811312A Tietorakenteet ja algoritmit, 2018 - 2019, Harjoitus 3, Ratkaisu

Harjoituksessa käsitellään algoritmien aikakompleksisuutta.

Tehtävä 3.1 Kuvitteelliset algoritmit A ja B lajittelevat syötteenään saamat lukutaulukot. Algoritmi A käyttää $32 \cdot nlg(n)$ operaatiota ja algoritmi B käyttää $3 \cdot n^2$ operaatiota, kun taulukon koko on n. Arvioi, minkä kokoisten taulukoiden lajitteluun kannattaa käyttää algoritmia A ja minkä kokoisten taulukoiden lajitteluun kannattaa käyttää algoritmia B.

Ratkaisu. Algoritmin A suoritusaika kasvaa hitaammin kuin algoritmin B, koska lg(n) kasvaa hitaammin kuin n. Näin ollen algoritmi A pääsee lopulta voitolle, kun edetään riittävän suuriin n:n arvoihin. Etsitään sellainen taulukon koon arvo n, jossa algoritmi A ohittaa B:n suoritusnopeudessa. Kurssin merkinnöissä lg tarkoittaa 2-kantaista logaritmia, joten $lg(2^k)= k$. Merkitään operaatioiden määriä taulukoituna, kun n saa arvoikseen luvun 2 potensseja.

k	$n=2^k$	A: $32*n*lg(n)$	B: $3*n^2 = 3*n*n$
4	16	32*16*4 = 2048	3*16*16 = 768
5	32	32*32*5 = 5120	3*32*32 = 3072
6	**64**	**32*64*6 = 12288**	**3*64*64 = 12288**
7	128	32*128*7= 28672	3*128*128 = 49152 jne.

Kun $n = 2^6 = 64$, algoritmit suorittavat yhtä monta operaatiota.
Tätä pienemmille taulukoille kannattaa käyttää algoritmia B ja suuremmille algoritmia A.

Riittävän suurilla aineiston koon arvoilla n kompleksisuusluokan $\Theta(nlg(n))$ algoritmi tulee nopeammaksi kuin luokan $\Theta(n^2)$ algoritmi, olivatpa vakiokertoimet mitä tahansa.

Tehtävä 3.2 Lataa koneellesi (linkki alempana) ohjelmatiedosto **`h3_t2.cpp`**. Käännä ohjelma ja suorita se (voi tehdä luokassa esimerkiksi CodeBlocksilla tai kääntämällä komentoriviltä). Komentoriviltä käännetään antamalla syöte

g++ h3_t2.cpp –o t2.exe (Windowsissa) tai **g++ h3_t2.cpp –o t2** (Linuxissa).

Ohjelmassa voidaan valita suoritukseen jokin kolmesta algoritmista (A, B ja C) halutulla syötteen koolla. Tee ohjelman avulla seuraavaa:

a) Taulukoi algoritmin A suoritukseen kuluneita aikoja, kun syötteen koko vaihtelee välillä 500 – 20 000. Pyri arvaamaan mikä on suoritusaika, kun syötteen koko on 50 000. Tarkista arvauksesi suorittamalla ohjelma.
b) Taulukoi algoritmin B suoritukseen kuluneita aikoja, kun syötteen koko vaihtelee välillä 25 – 200. Pyri arvaamaan mikä on suoritusaika, kun syötteen koko on 400. Tarkista arvauksesi suorittamalla ohjelma.

c) Taulukoi algoritmin C suoritukseen kuluneita aikoja, kun syötteen koko vaihtelee välillä 100 000 – 5 000 000. Pyri arvaamaan mikä on suoritusaika, kun syötteen koko on 80 000 000. Tarkista arvauksesi suorittamalla ohjelma.

Arvioi vielä, mikä olisi algoritmien A ja B suoritusaika, kun syötteen koko on 1000 000.

Ratkaisu.
HUOM! Tässä tehtävässä ei kannata käyttää kääntäjän optimointiasetusta, koska algoritmit toistavat samaa operaatiota tietyn kaavan mukaan. Näin ollen kääntäjä saattaa optimoida koodia siten, että tulokset muuttuvat.

HUOM2! Joissakin järjestelmissä C-ohjelmien suoritusajat eivät näy millisekunteina. Näin ollen ohjelman koodissa olisi syytä jakaa aika vielä vakiolla CLOCKS_PER_SEC, jolloin aika saadaan sekunneissa.

Alla taulukoituina algoritmien suoritusaikoja kurssin vastuuhenkilön tietokoneella.

