

Algoritmien suunnittelu ja analyysi

luennot kevätlukukaudella 2004

Jyrki Kivinen

- 58053-7 Algoritmien suunnittelu ja analyysi, 5 ov
- tietojenkäsittelytieteen laudatur-kurssi
- pakollinen algoritmien erikoistumislinjalla
- varsinaiset esitiedot Tietorakenteet, Laskennan teoria
- käytännössä tarvitaan "riittävä matemaattinen kypsyys"; joidenkin asioiden tunteminen todennäköisyyslaskennasta ja differentiaali- ja integraalilaskennasta on avuksi

Opetus kurssilla

- luennot 21.1.–7.5. ke 14–16, pe 10–12 A414 (15 luentoviikkoa)
- harjoitukset alkavat 29.1., muuten ks. opetusohjelma (14 laskuharjoituskertaa)
- kurssikokeet 9.3. (viikot 1–7) ja 10.5. (viikot 8–14)
- laskuharjoitustilaisuudet ”perinteisiä”

Työmääräarvio (hyvin summittainen)

- ”kontaktiopetus” luennot + harjoitukset + tentit = $15 \cdot 4 + 14 \cdot 2 + 2 \cdot 4 = 96$ tuntia
- harjoitustehtävien tekeminen ja muu **itseopiskelu** 6 tuntia/viikko $\Rightarrow 14 \cdot 6 = 84$ tuntia
- tentteihin kertaaminen $10 + 10 = 20$ tuntia

\Rightarrow kokonaistyömäärä $5 \cdot 40 = 200$ tuntia

Arvostelu

- maksimi 60 pistettä: kokeet $24 + 24$ p., harjoitukset 12 p.
- hyväksymisraja n. 30 p., arvosanan 3/3 raja n. 51 p.
- laskuharjoitustehtäviä $14 \cdot 5 = 70$
- 54 merkittyä tehtävää antaa täydet 12 pistettä
- 0 merkittyä tehtävää antaa 0 pistettä
- tällä välillä interpoloidaan lineaarisesti (eli noin 0,22 pistettä per tehtävä, eli 1 piste per 4,5 tehtävää)

Oppimateriaali

- luentojen kalvokopiot ilmestyvät mappiin (A412) ja kotisivulle
- laskuharjoitukset ja malliratkaisut samoin
- tentit perustuvat luentomateriaaliin
- Lähinnä kurssia vastaava oppikirja
 - T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein: Introduction to Algorithms, toinen painos, MIT Press 2001.
- Toinen hyvä mutta suppeampi kirja
 - R. E. Tarjan: Data Structures and Network Algorithms, SIAM 1983.
- jonkin verran materiaalia otetaan muista kirjoista ja artikkeleista
- kalvot on tarkoitettu luentojen tueksi, ei itseopiskeluun
- uutisryhmä hy.opiskelu.tkt1.asa

Tavoitteet

Kurssin suoritettuaan opiskelija

- osaa itsenäisesti soveltaa yleisimpiä perustekniikoita algoritmien suunnittelemisessa ja analysoimisessa
- ymmärtää hieman vaikeampiakin menetelmiä jos kohtaa niitä kirjallisuudessa
- tuntee perusteellisesti tärkeimmät verkkoalgoritmit
- tuntee joidenkin erityisalueiden (approksimointialgoritmit, satunnaisalgoritmit, rinnakkaisalgoritmit) kysymyksenasettelut ja osaa soveltaa perustekniikoita

Yleisemmällä tasolla kurssi kehittää aiemmilla kursseilla (Tietorakenteet, Laskennan teoria) opittua taitoa analysoida algoritmiongelmia matemaattisesti.

