811312A Tietorakenteet ja algoritmit, 2018 - 2019, Harjoitus 4, Ratkaisu

Harjoituksen aiheita ovat algoritmien aikakompleksisuus ja lajittelualgoritmit. Tällä kertaa keskitytään enemmän ohjelmointiin kuin aiemmissa harjoituksissa.

Tehtävä 4.1 Luennoissa esitettyä lausetta M (Master Theorem) voi käyttää joidenkin rekursioyhtälöiden ratkaisujen kasvunopeuden arvioimiseen:

Lause M. (*Master Theorem*) Olkoot $a \geq 1 b > 1$ kokonaislukuvakioita ja f funktio. Rekursion $T(n) = a \cdot T(\lfloor n/b \rfloor) + f(n)$ samoin kuin rekursion $T(n) = a \cdot T(\lceil n/b \rceil) + f(n)$ ratkaisulle pätee seuraavaa:

1) Jos $f(n) \in O(n^{\log_b a - \varepsilon})$ jollakin vakiolla $\varepsilon > 0$, niin

$$T(n) \in \Theta(n^{\log_b a})$$

2) Jos $f(n) \in \Theta(n^{\log_b a})$, niin

$$T(n) \in \Theta(n^{\log_b a} \cdot \lg n)$$

3) Jos $f(n) \in \Omega(n^{\log_b a + \varepsilon})$ jollakin vakiolla $\varepsilon > 0$ ja jos $a \cdot f(n/b) \leq c \cdot f(n)$ jollakin vakiolla $c < 1$ ja riittävän suurilla luvun n arvoilla, niin $T(n) \in \Theta(f(n))$

Seuraava rekursiivinen algoritmi toteuttaa **puolitushaun** järjestettyyn taulukkoon. Testaa algoritmia hakemalla taulukosta A = [11,23,31,47,52,68,71,89,94,105,112,126,133,148] alkio 105. Määritä lauseen M avulla algoritmin tiukka kompleksisuusluokka (Θ-notaatio), kun syötteen koon mittana on taulukon pituus.

```
Syöte: Taulukko A[1,..,n], n >= 1, taulukon alkiot ovat kasvavassa
järjestyksessä A[1] <= A[2] <= … <= A[n]. Luvut 1<=p<=q<=n. Luku x jota
haetaan taulukosta väliltä A[p,..,q].
Tulostus: Alkion x indeksi taulukossa tai arvo -1, jos x ei esiinny
taulukossa välillä A[p,..,q].
HAKU(A,p,q,x)
 1. if p==q
 2.    if A[p]==x
 3.      return p
 4.    else
 5.      return -1
 6. else
 7.    r = ⌊(p + q)/2⌋
 8.    if x<=A[r]
 9.      return HAKU(A,p,r,x)
10.    else
11.      return HAKU(A,r+1,q,x)
```

Ratkaisu. Kun haetaan arvoa 105 taulukosta A, saadaan seuraava kutsujono:
HAKU(A,1,14): r = 7 ja A[r] = 71; 105 > A[r]
HAKU(A,8,14): r = 11 ja A[r] = 112; 105 <= A[r]
HAKU(A,8,11): r = 9 ja A[r] = 94 <= 105; 105 > A[r]
HAKU(A,10,11) : r = 10 ja A[10] = 105; 105 <= A[r]
HAKU(A,10,10): p==q ja A[10]==105: palautetaan 10
Algoritmi palauttaa siis alkion 105 indeksin taulukossa A.

Olkoon algoritmin suoritusaika $T(n)$, kun haetaan paikkaa taulukon osasta, jonka koko on *n*. Lukuun ottamatta rekursiivista kutsua, algoritmissa suoritetaan vakiomäärä vakioaikaisia operaatioita (korkeintaan kaksi vertailua ja yksi laskutoimitus sekä sijoitus). Näin ollen rekursiota lukuun ottamatta suoritusaika on vakio *c*. Jos *n*>1, taulukko puolitetaan ja kutsutaan algoritmia puolitetulle taulukolle. Tämä vie ajan $T(n/2)$. Näin saadaan rekursioyhtälö

$$T(n) = T(n/2) + c.$$

Sovelletaan lausetta M: Nyt $a = 1, b = 2$, joten $\log_b a = \log_2 1 = \log_2 2^0 = 0$. Edelleen $f(n) = c = c \cdot n^0 \in \Theta(n^0)$. Siten lauseen M tapauksen 2 mukaan

$$T(n) \in \Theta(\, n^{\log_b a} \lg n) = \Theta(\, n^0 \lg n) = \Theta(\lg n).$$

Algoritmi on siis kompleksisuusluokaltaan logaritminen. Tämä olisi ollut varsin helppo päätellä myös käyttämättä lausetta M, koska taulukon osa puolittuu jokaisella kutsulla.

Tehtävä 4.2 Kirjoita ohjelma (joko C- tai Python-kielellä), jolla voidaan testata lajittelualgoritmien suoritusaikoja. Kirjoita funktiot, jotka suorittavat kekolajittelun ja pikalajittelun (Quicksort) ja testaa näiden suoritusaikoja erikokoisilla kokonaislukutaulukoilla, jotka alustetaan satunnaisluvuilla. Algoritmit ovat liitteenä tehtävien jälkeen. Huomaa, että Heapsort-algoritmi on tässä sovitettu taulukolle, jonka indeksit alkavat nollasta, toisin kuin luennoissa.