Algoritmi A:

Syötteen koko	500	1000	2000	5000	10000	20000
Aika (ms)	143	289	587	1508	2984	5954

Algoritmi B:

Syötteen koko	25	50	100	200
Aika (ms)	110	422	1641	6594

Algoritmi C:

Syötteen koko	100000	400000	1600000	5000000
Aika (ms)	218	454	922	1828

Algoritmi A:n suoritusaika näyttää kasvavan lineaarisesti: kun syötteen koko kaksinkertaistuu, myös suoritusaika kaksinkertaistuu ja kun syöte viisinkertaistuu, suoritusaika viisinkertaistuu. (Tällöin algoritmin suoritusaika on luokkaa $\Theta(n)$, kun syötteen koko on n.) Näin ollen voidaan arvioida algoritmin suoritusajaksi syötteen koolla 50000 viisi kertaa suoritusaika syötteen koolla 10000, ts. 5*2984 = n. 14900 ms. Kun algoritmi suoritettiin, saatiin ajaksi 14938 ms, mikä vastaa hyvin arviota. Jos syötteen koko kasvatettaisiin 1000000:aan, niin suoritusaika satakertaistuisi verrattuna aikaan syötteen koolla 10000, joten voitaisiin olettaa suorituksen kestävän noin 100*2984 ms = n. 298 s = n. 5 min. Tämän voisi kärsivällinen testaaja vielä tarkistaa.

Algoritmi B:n suoritusaika näyttää sen sijaan nelinkertaistuvan, kun syötteen koko kaksinkertaistuu. Näin ollen voidaan arvioida algoritmin suoritusajaksi syötteen koolla 400 neljä kertaa suoritusaika syötteen koolla 200, ts. 4*6594 = n. 26400 ms. Kun algoritmi suoritettiin, saatiin ajaksi 26563 ms, mikä vastaa hyvin arviota. Tällainen algoritmi on neliöllinen, ts. algoritmin suoritusaika on luokkaa $\Theta(n^2)$, kun syötteen koko

on n. Jos syötteen koko kasvatettaisiin 1000000:aan, niin suoritusajan suhde suoritusaikaan syötteen koolla 100 olisi $(10000)^2$, koska 1000000/100 = 10000. Siten voitaisiin olettaa suorituksen kestävän noin $(10000)^2$*1641 ms = 164 100 000 s = n. 5.2 vuotta. Tätä ei liene enää tarkoituksenmukaista verifioida.

Algoritmi C:n suoritusaika näyttää puolestaan kaksinkertaistuvan, kun syötteen koko nelinkertaistuu. Näin ollen voidaan arvioida algoritmin suoritusajaksi syötteen koolla 80 000 000 neljä kertaa suoritusaika syötteen koolla 5 000 000, ts. 4*1828 = n. 7300 ms, koska 80000000/5000000 = 16. Kun algoritmi suoritettiin, saatiin ajaksi 7391 ms, mikä vastaa hyvin arviota. Tässä tapauksessa algoritmin suoritus kasvaa kuten $\sqrt{n}$, kun syötteen koko on n. (Algoritmin suoritusaika on siis luokkaa $\Theta(\sqrt{n})$, kun syötteen koko on n.)

Tehtävä 3.3 Seuraava lisäyslajittelu on hyvä pienten taulukoiden lajittelualgoritmi. Algoritmin oikeellisuus on perusteltu luennoilla. Määritä (ja perustele) algoritmin kompleksisuusluokka. Riittää tarkastella ainoastaan huonointa tapausta.

```
Syöte: Taulukko A[0,..,n-1], n >= 1
Tuloste: Taulukon luvut järjestyksessä A[0] <= … <= A[n-1]

LISAYS(A)
 1.   for j = 1 to n-1
 2.      k = A[j]
 3.      i = j-1
 4.      while i>=0 && A[i]>k
 5.         A[i+1] = A[i]
 6.         i = i-1
 7.      A[i+1] = k
 8.   return
```

Ratkaisu. Rivejä 1-3 suoritetaan *n-1* kertaa, samoin riviä 7. Jos taulukko on käänteisessä järjestyksessä (huonoin tapaus), while-silmukan rivejä 5 ja 6 joudutaan suorittamaan jokaisella kierroksella *j* kertaa eli yhteensä $1+2+ \ldots + (n-1) = n(n-1)/2$ kertaa.
Riviä 4 suoritetaan joka kierroksella yhden kerran enemmän kuin sen sisällä olevia rivejä, joten huonoimmassa tapauksessa riviä 4 suoritetaan
$2+3+ \ldots + (n-1) + n = n(n-1)/2 + n-1$ kertaa.
Siten suoritettavia rivejä tulee pahimmassa tapauksessa
$4(n-1) + 2n(n-1)/2 + n(n-1)/2 + n-1 = 4n -4 +n^2 – n + 1 +n^2/2 - n/2 + n -1$
$= 3n^2/2 + 7n/2 -5$.
Huomataan, että lisäyslajittelun kompleksisuusluokka pahimmassa tapauksessa on $\Theta(n^2)$, sillä suurilla n:n arvoilla vaikuttavin termi on $3n^2/2$. Voidaan myös osoittaa, että keskimääräinen tapaus on samaa luokkaa.