Sisältö

1. **Johdanto:** peruskäsitteet, yleiset kysymyksenasettelut, (matemaattisten) perustietojen kertaus
2. **Algoritmien analyysitekniikoita:** iteratiiviset algoritmit, rekursiiviset algoritmit, aika- ja tilavaativuus, keskimääräisen tapauksen analyysi, tasoitettu analyysi
3. **Algoritmien suunnittelutekniikoita:** osittaminen, taulukointi, ahneet algoritmit, peruutus, paikallinen etsintä
4. **Laskennan mallit ja alarajatodistukset:** Turingin kone, RAM, laskentapiirit; päätöspuut ja järjestämisiongelma
5. **Algoritmeja joukkojen käsittelemiseen:** universaali ja täydellinen hajautus, erillisten joukkojen yhdisteet
6. **Algoritmeja verkko-ongelmiin:** etenkin syvyysuuntainen etsintä, lyhimmät polut, verkkovuot

7. **Approksimointialgoritmit** erit. NP-täydellisille ongelmille
8. **Satunnaisalgoritmit:** esimerkkejä, perustekniikoita
9. **Rinnakkaisalgoritmit:** PRAM-malli, perustekniikoita, rinnakkainen järjestäminen

Erityisiä painopistealueita:

- rekursiivisten algoritmien analysointi rekursioyhtälöillä
- taulukointi algoritmien suunnittelutekniikkana
- verkkoalgoritmit

1. Johdanto

Kerrataan peruskäsitteitä ja (matemaattisia) pohjatietoja.

Tämän luvun jälkeen opiskelija

- tietää millä mittareilla algoritmin tehokkuutta arvioidaan (tällä kurssilla)
- osaa käsitellä sujuvasti funktioiden kertaluokkia ("iso O -notaatio")
- ymmärtää polynomisen ja eksponentiaalisen vaativuusluokan eron

1.1 Johdattelevia esimerkkejä

Algoritmi voidaan formalisoida Turingin koneiden tms. avulla. Usein (ja tällä kurssilla) pseudokoodi on käytännöllisempi.

Esimerkki lisäysjärjestäminen

```
insert-sort( $A[1 \dots n]$ ):  
  for  $j := 2$  to  $n$  do  
     $x := A[j]$   
     $i := j - 1$   
    while  $i > 0$  and  $A[i] > x$  do  
       $A[i + 1] := A[i]$   
       $i := i - 1$   
    end while  
     $A[i + 1] := x$   
  end for
```

Ylläoleva esimerkki on esitetty yksityiskohtaisemmin kuin mitä jatkossa yleensä tehdään. Tämän kurssin tarpeisiin voidaan yleensä sanoa

Järjestä taulukko A kasvavaan järjestykseen.

(mikä tosin yleensä tarkoittaa jotain tehokkaampaa algoritmia).

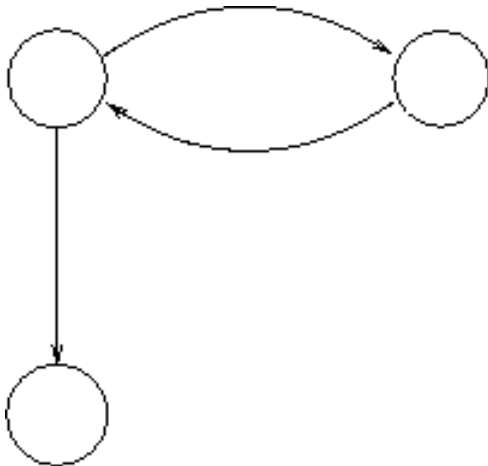
Algoritmilla ratkaistaan **laskennallinen ongelma**:

Annettu: syöte (eli ongelman **tapaus**)

Halutaan: jokin tietty tuloste

Yleensä syöte ja tuloste ajatellaan koodatuiksi jonkin äärellisen aakkoston merkkijonoiksi. Koodaukset ovat yleensä aika selviä eikä niihin tarvitse kiinnittää erityistä huomiota.

Esimerkki Syötteenä suunnattu verkko



$$V = \{ a, b, c \}$$
$$E = \{ (a, b), (a, c), (c, a) \}$$

Vierusmatriisiesitys

A		1	2	3	
1		0	1	1	$a = v_1, b = v_2, c = v_3$
2		0	0	0	$A(i, j) = 1$ joss $(v_i, v_j) \in E$
3		1	0	0	

- Vierusmatriisi aakkoston $\{0, 1, \#\}$ merkkijonona
esim. $\#011\#000\#100\#$.

Vieruslistaesitys

$((2, 3), (), (1))$ Alilista i sisältää ne j joilla
 $(v_i, v_j) \in E$

- Vieruslista aakkoston $\{0, \dots, 9, \#\}$ merkkijonona
esim. $\#\#2\#3\#\#\#\#1\#\#$

(luvun k koodaus aakkostossa $\{0, \dots, 9\}$ vie
 $\lfloor \log_{10} k \rfloor + 1$ merkkiä.)

