1e deeltentamen Datastructuren 2012/2013

Je hebt twee uur voor dit deeltentamen.

Schrijf op ieder vel je naam. Schrijf op het eerste vel je collegekaartnummer, en het aantal ingeleverde bladen.

Bij verschillende opgaven wordt om een algoritme gevraagd. Probeer hierbij duidelijke antwoorden te geven. Pseudocode met een toelichting in tekst is daarbij een goede optie. Als om een tijdgrens wordt gevraagd, geef dan een 'zo goed mogelijke grens', waarbij je steeds naar keuze van de O-notatie of de Θ-notatie gebruik mag maken.

Schrijf leesbaar en netjes. Zet je mobiel uit voor aanvang van het tentamen. Je mag geen rekenmachine gebruiken, maar wel vier vellen a4 met zelfgemaakte aantekeningen. Als je deze aantekeningen met een computer hebt gemaakt, dan moet je ze bij je werk inleveren. Handgeschreven aantekeningen mag je mee naar huis nemen na afloop van het tentamen.

Veel succes!

1. O, Ω, en Θ (1.75 punten.)

Zeg van elk van de volgende beweringen of ze waar zijn of onwaar. Je hoeft je antwoord niet toe te lichten. Normering: -0.5 voor elk fout antwoord, -0.25 voor elke niet beantwoorde vraag. (Je kan geen negatieve score halen)

1. $n^3 + 2n^2 + 3n + 4 = \Omega(n^2)$: WAAR of ONWAAR
2. $n^3 + 2n^2 + 3n + 4 = O(n^2)$: WAAR of ONWAAR
3. $n^3 + 2n^2 + 3n + 4 = \Theta(n^2)$: WAAR of ONWAAR
4. $2n \log(n\sqrt{n}) + 2n = \Theta(n \log n)$: WAAR of ONWAAR
5. $3^n = O(n^3)$: WAAR of ONWAAR
6. $n^2 = O(2^n)$: WAAR of ONWAAR
7. $42 = O(1)$: WAAR OF ONWAAR

2. Heaps (2 punten)

Stel, we hebben een max-heap met de array implementatie zoals in het boek en het college beschreven; de elementen staan in een array $A[1 \cdots heapsize]$.

Geef duidelijke pseudocode met een korte toelichting in woorden voor de volgende procedure: CHANGEMAX(int q).

De methode CHANGEMAX krijgt een waarde q mee; de methode verandert de maximum waarde in A naar q, en zorgt er daarna weer voor dat de elementen in A zo verplaatst worden dat de max-heap eigenschap weer geldt.

Als er meerdere keys met dezelfde maximum waarde zijn, hoeft er maar één van deze keys te veranderen.

Je methode moet $O(\log n)$ tijd gebruiken als er $n = heapsize$ keys in A zijn opgeslagen. Je hoeft deze tijdgrens niet te beargumenteren.

Bij deze vraag moet je zelf alle (pseudo)code geven; je mag niet volstaan met te verwijzen naar methoden die in het college behandeld zijn en/of in het boek staan.

Voorbeeld 1: Stel in A zijn de waardes 1, 3, 5 opgeslagen (in een of andere volgorde). Na CHANGEMAX(4) staan in A de waardes 1, 3, 4.

Voorbeeld 2: Stel in A zijn de waardes 1, 5, 5 opgeslagen (in een of andere volgorde). Na CHANGEMAX(4) staan in A de waardes 1, 4, 5.

3. Analyse van Algoritmen (1.5 punten)

Maak een zo goed mogelijke tijd-analyse (worst-case) van methode LENTEBUI, in termen van grote O van een functie in n. Je beantwoording moet ook uitleg bevatten hoe je aan het antwoord komt.

```
methode LENTEBUI (int n, array A)
Input: integer n; integer array A[1 · n]
Uitvoer: integer ant
    ant = 0;
    r = 0;
    for i = 1 to 2 · n do
        j = 2;
        while (j < n) do
                j + +;
                r = ⌊n/j⌋ ;
                for k = 1 to r do
                    ant = ant + A[k] · j;
                    if (ant > 1000) then ant = ant − 1; endif;
                enddo;
        enddo;
        ant ++;
    enddo;
    return (ant);
```

4. Sorteren en analyse van algoritmen (2.5 = 0.75 + 0.75 + 1 punten)

In deze opgave gaan we de tijd analyseren van een methode HEAPMERGESORT. De invoer van deze methode is een array $A[1 \cdot n]$, en twee integers p, q met $1 \leq p \leq q \leq n$. Het algoritme is bedoeld om het gedeelte van A vanaf $A[p]$ tot en met $A[q]$ te sorteren.

