestelmässä on pituudeltaan ykkösen tarkkuudella $\log_b n$. Tarkistetaan vielä, että tulos pätee esimerkiksi budjetin loppusummaan $n = 209\ 172\ 310\ 000$. Nyt $10^{11} < n < 10^{12}$, joten $\lfloor \log_{10} n \rfloor + 1 = 11 + 1 = 12$, mikä täsmää luvun pituuden kanssa.

Positionaalisen lukuesityksen voimasta ja logaritmisen kasvun hitaudesta saa käsityksen tarkastelemalla vaihtoehtoista *unaarista* esitystapaa. Unaarinen esitys tarkoittaa alkeellista ”tukkimiehen kirjanpitoa”, jossa yksinkertaisesti kirjoitetaan peräkkäin esitettävää lukua vastaava määrä ykkösiä.

Kuinka pitkä budjetin loppusumma olisi unaariesityksenä? Arvioidaan, että kirjoitamme noin yhden ykkösen millimetriä kohden. Tällöin budjetin tulojen kokonaismäärän unaariesityksen pituus on noin 209 172,31 km. Tällainen esitys ei mahdu millekään paperiarkille, joten aletaan kirjoittaa sitä vaikka pitkin maantienvartta. Helsingin ja Kilpisjärven välinen etäisyys on Tielaitoksen mukaan 1209 km. Jos aloitamme kyseisen unaariluvun kirjoittamisen tien varteen pääkaupungissa, täytyy Helsinki-Kilpisjärvi-väli kulkea edestakaisin 86 kertaa ja lopuksi matkata vielä kertaalleen Kilpisjärvelle ennenkuin koko luku on kirjoitettu. Valtion budjetin valmistelu unaariluvuin olisi siis ilmeisen hankalaa! Bensaa palaisi, ja tienvartta tallustavien hirvien jäljet sotkisivat laskelmia. Sen sijaan tutussa kymmenjärjestelmässä saman luvun logaritminen pituus on vain 12 numeroa, ja kyseinen summa on siten melko kätevästi hahmotettavissa ja käsiteltävissä.

Ohjelmointikielten toteutukset esittävät kokonaislukuja tyypillisesti yhteen tietokoneen muistisanaan mahtuvina binäärilukuina. Budjetin loppusumma on jo niin suuri luku, että sen binääriesitys ei mahdu tyypillisen modernin tietokoneen 32-bittiseen muistisanaan: edellisen tuloksemme mukaan kyseisen luvun binääriesitys vaatii bittejä $\lfloor \log_2 209172310000 \rfloor + 1 = 37 + 1 = 38$ kappaletta. Budjetin lukuja on siten tietokoneella käsiteltävä esimerkiksi tuhansina markkoina tai käyttäen pidempää, tyypillisesti 64-bittistä kokonaislukujen esitystä.

Tietokoneen muisti koostuu suuresta joukosta yksittäisiä muistitavuja. Jokaisella muistitavulla on osoitteenaan ei-negatiivinen kokonaisluku, jota prosessori käsittelee *osoiterekisterissään*. Suurin n-bittiseen osoiterekisteriin mahtuva binääriluku on $(1_{n-1}1_{n-2}\ldots 1_0)_2$, jonka arvo on $2^n - 1$. Tällöin kone voi käyttää 2^n-tavuisen keskusmuistin muodostamaa *osoiteavaruutta* numeroimalla muistitavut $0, 1, \ldots, 2^n - 1$. Osoiterekisterin pituuden on siis oltava vähintään kaksikantainen logaritmi koneen osoiteavaruuden koosta. Tyypillinen osoiterekisterin pituus on 32 bittiä, mikä riittää toistaiseksi hyvin nykyisten tietokoneiden osoiteavaruuksille aina neljään gigatavuun ($4 \cdot 2^{30} = 2^{32}$) saakka. Keskusmuistien jatkuvasti kasvaessa osoiterekisterien kuitenkin odotetaan jatkossa pitenevän esimerkiksi 40-bittisiksi.



Kuva 3: Binäärihaku listasta nimiä.

Miten etsiä puhelinnumeroita?

Mikä on tehokas menetelmä selvittää ihmisen puhelinnumero, kun tiedämme hänen nimensä? Nykyään moni varmaan selvittää asian soittamalla kännykällä numerotiedusteluun. Perinteisen puhelinluettelon käyttäminen on kuitenkin halvempaa ja mahdollisesti myös nopeampaa.

Ihminen etsii nimeä puhelinluettelosta jotakuinkin seuraavasti: Luettelo avataan niiltä paikkeilta missä nimen arvellaan esiintyvän. Korhonen löytyisi luultavasti luettelon keskivaiheilta, kun taas vaikkapa Ylppöä kannattaisi etsiä loppupuolelta. Jos haettu nimi edeltää aakkosjärjestyksessä avatun sivun sisältöä, etsintä kohdistetaan seuraavaksi luettelon avaamiskohtaa edeltävään osaan. Päinvastaisessa tapauksessa etsintää jatketaan vastaavasti avaamiskohtaa seuraavasta luettelon osasta. Haettu nimi löytyy parhaimmillaan jo ensimmäisiltä avatuilta sivuilta, mutta muussa tapauksessa samaa avauskohdan etu- tai takapuolelta hakemista jatketaan kunnes nimi löytyy tai selviää, että haettua numeroa ei ole luettelossa.