Jatkon kannalta tärkeä parametri **syötteen koko**

- periaatteessa (ja teoreettisissa tarkasteluissa) koodaavan merkkijonon pituus
- käytännössä jonkin ”luonnollinen” suure (esim. solmujen lukumäärä $|V|$) joka on **polynomisessa suhteessa** koodaavan merkkijonon pituuteen

Funktiot f ja g ovat polynomisessa suhteessa jos $f(s) = O(g(s)^k)$ ja $g(s) = O(f(s)^k)$ jollain $k \in \mathbf{N}$.

Huomaa että jos syötteenä on suuria luonnollisia lukuja, luvun n kooksi pitää ajatella $O(\log n)$.

Esimerkkejä laskennallisista ongelmista

P1 Kertolasku

Annettu kokonaisluvut n ja m

Tulostettava nm

syötteen koko $O(\log |n| + \log |m|)$

P2 Alkuluvut $\{2, 3, 5, 7, 11, 13, \dots\}$

Annettu positiivinen kokonaisluku n

Tulostettava **kyllä** jos n alkuluku, **ei** muuten

syötteen koko $O(\log n)$

P3 Järjestäminen

Annettu kokonaislukujono $S = (s_1, \dots, s_n)$

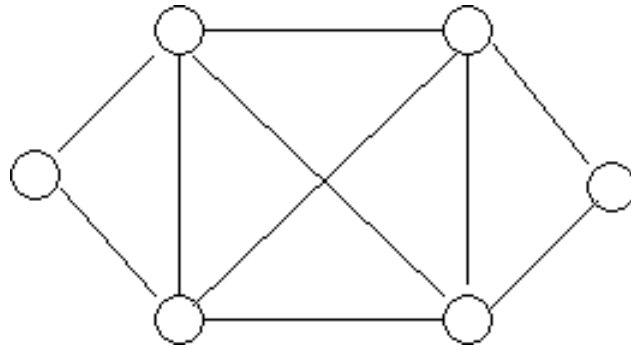
Tulostettava luvut suuruusjärjestyksessä

syötteen koko n (?)

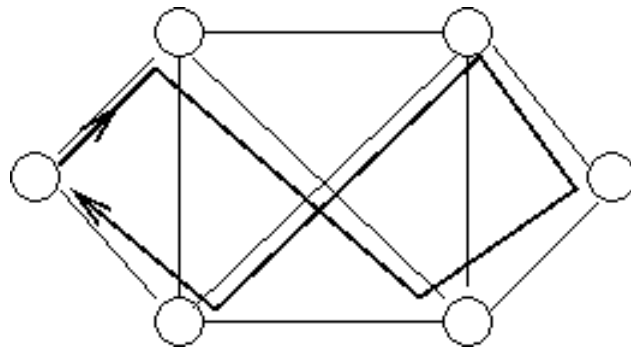
P4 Hamiltonin kehä

Annettu suuntaamaton verkko G

Tulostettava **kyllä** jos verkossa on polku joka käy tasan kerran jokaisessa solmussa ja palaa alkusolmuunsa; **ei** muuten



