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

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)
 if p==q
    if A[p]==x
      return p
    else
      return -1
 else
    r = ⌊(p + q)/2⌋
    if x<=A[r]
      return HAKU(A,p,r,x)
    else
      return HAKU(A,r+1,q,x)
```

Ohjelmointitehtävät seuraavalla sivulla ->

Ohjelmointitehtävät

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.

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.

Neuvoja tehtävään 4.2:
C-ohjelma. Satunnaisluvun generoiminen ja jonkin operaation keston laskeminen sekunneissa tapahtuu seuraavasti:

```
#include <stdlib.h>
#include <time.h>

clock_t start,end;
double totaltime;
int rand_x;

// Satunnaisluku
rand_x = rand();

// Isoilla taulukoilla kannattaa varmistaa että satunnaisluku
// on varmasti 32-bittinen ja kutsua funktiota
unsigned int bigRandom(){
	unsigned int random =
		(((unsigned int) rand() <<  0) & 0x0000FFFF) |
		(((unsigned int) rand() << 16) & 0xFFFF0000);

	return random;
}

// Operaation kesto
start = clock();
// OPERAATIO
end = clock();
totaltime = (double)(end-start)/CLOCKS_PER_SEC;
```

Pythonissa vastaavat operaatiot voidaan tehdä seuraavasti:

```
import time
import random

#Satunnaisluku x niin, että väliltä a <= x <= b
x = random.randint(a,b)

#Operaation kesto
start = time.perf_counter()
#OPERAATIO
end = time.perf_counter() - start
```

Liite. Tehtävien lajittelualgoritmit

Heapsort (kekolajittelu)

```
Syöte: Taulukko A[0,…,n-1] (n=A.length)
HEAPSORT(A)
Tuloste: Taulukko A järjestyksessä
BUILD_MAX_HEAP(A)
for i = A.length-1 downto 1
      vaihda A[0] ↔ A[i]
      A.heap_size = A.heap_size – 1
      MAX_HEAPIFY(A, 0)
```

```
BUILD_MAX_HEAP(A)
Tuloste: Taulukosta A maksimikeko
A.heap_size = A.length
for i = ⌊A.length/2⌋ downto 0
   MAX_HEAPIFY(A, i)
```

```
MAX_HEAPIFY(A, i)
   lft = LEFT(i)  ( = 2*i+1)
   rgt = RIGHT(i)  ( = 2*i+2)
   if lft < A.heap_size and A[lft] > A[i]
      largest = lft
   else
      largest = i
   if rgt < A.heap_size and A[rgt] > A[largest]
      largest = rgt
   if largest  !=  i
      vaihda  A[i] ↔ A[largest]
      MAX_HEAPIFY(A, largest)
```

Quicksort

```
Syöte: Taulukon osa A[p, ..,r]
Tuloste: A[p, .. ,r] lajiteltuna
QUICKSORT(A, p, r)
if p < r
   q = PARTITION(A, p, r)
   QUICKSORT(A, p, q-1)
   QUICKSORT(A, q+1, r)
```

```
PARTITION(A,p,r)
x = A[r]
i = p - 1
for j = p to r – 1
   if  A[j] <= x
       i = i + 1
       vaihda A[i] ↔ A[j]
vaihda A[i + 1] ↔ A[r]
return i + 1
```