Ohjelmointitehtävä

Tehtävä 3.4. Ohjelmissa tarvitaan usein erilaisten objektien joukkojen satunnaista järjestämistä. Tällaista algoritmia sanotaan **satunnaisen permutaation** tuottamiseksi. Tarkastele seuraavaa algoritmia, jolla voidaan tuottaa lukujen 1, …, n satunnaisen järjestyksen sisältävä kokonaislukutaulukko A[0,..,n-1].

```
Syöte: Luku n >= 1. Taulukko A[0,..,n-1].
Tulostus: Taulukossa A luvut 1..n satunnaisessa järjestyksessä.
RANDPERM(n,A)
 1. for i = 0 to n-1 // Taulukon alustus järjestykseen 1,2,..,n
 2.    A[i] = i+1
 3. for i = 0 to n-1
 4.    x = satunnaisluku väliltä 0..i
 5.    vaihda A[i] ja A[x]
 6. return
```

Ohjelmoi algoritmi (joko C- tai Python-kielellä) ja mittaa suoritukseen kuluvaa aikaa kasvattaen taulukon kokoa. Minkä kokoisilla taulukoilla on käytännöllistä suorittaa algoritmia? Mikä on algoritmin aikakompleksisuusluokka, kun syötteen koon mitta on taulukon koko?

Sovelletaan em. mainittua algoritmia. Korttipakassa on 52 korttia, jotka jakaantuvat neljään maahan (hertta, ruutu, risti, pata), joissa kaikissa on 13 korttia numeroituina 1, ...,13. Pokerissa jaetaan kaikille pelaajille aluksi viisi korttia. Halutaan tietää, mikä on pelaajan todennäköisyys saada jaossa väri, eli kaikki kortit samaa maata. Tätä voidaan arvioida simuloimalla: Mallinna pakan kortteja luvuilla 1, ...,52, tuota suuri määrä lukujen 1, …,52 permutaatioita ja laske, kuinka monessa näistä ensimmäiset viisi lukua vastaavat saman maan kortteja. Tämän lukumäärän suhde tuotettujen permutaatioiden lukumäärään arvioi haluttua todennäköisyyttä. (Laskettu todennäköisyys on n. 0.00198)

Ratkaisu. Esimerkkiratkaisut on linkitetty alle. Ohjelman valikosta voidaan valita, mitataanko satunnaispermutaation tuottamisaikaa vai simuloidaanko värin saamisen todennäköisyyttä. Permutaatioalgoritmi on selvästi aikakompleksisuudeltaan lineaarinen (luokkaa $\Theta(n)$, kun taulukon koko on n), joten suoritusaikojen pitäisi kasvaa suorassa suhteessa syötteen kokoon. Testitietokoneella saatiin satunnaispermutaation tuottamisen Python-toteutukselle seuraavat ajat (sekunneissa) vaihtelevan kokoisilla taulukoilla:

Taulukon koko	100 000	400 000	800 000	1 000 000
Aika (s)	0.24	0.98	1.99	2.45

Taulukosta nähdään, että Python-ohjelmaa voidaan kohtuudella käyttää tuottamaan vielä miljoonan alkion permutaatioita, ellei sen tarvitse tapahtua reaaliaikaisesti.

C-ohjelmat ovat käännettyinä yleisesti nopeampia kuin Python-ohjelmat. C-kielinen toteutus suoriutui seuraavasti:

Taulukon koko	1 000 000	5 000 000	10 000 000	20 000 000
Aika (s)	0.069	0.53	1.16	2.57

Huomataan, että C-ohjelma tuottaa noin 20 kertaa pitemmän permutaation kuin Python vastaavassa ajassa.

Kun arvioidaan simuloimalla värin todennäköisyyttä, havaitaan että ohjelman tuottama todennäköisyys on hyvin lähellä teoreettisesti laskettua todennäköisyyttä, kun jakojen lukumäärä lähestyy 100 000:a. C-ohjelmalla voidaan samassa ajassa tuottaa lähes 50-kertainen määrä jakoja kuin Python-ohjelmalla.