Verkko

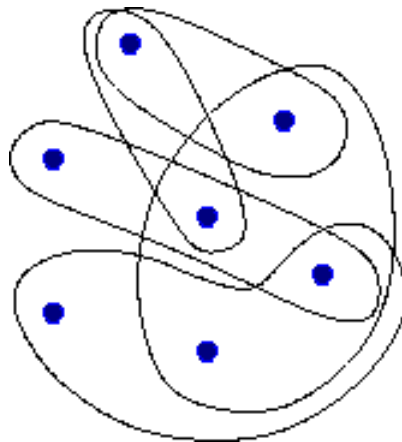


Eräs Hamiltonin kehä

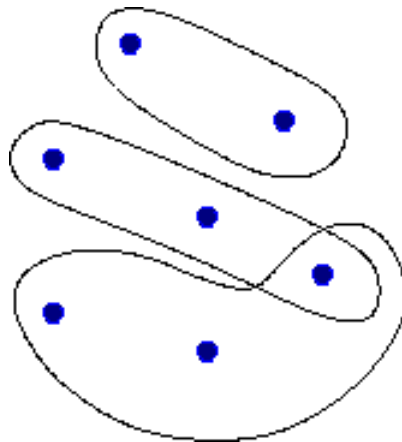
P5 Joukkopeite

Annettu kokoelma jonkin perusjoukon osajoukkoja

Tulostettava pienin määrä osajoukkoja, joka riittää peittämään koko perusjoukon



Perusjoukko ja kokoelma sen osajoukkoja



Joukkopeite jossa $k = 3$ osajoukkoa

P6 Pysähtymisongelma

Annettu ohjelma P , syöte x

Tulostettava **kyllä** jos ohjelma P pysähtyy syötteellä x ; **ei** muuten (s.o. jos ohjelma jää ikuisen silmukkaan)

P7 Totaalisuusongelma

Annettu ohjelma P

Tulostettava **kyllä** jos ohjelma P pysähtyy kaikilla mahdollisilla syötteillä; **ei** muuten

Kurssilla Laskennan teoria on tarkasteltu ongelmien ratkeavuutta:

- P6 ja P7 eivät ole ratkeavia, ts. niille ei ole olemassa ratkaisualgoritmiä joka aina toimisi oikein
- P6 on osittain ratkeava, ts. sille on ratkaisualgoritmi joka voi **ei**-tapauksilla jäädä ikuisen silmukkaan

Tämän kurssin kannalta kiinnostavia ovat esim. ongelmat P1–P5, joilla helposti nähdään olevan jonkinlainen ratkaisualgoritmi mutta ongelmana on löytää tehokas ratkaisualgoritmi.

Huomioita ja kysymyksiä

- P2, P4, P6 ja P7 päätösongelmia (kyllä/ei)
- P5 optimointiongelma (löydettävä pienin/suurin/...)
- P1 osataan ratkaista ajassa $O((\log n)(\log m))$ ja P3 ajassa $O(n \log n)$. Onko parempaan mahdollisuuksia?
- P2:lle löydettiin äskettäin polynomisessa ajassa toimiva algoritmi; aiemmin tunnettiin polynomisessa ajassa toimiva satunnaisalgoritmi. Vastaava etsintäongelma eli tekijöihinjako vaikuttaa nykytietämyksen valossa vaikeammalta.
- P4 on NP-täydellinen ongelma, joten polynomisessa ajassa toimivan algoritmin olemassaolo on merkittävä avoin ongelma
- Myös P5 ratkeaa polynomisessa ajassa jos ja vain jos $P = NP$. On kuitenkin tehokkaita tapoja löytää likimääräisesti pienin joukkopeite.

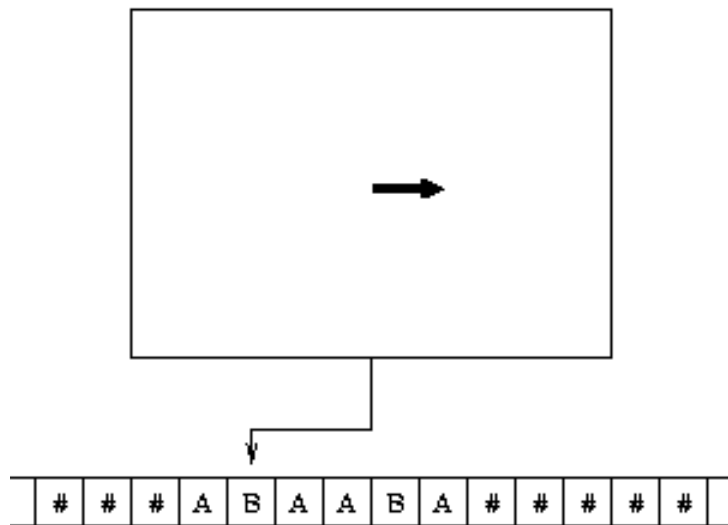
1.2 Algoritmitutkimuksen peruskysymyksiä

- Mikä on algoritmi?
- Miten kehitetään hyviä algoritmeja?
- Miten algoritmin hyvyyttä mitataan?

Viimeiset kaksi kysymystä liittyvät läheisesti toisiinsa: algoritmeja kehitetään tavoitteena optimoida jokin hyvyysmitta (tai useampia mittoja samanaikaisesti).

