Tentamen Complexiteit
Dinsdag 4 juli 2017, 14.00 – 17.00 uur

Geef een *duidelijke* toelichting bij al je antwoorden! Veel succes!

Opgave 1. (4+6+8+3+4=25 punten)
We hebben een array $A = A[1], A[2], \ldots, A[n]$ dat $n \geq 2$ verschillende getallen bevat.

a. Gegeven een beslissingsboom corresponderend met een algoritme dat gebaseerd is op arrayvergelijkingen. Wat stelt –in termen van het algoritme– een pad van de wortel naar een blad voor, wat stelt de hoogte van zo'n boom voor en wat kun je zeggen over de bladeren?
b. Bekijk het volgende algoritme voor het vinden van het minimum en het maximum van A: bepaal eerst de kleinste waarde met behulp van het gebruikelijke algoritme; bepaal vervolgens op dezelfde manier de grootste waarde uit de resterende $n-1$ elementen. Samen kost dit $n-1+n-2 = 2n-3$ arrayvergelijkingen.
Teken de beslissingsboom corresponderend met dit algoritme voor $n = 3$. Laat takken die nooit gevolgd worden weg.

We bekijken vanaf nu een speciaal soort arrays, met n even en $n \geq 2$. Zo'n array heeft de volgende eigenschap: voor alle oneven posities i geldt dat $A[i] < A[i+1]$. *Voorbeeld* met $n = 10$: $14, 17, 8, 12, 6, 11, 4, 9, 10, 15$. Gevraagd: het minimum en het maximum van A.

c. Bewijs met behulp van een *beslissingsboomargument* dat elk algoritme voor bovenstaand probleem dat is gebaseerd op arrayvergelijkingen, voor de bekeken soort invoerrijtjes ten minste $\lceil 2 \lg n \rceil - 2$ vergelijkingen moet doen in de worst case. Formuleer je bewijs zorgvuldig en geef aan welke stellingen/eigenschappen je gebruikt.
d. Beschrijf in woorden in maximaal 3 regels een voor de hand liggend algoritme voor het vinden van het minimum en het maximum van een array van bovenbeschreven soort. Het moet gebaseerd zijn op arrayvergelijkingen en $n-2$ arrayvergelijkingen doen. Geef ook aan waarom je algoritme zoveel arrayvergelijkingen doet.
e. (i) Op college is bewezen dat het vinden van het minimum en het maximum van een rij met n verschillende waarden in de worst case altijd minstens $\lceil \frac{3n}{2} \rceil - 2$ arrayvergelijkingen kost. Leg uit waarom het algoritme uit **d** niet in tegenspraak is met deze ondergrens.
(ii) Geef aan of de volgende bewering correct is: het algoritme uit **d** is niet optimaal, want $n-2$ is (voor $n \geq 8$) echt groter dan de bewezen ondergrens uit **c**. Licht je antwoord toe!

Opgave 2. (5+10=15 punten)
We bekijken *Shellsort* met sprongafstanden (= stapgroottes) $\frac{n}{2}, \frac{n}{2^2}, \frac{n}{2^3}, \ldots, 2, 1$.

a. Leg in woorden uit wat Shellsort doet en waarom het correct sorteert.
b. Bewijs dat het aantal arrayvergelijkingen dat Shellsort met bovengenoemde sprongafstanden doet in de worst case $O(n^2)$ is. Neem hierbij aan dat $n = 2^k$ een 2-macht is.

Hint bij het bewijs: laat zien dat het $\frac{n}{2^i}$-sorteren $O(n \cdot 2^i)$ arrayvergelijkingen kost in de worst case ($i = 1, 2, \ldots, k$). Je mag verder gebruiken dat $\sum_{i=0}^{k} 2^i = 2^{k+1} - 1$.

Opgave 3. (5+5+5+8=23 punten)
Gegeven is een array $A = A[1], A[2], \ldots, A[n]$ dat $n > 1$ gehele getallen bevat. We bekijken een algoritme dat de index van de kleinste waarde oplevert. Indien de kleinste waarde meerdere keren voorkomt wordt de index van het eerste voorkomen vanaf links geretourneerd. Het algoritme werkt door voor $A[i]$ te tellen hoeveel array-elementen kleiner zijn dan $A[i]$. Het algoritme is als volgt.

```
(1)   i := 1; gevonden := False;
(2)   while i < n and not gevonden do
(3)       j := 1; kleiner := 0;
(4)       while j ≤ n and kleiner = 0 do
(5)           if j ≠ i then
(6)               if A[j] < A[i] then
(7)                   kleiner := kleiner + 1;
                  fi
              fi
(8)           j := j + 1;
          od
(9)       if kleiner = 0 then
(10)          gevonden := True;
          fi
(11)      i := i + 1;
      od
(12)  if gevonden then
(13)      return i − 1;
      else
(14)      return n;
```

Het *vergelijken* van array-elementen in regel (6) (binnenste while-loop) is maatgevend voor de complexiteit.

a. Hoeveel arrayvergelijkingen doet het algoritme voor het rijtje $3, 2, 3, 4, 1$? Geef voor elke ronde $i = 1$ t/m 4 het aantal arrayvergelijkingen dat wordt gedaan.
b. Leg uit waarom het algoritme voor elke invoer ten minste $n - 1$ arrayvergelijkingen doet. Onderscheid hierbij twee situaties: (i) de kleinste waarde is uniek en staat achteraan op positie n, en (ii) de kleinste waarde komt voor op een positie $< n$.
c. Hoeveel vergelijkingen van array-elementen (regel (6)) worden er gedaan in het *beste* geval, voor algemene n? Voor wat voor invoerrijtjes komt dat voor? Geef *alle* gevallen. Maak hetzelfde onderscheid als in **b** en leg op basis van / in termen van het algoritme uit hoe je aan je uiteindelijke antwoord komt.
d. Hoeveel vergelijkingen van array-elementen (regel (6)) worden er gedaan in het *slechtste* geval, voor algemene n? Voor wat voor invoerrijtjes komt dat voor? Geef *alle* gevallen

en leg op basis van het algoritme uit hoe je aan je antwoord komt. Geef tevens een illustratief voorbeeld van een worst case rijtje met $n = 7$.
Tip: onderzoek eerst wat het betekent voor A als de buitenste while-loop zo vaak mogelijk moet worden gedaan en doe daarna hetzelfde voor de binnenste while-loop.