Samaan menetelmään perustuu yleinen *binäärihaun* nimellä tunnettu algoritmi. Haettavan arvon – edellä nimen – etsintä järjestyksessä olevien arvojen jonosta aloitetaan tutkimalla jonon keskimmäistä alkiota.[2] Jos arvo löytyy, etsintä päättyy onnistuneesti. Muuten täsmälleen samaa metodia sovelletaan jonon alku- tai loppupuoliskoon sen mukaan, havaittiinko etsittävä arvo pienemmäksi vai suuremmaksi kuin jonon keskeltä tutkittu alkio. Kuvassa 3 on esimerkki binäärihausta etsittäessä nimeä ”Piippo” aakkosjärjestyksessä olevien nimien listasta.

Binäärihaku on erittäin tehokas tapa etsiä tietoa, mikä nähdään seuraavasti: Tarkastellaan työläintä tilannetta, jossa alkio löytyy (tai sen puuttuminen havaitaan) vasta kun etsittävä jono on toistuvien puolitusten tuloksena kutistunut yhdeksi ainoaksi alkioksi. Jos ensimmäinen vertailu tutkii n-alkioisen jonon keskialkiota, seuraavalla kerralla jonon pituus on puolittunut arvoon $n/2$, sitten arvoon $n/4$, ja niin edelleen, kunnes jäljellä on vain yksi alkio. Montako tällaista vaihetta tarvitaan? Ajattellaanpa prosessia takaperin: montako kertaa ykkösen mittaisen jonon pituutta on kaksinkertaistettava, jotta saadaan vähintään alkuperäisen n pituinen jono? Kysymys on lähes sama kuin ”montako kertaa luku 2 on kerrottava itsellään, jotta saadaan luku n”, joten vastaus on likimain $\log_2 n + 1$.

Olisiko järjestetystä jonosta etsintää mahdollista suorittaa binäärihakua oleellisesti tehokkaammin? Vastaus on kielteinen, ainakin sellaisten algoritmien osalta, joiden toiminta perustuu etsittävän arvon ja jonon alkioiden välisiin vertailuihin.[3] Binäärihaun *optimaalisuus* voidaan perustella seuraavasti.

Tietokoneohjelmat käsittelevät järjestettyjä jonoja *taulukoina*, joitten alkioihin viitataan niiden järjestysnumerolla. Ajatellaan arvojen välisiin vertailuoperaatioihin ($<$, $\leq$, $=$, $\geq$ ja $>$) perustuvaa proseduuria Search, joka saa syötteenään järjestetyn n-alkioisen taulukon sekä siitä etsittävän arvon x. Proseduuri palauttaa taulukon alkion

[2] Jos jonon pituus on parillinen, täsmällisen algoritmin täytyy päättää, kumpaa keskimmäisistä alkioista tutkitaan.

[3] Vaihtoehtoisena strategiana voisi ajatella esimerkiksi yritystä laskea haettavan arvon mahdollinen sijaintipaikka hyödyntäen jonkinlaisia jonon arvojakaumaa kuvaavia tietoja.

järjestysnumeron $k \in \{1, \ldots, n\}$, jos etsitty arvo x löytyy taulukossa paikasta k. Mikäli arvoa ei löydy, Search palauttaa arvon 0.

Montako vertailuoperaatiota proseduuri Search joutuu enimmillään suorittamaan? Jokainen hyödyllinen vertailu voi olla tosi tai epätosi eli tuottaa täsmälleen yhden bitin verran informaatiota haetun arvon sijainnista taulukossa. Erisuuruusvertailut $<$, $\leq$, $\geq$ ja $>$ kertovat täytyykö etsityn arvon löytyä vertailukohdan etu- vai takapuolelta, ja yhtäsuuruusvertailu viimeiseksi vaihtoehdoksi jääneen alkion kanssa ilmoittaa, löytyykö arvo tutkitusta paikasta vai puuttuuko se taulukosta kokonaan.

Proseduurin tulosarvoa k voi nyt ajatella arvojen $0, \ldots, n$ esittämiseen riittävän pituisena binäärilukuna, jonka kutakin bittiä vastaa yksi algoritmin suorittama vertailu. Kuten edellä näimme, tämän luvun pituus on $\lfloor \log_2 n \rfloor + 1$, joten Search-proseduuri joutuu väistämättä joskus suorittamaan näin monta vertailuoperaatiota.

Vaikka binäärihaun tehokkuuden ja binääriluvun pituuden välinen yhteys on kiinnostava, algoritmin työmäärän analysointi näin tarkasti, yksittäisten suoritusvaiheitten tarkkuudella, on usein tarpeetonta. Oleellisempaa on algoritmin suoritustehon karkea riippuvuus käsiteltävien syötteiden koosta. Edellä tarkastellun logaritmien hitaan kasvuvauhdin ansiosta binäärihaun kaltaiset *logaritmisessa ajassa toimivat* algoritmit ovat erittäin tehokkaita. Niiden tietokonetoteutukset suoriutuvat käytännössä ratkottavista tapauksista silmänräpäyksessä eivätkä hidastu havaittavasti, vaikka käsiteltävät syötteet pitenisivät moninkertaisiksi.

Suositeltavaa kirjallisuutta

1. J.L. Bentley: *Programming Pearls*, 2nd ed. ACM Press, 1999.
2. D. Harel: *Algorithmics – The Spirit of Computing*, 2nd ed. Addison-Wesley, 1992.
3. G.M. Schneider, J.L. Gersting: *An Invitation to Computer Science*, 2nd ed. Brooks/Cole Publishing Company, 1999.