Algoritmikäsitteen formalisointeja

Turingin kone (Alan Turing, 1936)



Ohjausyksikkö: kone tilassa q_1

Nauhapää osoittaa merkkiä B

Työnauha sis. merkkijonon ABAAB

Koneen **siirtymäfunktio** määrää

- mikä merkki kirjoitetaan nauhapään kohdalle,
- mihin suuntaan nauhapää liikkuu ja
- mikä on seuraava tila

kun on annettu

- nykyinen tila ja
- nauhapään alla oleva merkki.

Motivaatio: yritetään tehdä abstrakti malli siitä, millaista laskentaa matemaatikko (tms.) voi tehdä "mekaanisesti"

Turingin koneita on käsitelty kurssilla Laskennan teoria; niihin palataan lyhyesti myöhemmin.

Rekursiiviset funktiot luonnollisille luvuille (Kleene 1936)

perusfunktiot ovat rekursiivisia:

$$\begin{aligned} z: \mathbf{N} &\rightarrow \mathbf{N}, z(x) = 0 && \text{(vakiofunktio nolla)} \\ s: \mathbf{N} &\rightarrow \mathbf{N}, z(x) = x + 1 && \text{(seuraajafunktio)} \\ p_i: \mathbf{N}^k &\rightarrow \mathbf{N}, z(x_1, \dots, x_k) = x_i && \text{(projektiot)} \end{aligned}$$

yhdistäminen: jos f ja g_1, \dots, g_k ovat rekursiivisia niin h on rekursiivinen kun

$$h(x_1, \dots, x_m) = f(g_1(x_1, \dots, x_m), \dots, g_k(x_1, \dots, x_m))$$

rekursio: jos f ja g ovat rekursiivisia niin h on rekursiivinen kun

$$\begin{aligned} h(0, x_1, \dots, x_m) &= g(x_1, \dots, x_m) \\ h(y + 1, x_1, \dots, x_m) &= f(h(y, x_1, \dots, x_m), y, x_1, \dots, x_m) \end{aligned}$$

minimointi: jos f on rekursiivinen niin h on rekursiivinen kun

$$h(x_1, \dots, x_m) = \min \{ y \mid f(y, x_1, \dots, x_m) = 0 \}$$

Motivaatio: määritellään sellaiset sulkeumaominaisuudet jotka laskettavien funktioiden joukon ainakin pitäisi toteuttaa.

RAM (Random Access Machine)

- abstraktio "tavalliselle" tietokoneelle jossa prosessori ja muistia
- algoritmit esitetään "konekieliohjelmina"

Muita formalismeja

- λ -kalkyyli (Church 1936)
- Postin systeemit (Post 1936)
- rajoittamattomat kieliopit (Chomsky 1955)

Laskettavuuden peruslause: kaikki em. mallit ovat yhtä voimakkaita, ts. funktio on rekursiivinen jos ja vain jos se voidaan laskea Turingin koneella jne.

Churchin-Turingin teesi väittää, että Turing-laskettavat funktiot ovat tasan se luokka funktioita, jotka voidaan laskea mekaanisesti annettuja sääntöjä noudattamalla.

Algoritmin suunnittelemisesta

- luovaa toimintaa, vähän yleisiä sääntöjä
- perustavanlaatuiset suunnittelutekniikat: osittaminen, taulukointi, ahneus, peruutus, karsiva etsintä; satunnaisalgoritmit
- palautukset tunnettuihin ongelmiin
- tehokkaiden perustietorakenteiden käyttö (hakupuut, tasapainoiset puut, keko, ...)

Algoritmin analysoimisesta

- peruslähtökohta oikeellisuus: kaikilla syötteillä oikea tuloste.
- oikeellisuusvaatimuksen lievennyksiä:
 - satunnaisalgoritmit:** sallitaan väärä vastaus pienellä todennäköisyydellä
 - approksimointialgoritmit:** sallitaan hieman suboptimaalinen ratkaisu optimointiongelmaan
- tyypillisin ongelma: algoritmin **pahimman tapauksen aikavaatimuksen** määrittäminen (kertaluokan tarkkuudella; O-notaatio)
- aikavaatimuksen sijasta/lisäksi voidaan analysoida tilavaatimusta, prosessorien määrää, mikropiirin pinta-alaa, ...
- pahimman tapauksen sijaan voidaan tarkastella keskimääräistä tapausta tai tehdä **tasoitettu analyysi**

Ongelman vaativuusanalyysi: onko annettu algoritmi jossain mielessä paras mahdollinen?

