1e deeltentamen Datastructuren 2013/2014

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 (1.5 punt)

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. $\frac{n^2}{5} + 3n \log n = O(n^2)$: WAAR of ONWAAR
2. $\frac{n^2}{5} + 3n \log n = \Omega(n^2)$: WAAR of ONWAAR
3. $\frac{n^2}{5} + 3n \log n = \Theta(n^2)$:WAAR of ONWAAR
4. $3n^3 = \Omega(n^2 \log n)$: WAAR of ONWAAR
5. $3n^3 = O(n^2 \log n)$: WAAR of ONWAAR
6. $3 \cdot 3^{3n} = \Theta(3^n)$: WAAR of ONWAAR

Bij deze vraag hoef je dus alleen steeds WAAR of ONWAAR op te schrijven voor elk van de onderdelen.

2. Functies en Θ-notatie (0.5 punt)

Schrijf de volgende functie zo compact mogelijk in Θ-notatie. Voorbeeld: $3n^{\sqrt{4}} + 2n^2 = \ldots = \Theta(n^2)$

$$3n^3 \cdot \log(3\sqrt{n/3}) + 5n^3 + 400000 = \Theta(\cdots)$$

3. Analyse van algoritmen I: 1.5 punten

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

methode MIERENNEST (int n, array A)
Input: integer n; integer array $A[1 \cdot n]$
Uitvoer: integer ant

```
r = 0;
q = n;
while q > 1
do
    q = ⌊q/2⌋;
    r++;
    for i = 1 to n
    do
        r = r + A[q];
        for k = 1 to ⌈n/i⌉
        do
            r = r + A[k];
        enddo;
    enddo
enddo
ant = r;
return ant;
```

4. Een andere sortering (1.5 punten)

Bij een computerspel hebben spelers een *score* en een *level*. Beide zijn niet-negatieve integers.

We weten dat het aantal spelers groter is dan het hoogste level. Dus: als er n spelers zijn, dan voldoet ieder level ℓ aan: $0 \leq \ell < n$.

Scores en levels hebben niet iets met elkaar te maken: je kan bijvoorbeeld een hoge score en een laag level hebben.

Stel, we hebben n objecten: ieder object bevat de naam van een speler, de score van de speler, en het level van de speler. Stel dat deze objecten gesorteerd zijn op (niet-dalende) score. Als twee spelers dezelfde score hebben zijn ze gesorteerd op (niet-dalend) level. Hieronder is een voorbeeld.

naam	level	score
ferdinand	3	100
ambrosius	2	140
deodaat	4	140
reinoud	1	150
jacob	1	200

We willen nu de sortering veranderen, en willen dat de spelers gesorteerd worden op (niet-dalend) level eerst; als spelers hetzelfde level hebben dan moeten ze op (niet-dalende) score worden gesorteerd.

De onderlinge volgorde van spelers met dezelfde score en hetzelfde level is verder niet gespecifieerd.

Hier hoe de volgorde moet worden in het voorbeeld:

naam	level	score
reinoud	1	150
jacob	1	200
ambrosius	2	140
ferdinand	3	100
deodaat	4	140

(a) **Is het mogelijk om in $O(n)$ tijd de sortering te veranderen, als omschreven?** D.w.z., we zoeken een algoritme dat, gegeven de eerste soort sortering van de spelers, de tweede soort sortering maakt, en $O(n)$ tijd gebruikt. De vraag hier is dus of zo'n algoritme bestaat. (Antwoord bij deel (a) alleen met JA of NEE.)

(b) **Zo ja, leg *duidelijk* uit hoe dit kan in $O(n)$ tijd. Zo nee, leg uit waarom dit niet kan.** Je mag hierbij verwijzen naar resultaten die in het college of boek behandeld zijn.

5. Analyse van algoritmen II: 1.5 punten

Beschouw het volgende recursieve algoritme.

```
method Reiger (integer n)
    if n ≤ 3 then return 1
    else
        quartup = ⌊n/4⌋;
        quartdown = ⌈n/4⌉;
        return 1+ Reiger(quartup) + Reiger(quartdown)
    endif
```

In deze opgave gaan we de tijd van dit algoritme analyseren.

(a) **Stel een recurrente betrekking op** van de tijd van dit algoritme, als functie van n.

(b) **Hoeveel tijd kost dit algoritme**, in Θ-notatie, als functie van n? **Beargumenteer je antwoord, met hulp van de master-theorem, en je antwoord bij deel (a).**

Mocht onderdeel (a) niet lukken, dan kan je deelpunten voor deze vraag halen door de volgende recurrente betrkking op te lossen.

$$T(n) = 3T(n/3) + \sqrt{n}$$

6. Algoritme ontwerp: 1.75 punten

Een bioloog ontdekt een bijzonder soort lemmings. Deze lemmings hebben de eigenschap om in een nette rij achterelkaar te lopen. Vooraan de rij loopt altijd een mannetje; achteraan de rij loopt altijd een vrouwtje.

Op een gegeven moment ziet de bioloog een rij van 1000 lemmings. Hij zoekt nu een combinatie van een mannetje en vrouwtje die direct achterelkaar lopen.

Toon aan dat hij zo'n combinatie kan vinden door 15 of minder keer bij een lemming te kijken of het een mannetje of een vrouwtje is.

In meer formele zin: we hebben een array A met lengte 1000 van objecten van het type Lemming. Elk object heeft een Boolean variabele male; gegeven is dat A[1].male true is en A[1000].male false is. We zoeken een i, $1 \leq i < 1000$ zodat A[i].male true is en A[$i+1$].male false is. Daarbij mogen we van maximaal 15 objecten de waarde male inspecteren.

Geef een duidelijke beschrijving van je algoritme. Je mag hierbij niet verwijzen naar resultaten in het boek/college.

7. Min-heap-checker: 1.75 punten

De opleiding Informatica van de Universiteit van Harderwijk gebruikt geen domjudge voor het programmeerpracticum, maar heeft een systeem dat automatisch datastructuren controleert. In één van de opgaven moeten de studenten een min-heap maken. Wat we nu willen is een methode die controleert of een gegeven array voldoet aan de eigenschappen van een min-heap.

(a) **Geef een algoritme dat voldoet aan de volgende voorwaarden**:

- De invoer is een array $A[1 \cdots n]$ en een integer A.heapsize.
- De uitvoer is een Boolean: *true*, als de array voldoet aan de eisen van de behandelde implementatie van een min-heap, en *false* als de array daar niet aan voldoet.
- Het algoritme gebruikt lineaire tijd

Hierbij mag je niet verwijzen naar resultaten in het college behandeld: je moet ook subroutines die je eventueel nodig hebt zelf geven. (Je kan die ten koste van deelpunten van deze vraag weglaten.)

(b) **Leg ook kort uit in woorden hoe je algoritme werkt**.

Je hoeft de tijdgrens van je algoritme niet aan te tonen. Je hoeft ook de correctheid van je algoritme niet aan te tonen.