Opgave 4. (4+11=15 punten)
Gegeven is een array $A = A[1], A[2], \ldots, A[n]$, dat n verschillende getallen bevat en waarbij $n = 2^k$ voor zekere gehele $k \geq 0$. We gaan het array *recursief* sorteren als volgt.
Voor $n = 1$ hoeven we niets te doen. Voor $n \geq 2$ zorgen we ervoor dat alle elementen in de linkerhelft kleiner zijn dan alle elementen in de rechterhelft. Dit doen we door eerst voor *elke* $A[i]$ te tellen hoeveel array-elementen kleiner zijn dan $A[i]$ en zo de twee middelste waardes in grootte te bepalen. Vervolgens wisselen we die naar hun goede positie (plek $\frac{n}{2}$ en plek $\frac{n}{2} + 1$). Daarna lopen we met twee indices door het array: de ene van links naar rechts in de linkerhelft, de andere van rechts naar links in de rechterhelft. Elementen die in de verkeerde helft staan (bijvoorbeeld $A[j]$ met $A[j] < A[\frac{n}{2}]$ staat in de rechterhelft) worden twee aan twee verwisseld. Daarna staat het array in de gewenste vorm. Vervolgens sorteren we recursief de linker- en de rechterhelft. Het aantal arrayvergelijkingen dat dit algoritme doet noemen we $S(n)$. Deze $S(n)$ voldoet aan de volgende recurrente betrekking:

$$S(n) = \begin{cases} 0 & n = 1 \\ 2S(\frac{n}{2}) + n^2 - 2 & n \geq 2, n = 2^k \end{cases}$$

a. Verklaar de term $n^2 - 2$.
b. Los de recurrente betrekking exact op door deze herhaald in zichzelf in te vullen (substitutiemethode). Schrijf $S(n)$ als functie van n. Bewijs vervolgens met behulp van volledige inductie dat de aldus gevonden oplossing inderdaad voldoet.
Je mag gebruiken dat $\sum_{i=0}^{\ell-1}(\frac{1}{2})^i = 2(1 - \frac{1}{2^\ell})$ en $\sum_{i=0}^{\ell-1} 2^i = 2^\ell - 1$

Opgave 5. (10+4+8=22 punten)
We bekijken het volgende beslissingsprobleem:

Partitie: Gegeven een eindige verzameling natuurlijke getallen $X \subseteq \mathbb{N}$. Kan X worden opgesplitst in twee disjuncte deelverzamelingen $Y \subseteq X$ en $X \setminus Y$ zodanig dat de som der elementen uit Y gelijk is aan de som der elementen uit $X \setminus Y$?

Een *voorbeeld:* $X = \{2, 3, 4, 5, 6, 7, 8, 9, 10\}$ is op te splitsen in $\{4, 6, 7, 10\}$ en $\{2, 3, 5, 8, 9\}$, beide met som der elementen gelijk aan 27. X is dus een ja-instantie voor Partitie.

a. Toon aan dat Partitie $\in \mathcal{N}P$ door een *niet-deterministisch polynomiaal* algoritme voor Partitie te geven. Het algoritme heeft dus als invoer een eindige verzameling natuurlijke getallen X. Het algoritme moet "ja" opleveren dan en slechts dan als X een ja-instantie is voor Partitie (&).
Leg onder andere uit wat in elke fase van het algoritme gebeurt, en in het bijzonder wat in fase 2 wordt gecontroleerd en hoe, en hoeveel stappen dat kost. Leg ook uit waarom je niet-deterministische algoritme aan (&) voldoet en waarom het polynomiaal is in $|X|$.

b. (i) Wat wordt verstaan onder een polynomiale reductie van een beslissingsprobleem P naar een beslissingsprobleem Q?
(ii) Wanneer is een beslissingsprobleem P NP-hard?
(iii) Wanneer is een beslissingsprobleem P NP-volledig? (De definitie van $\mathcal{NP}$ hoeft niet te worden gegeven.)

Van college is bekend dat SAT $\in \mathcal{NPC}$ en dat DNFSAT $\in \mathcal{P}$. Verder is Partitie $\in \mathcal{NP}$.

c. Beantwoord de volgende vragen en geef bij elk antwoord een duidelijke uitleg.
(i) Professor K. Nap heeft een polynomiale reductie bedacht van DNFSAT naar SAT. Als zijn reductie correct is, heeft hij dan iets bijzonders bewezen?
(ii) Hij beweert ook bewezen te hebben dat SAT $\leq_{\mathrm{P}}$ DNFSAT. Leg uit waarom vrijwel iedereen denkt dat zijn bewijs niet klopt.
(iii) Als de professor wil aantonen dat Partitie $\in \mathcal{NPC}$, moet hij dan bewijzen dat Partitie $\leq_{\mathrm{P}}$ SAT of dat SAT $\leq_{\mathrm{P}}$ Partitie?

EINDE