- siis halutaan **alaraja** ongelman vaativuudelle
- yleensä hyvin vaikea osoittaa
- usein vedotaan lisäoletuksiin ("jos $P \neq NP$ niin ...") tai rajoitetaan laskennan mallia (esim. vertailuihin perustuva järjestäminen)

Esimerkki: järjestämisongelma

- lisäysjärjestäminen: aika $O(n^2)$
- ei optimaalinen, sillä esim. lomitusjärjestäminen (merge sort): aika $O(n \log n)$
- mikä tahansa **vertailuihin perustuva** järjestämisalgoritmi joutuu tekemään $O(n \log n)$ vertailua, joten lomituslajittelu on jossain mielessä optimaalinen
- lisäysjärjestäminen helppo koodata, vakiotyötila
- pikajärjestäminen (quicksort) menee "keskimäärin" ajassa $O(n \log n)$; onko tällainen keskimääräinen tapaus käytännössä oikea?

1.3 Algoritmin tehokkuusmitat

Olkoon $T(x)$ algoritmin käyttämä aika syötteellä x ja $|x|$ syötteen x koko (merkkijonon pituus, verkon solmujen lukumäärä tms.)

Pahimman tapauksen aikavaativuus määritellään

$$T_{\max}(n) = \max \{ T(x) \mid |x| = n \}$$

Keskimääräinen aikavaativuus: Olkoon P_n todennäköisyysmitta kokoa n oleville syötteille: $P_n(x) \geq 0$ jos $|x| = n$, $P_n(x) = 0$ jos $|x| \neq n$, $\sum_x P_n(x) = 1$. Nyt

$$T_{\text{ave}}(n) = \sum_{|x|=n} P_n(x)T(x)$$

kuvaa toivottavasti "tyypillistä" käyttäytymistä paremmin kuin "pessimistinen" T_{\max} . Ongelmia:

- jakauman P_n valinta: tasainen ei usein vastaa todellisuutta
- laskeminen vaikeaa
- usein hajonta ja "hännät" tärkeitä

Esimerkki kaksi algoritmia A ja B , joiden aikavaativuudet T^A ja T^B

Kymmenen erilaista syötetapausta x_1, \dots, x_{10} ; tasainen jakauma $P_n(x_1) = P_n(x_2) = \dots = P_n(x_{10}) = 1/10$

Oletetaan

$$T^A(x_1) = T^A(x_2) = \dots = T^A(x_5) = 1,0 \text{ s}$$

$$T^A(x_6) = T^A(x_7) = \dots = T^A(x_{10}) = 2,0 \text{ s}$$

joten $T_{\text{ave}}^A(n) = 1,5 \text{ s}$.