Ratkaisu. Esimerkkiohjelmat C- ja Python-kielillä on linkitetty oheen. Sekä kekolajittelun että Quicksortin (keskimääräinen) aikakompleksisuus on luokkaa Θ(nlg(n)). Tosin Quicksortin huonoimman tapauksen aikakompleksisuus on luokkaa $\Theta(n^2)$, mutta Quicksort toimii kuitenkin keskimäärin 2-5 kertaa nopeammin kuin kekolajittelu. Testikoneella saatiin seuraavia suoritusaikoja kekolajittelulle:

Kekolajittelu: C-ohjelma

Taulukon koko	200 000	500 000	1000 000	2000 000
Aika (s)	0.100	0.285	0.60	1.38

Kekolajittelu: Python-ohjelma

Taulukon koko	20 000	50 000	100 000	200 000
Aika (s)	0.260	0.712	1.60	3.40

Quicksort-algoritmi osoittautui vajaa kolme kertaa nopeammaksi kuin kekolajittelu. C-kielisellä ohjelmalla algoritmi suoriutui kohtuudella vielä taulukoista, joiden koko oli 50 000 000.

Quicksort: C-ohjelma

Taulukon koko	200 000	500 000	1000 000	2000 000
Aika (s)	0.049	0.116	0.239	0.543

Quicksortin suoritusaikaan vaikuttaa melko voimakkaasti se, kuinka suurelta lukuväliltä satunnaislukuja arvotaan. Mikäli luvut arvotaan pieneltä väliltä, taulukkoon arvotaan huomattava määrä samoja lukuja. Tällöin Quicksort hidastuu, koska taulukkoon muodostuu pitkiä järjestyksessä olevia osia, joiden lajittelu sujuu hitaasti. Tässä on C-ohjelmissa arvottu 32-bittisiä satunnaislukuja ja Python-ohjelmissa satunnaislukuja väliltä [1,1000000]. Lopuksi vielä Python-ohjelmalla saatuja Quicksort-lajitteluaikoja.

Quicksort: Python-ohjelma

Taulukon koko	20 000	50 000	100 000	200 000
Aika (s)	0.101	0.304	0.624	1.48

Käytetyt ohjelmakoodit on linkitetty ratkaisutiedoston alle.

Tehtävä 4.3 Lukujonon **tyyppiarvoksi** eli **moodiksi** sanotaan siinä useimmin esiintyvää lukua. Esimerkiksi jonon 3,1,2,5,3,3,4,1,4,4,3,5 tyyppiarvo on 3. **Suunnittele ja implementoi** (joko C- tai Python-kielellä) algoritmi, joka etsii lukutaulukon A moodin. Algoritmin aikakompleksisuusluokka saa olla $O(n \cdot \lg(n))$, kun syötetaulukon koko on n.

Ratkaisu. Käytetään hyväksi lajittelua: Suuruusjärjestykseen saatetusta taulukosta on helppo hakea moodi käymällä taulukko kerran läpi: Lasketaan kuinka monta kertaa sama arvo esiintyy peräkkäin ja pidetään yllä tietoa siitä, mikä arvo on esiintynyt suurimman määrän kertoja. Tällöin algoritmiksi saadaan seuraava:

```
Syöte: Taulukko A[1,..,n], n >= 1.
Tulostus: Taulukon alkioiden tyyppiarvo

MODE(A)
1.   HEAPSORT(A) //Lajittele taulukko käyttäen kekolajittelua
2.   mode = A[1]
3.   freq = 1
4.   temp = 1
5.   i = 2
6.   while i <= n do
7.      if A[i] != A[i-1]
8.         temp = 1
9.      else
10.        temp = temp + 1
11.        if temp > freq
12.             freq = temp
13.             mode = A[i]
14.     i = i+1
15.  return mode
```

Kun tullaan algoritmin riville 2, taulukko on järjestyksessä. Aluksi muuttuja **mode** on taulukon ensimmäinen arvo. Silmukassa tarkastellaan taulukon jokainen alkio indeksistä 2 lähtien. Muuttuja **temp** laskee kulloinkin tarkasteltavan taulukon arvon esiintymiskertoja: kun arvo vaihtuu, aloitetaan uudelleen arvosta 1. Muuttujan **freq** arvo on toistaiseksi useimmin esiintyneen arvon esiintymislukumäärä: jos muuttujan **temp** arvo ylittää muuttujan **freq** arvon, muuttujaa **freq** päivitetään ja myös muuttuja **mode** päivitetään taulukossa esiintyväksi arvoksi. Näin ollen muuttujan **mode** arvo on taulukon tarkastellun osan moodi. Kun algoritmi päättyy, muuttujan arvo on koko taulukon moodi. Algoritmi etsii siis taulukon moodin.

Kekolajittelun aikakompleksisuus on luokkaa $O(n \cdot \lg(n))$. Tämän lisäksi algoritmissa tehdään vakioaikainen operaatio jokaiselle taulukon alkiolle. Algoritmin loppuosan aikakompleksisuus on siis luokkaa $O(n)$ ja algoritmin kokonaiskompleksisuus luokkaa $O(n \cdot \lg(n))$.

Yllä mainittu algoritmi on implementoitu C- ja Python-kielillä linkitettyihin kooditiedostoihin. Lajittelumetodiksi voi valita kekolajittelun tai kirjaston Quicksort-lajittelun.