Het algoritme kan drie verschillende methoden aanroepen:

- HEAPMERGESORT (d.w.z., zichzelf, op een deelstuk).
- MERGESORT. MERGESORT(A, s, t) voert het mergesort algoritme uit op het deelstuk van A dat begint met s en eindigt op t.
- MERGE. Als van A de deelstukken vanaf s tot en met m, en vanaf $m+1$ tot en met t gesorteerd zijn, dan worden deze door MERGE(A, s, m, t) 'gemerged' tot een gesorteerd stuk van s tot en met t.

methode HEAPMERGESORT (array A, int p, int q)
int n= A.length;
if (**not** ($(1 \leq p)$ **and** $(p \leq q)$ **and** $(q \leq n)$))
then *geef foutmelding (buiten de grenzen)*; stop.
endif;
if $(q - p > 1)$ **then**
int $m = \lfloor (p+q)/2 \rfloor$;
HEAPMERGESORT(A, p, m);
~~MERGE~~SORT(A, $m+1$, q);
MERGE(A, p, m, q);
endif

De intuitie: de 'linkerhelft' wordt recursief gesorteerd, terwijl de 'rechterhelft' gesorteerd wordt met behulp van het ~~merge~~sort algoritme. Deze twee helften worden daarna ineengeschoven tot een goed gesorteerd stuk.

(a) **Geef pseudocode voor de methode MERGE.**

(b) **Welk van de volgende recurrente betrekkingen geeft het beste aan hoeveel tijd het de HEAPMERGESORT methode kost om een array met n elementen te sorteren? Geef zowel het nummer van de betrekking, als een uitleg waarom die betrekking de juiste is.**

1. $T(n) = T(n/2) + \Theta(n \log n)$
2. $T(n) = 2T(n/2) + \Theta(n \log n)$
3. $T(n) = T(n/2) + \Theta(n)$
4. $T(n) = 2T(n/2) + \Theta(n)$
5. $T(n) = T(n/2) + \Theta(n^2)$
6. $T(n) = 2T(n/2) + \Theta(n^2)$
7. Een andere betrekking, nl. ... (geef degene die je denkt dat het is)

(c) **Hoeveel tijd gebruikt het HEAPMERGESORT algoritme om een array met n elementen te sorteren? Toon je antwoord aan met behulp van de mastertheorem.**

5. Analyse en ontwerp van algoritmen op lijsten (2.25 punten)

Deodaat en Reinout discusseren over de volgende data structuur. We hebben een gelinkte lijst L. head[L] verwijst naar het eerste element van L; ieder object in de lijst bevat een waarde *key*, en een verwijzing *next* naar het volgende object in de lijst (alleen bij het laatste object in de lijst verwijst *next* naar *nil*.) We hebben ook een waarde size[L]: deze geeft het aantal objecten in L aan.

Keys zijn integers. Er geldt bovendien dat de lijst *geordend* is, dus geldt voor ieder object x in L dat als next[x]$\neq$ *nil*, dan $key[x] \leq key[next[x]]$: de keys staan gesorteerd in de lijst van klein naar groot.

Deodaat heeft een aantal methoden gemaakt. De eerste is de methode FINDRANK:

```
methode FINDRANK(L, k)
    x = head[L];
    if (k > size[L])
    then return nil;
    else for i = 1 to k − 1 do
            x = next[x];
        enddo;
    endif;
    return x;
```

Deze methode levert het k-de element in L, als het bestaat.

(a) **Wat is de worst case (slechtste geval) tijd van de methode FINDRANK op een lijst met n elementen?** Geef de tijd in O-notatie of Θ-notatie (naar keuze), zo goed mogelijk, als functie van n en/of k. Leg je antwoord kort uit.

Om te zoeken of een bepaalde key in L zit, heeft Deodaat het volgende algoritme bedacht.

```
methode SearchLKey(L, r)
    for i = 1 to size[L] do
        x = FindRank(i) ;
        if (key[x] == r)
        then return true;
        endif;
    enddo;
    return false;
```

(b) **Wat is de worst case (slechtste geval) tijd van deze methode SearchLKey op een lijst met n elementen?** Geef de tijd in O-notatie of Θ-notatie (naar keuze), zo goed mogelijk. Leg je antwoord kort uit.

Reinout denkt dat hij een sneller algoritme heeft.

methode BinSearchLKey(L, r)
    $low = 1$;
    $up =$ size[L];
    while ($low < up$) **do**
        $mid = \lfloor (low + up)/2 \rfloor$;
        $x =$ FINDRANK(mid) ;
        if key[x] < [redacted] r
        then $low = mid + 1$
        else $up = mid$
        endif;
    enddo;
    $y =$ FINDRANK(low);
    if (key[y] == r)
    then return true
    else return false
    endif;

(c) **Wat is de tijd (worst case / slechtste geval) van deze methode BinSearchLKey op een lijst met n elementen?** Geef de tijd in O-notatie of Θ-notatie (naar keuze), zo goed mogelijk. **Leg je antwoord kort uit.**

(d) **Welk van de volgende algoritmen om te bepalen of er een element in L zit met een bepaalde key heeft je voorkeur?**

1. Methode SearchLKey. Als je deze kiest: Geef kort aan **waarom** dit je voorkeur heeft.
2. Methode BinSearchLKey. Als je deze kiest: Geef kort aan **waarom** dit je voorkeur heeft.
3. Een andere methode. Als je dit kiest: Geef aan **waarom** een andere methode beter geschikt zou zijn, en **beschrijf** ook je methode (duidelijk in woorden *of* in pseudocode *of* beide).