Tentamen Complexiteit
Dinsdag 29 mei 2007, 14.00-17.00 uur

Geef een *duidelijke* toelichting bij al je antwoorden!

Opgave 1. (20 punten)

a. (5 punten)
Leg uit wat een beslissingsboom is voor algoritmen gebaseerd op arrayvergelijkingen: wat stellen de interne knopen en de bladeren voor, een pad van de wortel naar een blad, linker- en rechtersubboom van een knoop, de hoogte van de boom, etcetera.
b. (5 punten)
Bewijs met behulp van een beslissingsboomargument dat *elk* algoritme dat van n verschillende waarden het maximum *en* het minimum bepaalt (m.b.v. arrayvergelijkingen) ten minste $\Omega(\lg n)$ vergelijkingen moet doen in de worst case. Gebruik hierbij dat voor een binaire boom met hoogte h en b bladeren geldt dat $h \geq \lceil \lg b \rceil$.
c. (5 punten)
Gegeven een array A met n verschillende waarden $A[1], A[2], \ldots, A[n]$. Een *inversie* is een paar $(A[i], A[j])$ met $i < j$ en $A[i] > A[j]$, m.a.w. een paar dat verkeerd om staat indien we oplopend willen sorteren.
Bewijs nu het volgende: elk algoritme dat A oplopend sorteert met behulp van arrayvergelijkingen en dat na elke vergelijking ten hoogste één inversie opheft, moet in de worst case ten minste $\frac{1}{2}n(n-1)$ vergelijkingen doen.
d. (5 punten)
Zowel voor Bubblesort als voor Quicksort geldt dat ze ten minste $\frac{1}{2}n(n-1)$ arrayvergelijkingen moeten doen in de worst case.
Kan men dit voor Bubblesort op grond van de stelling uit **c.** concluderen? En voor Quicksort? Motiveer duidelijk je antwoord. Als het antwoord "nee" luidt moet je een ander, eenvoudig, argument geven dat het gestelde bewijst.

Opgave 2. (15 punten)

Het volgende *recursieve* algoritme bepaalt het aantal inversies (zie **1.c.**) in een gegeven array A met n verschillende gehele getallen. We nemen aan dat $n = 2^k$ voor zekere gehele $k \geq 0$.
Als $n \geq 2$ bepalen we eerst recursief het aantal inversies in de linkerhelft en in de rechterhelft. Vervolgens moeten we nog paren $(A[i], A[j])$ bekijken met $A[i]$ in de linkerhelft en $A[j]$ in de rechterhelft van het array. Noem het aantal arrayvergelijkingen dat dit algoritme doet $S(n)$.

a. (5 punten)
Leg duidelijk uit waarom $S(n)$ voldoet aan de volgende recurrente betrekking:

$$S(n) = \begin{cases} 0 & n = 1 \\ 2S(\frac{n}{2}) + \frac{1}{4}n^2 & n \geq 2,\ n = 2^k \end{cases}$$

b. (10 punten)
Los de recurrente betrekking uit **a.** op door deze herhaald in zichzelf in te vullen en bewijs met behulp van volledige inductie dat de aldus gevonden oplossing (uitgedrukt in n) inderdaad voldoet.
Hint bij het uitrekenen: haal uiteindelijk een factor $\frac{n^2}{2}$ buiten haakjes, en gebruik dat:

$$\frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \cdots + \frac{1}{2^k} = \sum_{i=1}^{k} \frac{1}{2^i} \quad = \quad 1 - \frac{1}{2^k}$$

Opgave 3. (30 punten)

Gegeven is een array A dat n gehele getallen, $A[1], A[2], \ldots, A[n]$, bevat. Hier is $n \geq 2$ een *even* geheel getal. Het array bevat precies $\frac{n}{2}$ negatieve (< 0) getallen en $\frac{n}{2}$ positieve (> 0) getallen. De bedoeling is om alle negatieve getallen op de oneven posities in het array te zetten, en alle positieve getallen op de even posities.
Onderstaand algoritme doet dit door eerst (regel 1 t/m 5) een keer alle paren $(A[i], A[i+1])$ (i oneven) af te lopen en de twee elementen te verwisselen indien op de even positie een negatief getal staat. Na afloop hiervan zijn de paren $(A[i], A[i+1])$ met i oneven dus alleen nog van de vorm $(< 0, > 0)$, $(< 0, < 0)$ of $(> 0, > 0)$. Vervolgens wordt nogmaals door het array gelopen om de getallen die nog niet goed staan op een juiste plek te zetten (regel 6 t/m 13). De functie `wissel`(a, b) verwisselt uiteraard de waarden van zijn argumenten. Merk op dat twee van de drie while-loops for-loops voorstellen.
Het algoritme gaat als volgt:

```
(1)   i := 1;
(2)   while i ≤ n − 1 do
(3)       if A[i + 1] < 0 then
(4)           wissel(A[i], A[i + 1]);
          fi
(5)       i := i + 2;
      od
(6)   i := 1;
(7)   while i ≤ n − 1 do
(8)       if A[i] > 0 then  // deze moet naar een even positie
(9)           j := 2;
(10)          while (A[j] > 0) do
(11)              j := j + 2;
              od
(12)          wissel(A[i], A[j]);
          fi
(13)      i := i + 2;
      od
```