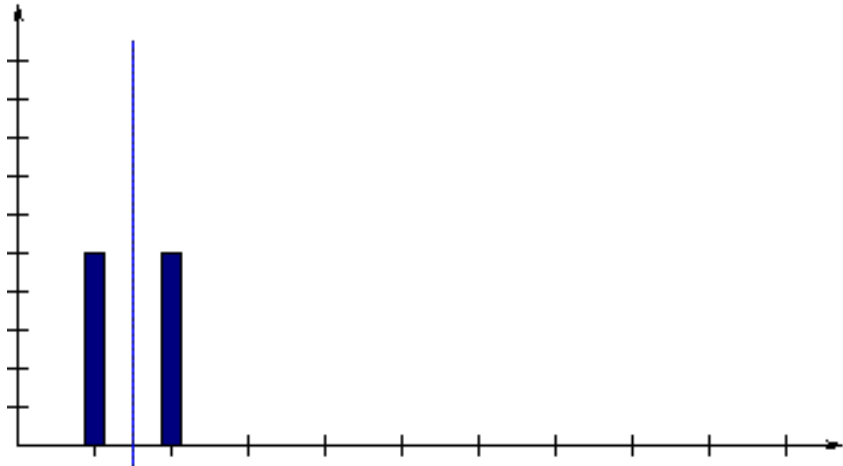
Vastaavasti olkoon

$$T^B(x_1) = T^B(x_2) = \dots = T^B(x_9) = 0,5 \text{ s}$$

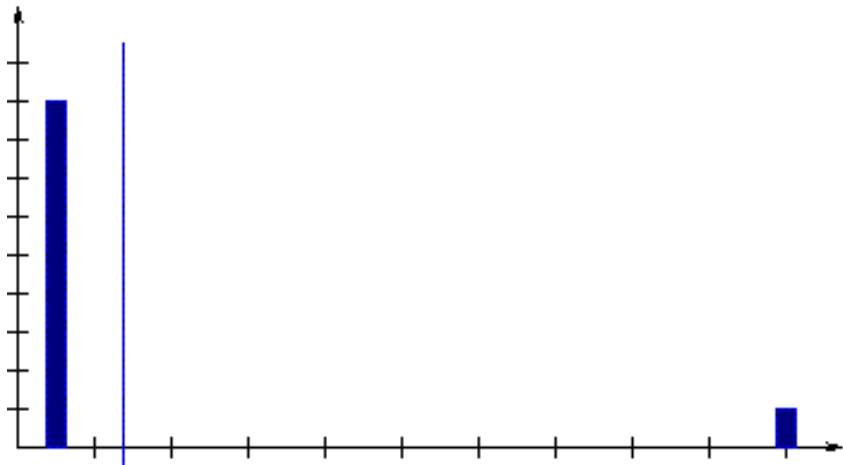
$$T^B(x_{10}) = 10,0 \text{ s}$$

joten $T_{\text{ave}}^B = 1,45 \text{ s}$.

Siis keskimäärin B on nopeampi, mutta hajonta on hyvin suuri.



Algoritmin *A* suoritusajan jakauma



Algoritmin *B* suoritusajan jakauma

Keskimääräisen tapauksen sijaan voidaan analysoida suureita T_p , $0 \leq p \leq 1$:

$$T_p(n) = \min \{ t \mid P_n(T(x) > t) \leq p \}$$

missä kaikilla $t \in \mathbf{R}$

$$P_n(T(x) > t) = \sum_{T(x) > t} P_n(x)$$

on todennäköisyys että suoritus aika on yli t

Esimerkkitapauksessa

$$\begin{aligned} T_p^A(n) &= 2,0 \text{ s} & \text{kun } p < 0,5 \\ T_p^B(n) &= 0,5 \text{ s} & \text{kun } 0,1 \leq p \leq 1 \\ T_p^B(n) &= 10,0 \text{ s} & \text{kun } p < 0,1 \end{aligned}$$

Tasoitettu aikavaativuus (amortized): pahimman tapauksen analyysia, mutta kokonaisessa jonossa operaatioita kustannukset tasataan koko jonon kesken. Olkoon $T(x_1, \dots, x_n)$ operaatiojonon (x_1, \dots, x_n) aikavaativuus. Määritellään

$$T_{\text{amort}}(n) = \frac{1}{n} \max \{ T(x_1, \dots, x_n) \}$$

Esimerkki: x_i on lisäys, haku tai poisto tietokannasta. Kukin operaatio muuten ajassa $O(\log n)$, mutta jonon keskivaiheilla joudutaan uudelleenorganisoimaan jokin hakemisto mihin kuluu aika $O(n)$. Nyt

$$T_{\text{amort}}(n) = O(\log n)$$

vaikka

$$\max T(x_i) = O(n).$$

(Onko T_{amort} tällaisessa tilanteessa oikea mitta?)

Mitä oikeastaan mitataan?

- sovelluksen kannalta kiinnostava suure on tietysti suoritukseen kuluva fysikaalinen aika
- todellinen aika kuitenkin riippuu laitteistosta jne.

⇒ teoreettisissa tarkasteluissa arvioidaan **alkeisoperaatioiden** määrää (Turingin koneen siirtymät; RAM-koneen konekäskyt; järjestämisalgoritmin vertailut; ...)

- oletetaan, että kukin alkeisoperaatio voidaan toteuttaa vakioajassa

⇒ fysikaalinen aika = $O(\text{alkeisoperaatioiden määrä})$