a. (6 punten)
Laat zien dat het aantal keren dat getest wordt of een array-element positief of negatief is (regel 3, 8 en 10) maatgevend is voor de complexiteit van het algoritme. Doe dit door het aantal keren dat een regel wordt gedaan in verband te brengen met het aantal keren dat de test in regel (3) of de test in regel (8) of de test in regel 10 wordt uitgevoerd.
b. (4 punten)
In regel 10 wordt niet getest of j het array uitloopt. Waarom is de test $j \leq n$ daar niet nodig? Beantwoord hiertoe eerst de volgende vraag: wat kun je zeggen over het aantal paren $(A[i], A[i+1])$ —met i oneven— van de vorm $(< 0, < 0)$ versus die van de vorm $(> 0, > 0)$?
c. (5 punten)
Hoe vaak worden vergelijkingen van de vorm $A[..] > 0$ of $A[..] < 0$ (dus regel 3, 8 en 10 samen) minimaal (best case) gedaan? En voor wat voor soort invoerrijtjes komt dat voor? Geef alle gevallen.
d. (10 punten)
Hoe vaak worden vergelijkingen van de vorm $A[..] > 0$ of $A[..] < 0$ maximaal (worst case) gedaan? En voor wat voor soort invoerrijtjes komt dat voor? Geef alle gevallen.
Neem voor het gemak aan dat n deelbaar is door 4. Ter herinnering: $\sum_{i=1}^{m} i = \frac{1}{2}m(m+1)$.
e. (5 punten)
Een *eenvoudige* aanpassing van (regel 6 t/m 13 van) het algoritme levert al een ordeverbetering op van de worst case complexiteit. Geef aan wat er veranderd moet worden en hoe, en waarom die aanpassing inderdaad een ordeverbetering geeft.

Opgave 4. (35 punten)

We bekijken de volgende twee beslissingsproblemen:

Subset Sum (SUM): Gegeven een getal $t \in \mathbb{N}$ en een verzameling $S = \{s_1, s_2, \ldots, s_n\}$ met alle $s_i \in \mathbb{N}$. Bestaat er een deelverzameling $S' \subseteq S$ met $\Sigma_{s \in S'}\ s = t$?

Partition (Part): Gegeven een verzameling $S = \{s_1, s_2, \ldots, s_m\}$, met alle $s_j \in \mathbb{N}$. Kan S worden opgesplitst in twee disjuncte deelverzamelingen zodat de som van de elementen uit de ene deelverzameling gelijk is aan de som der elementen van de andere?

Een *voorbeeld* bij Part: $S = \{2, 3, 4, 5, 6, 7, 8, 9, 10\}$ is op te splitsen in $\{4, 6, 7, 10\}$ en $\{2, 3, 5, 8, 9\}$, beide met som der elementen gelijk aan 27.

a. (10 punten)
Toon aan dat Part $\in \mathcal{NP}$ door een *niet-deterministisch polynomiaal* algoritme voor Part te geven. Het algoritme heeft dus als invoer een verzameling positieve gehele getallen S, en moet "ja" opleveren dan en slechts dan als S een ja-instantie is. 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.

We bekijken een transformatie T van instanties van SUM naar instanties van Part.
Gegeven een instantie $< S, t >$ van SUM, dus een verzameling $S = \{s_1, s_2, \ldots, s_n\}$ met $s_i \in \mathbb{N}$ en een getal $t \in \mathbb{N}$. We beelden deze als volgt af op een instantie van Part. Bereken $A = \sum_{i=1}^{n} s_i$, en laat $T(< S, t >) = S^* = \{s_1, s_2, \ldots, s_n, b, c\}$, waarbij $b = 2A - t$ en $c = A + t$.
Het is duidelijk dat deze transformatie polynomiaal is.

b. (4 punten)
Bewijs nu het volgende. Als $\{s_1, s_2, \ldots, s_n, b, c\}$ met b en c als boven, op te splitsen is in twee deelverzamelingen met dezelfde som s^*, dan geldt:

1. $s^* = 2A$
2. b en c zitten niet in dezelfde deelverzameling (en dus zit b in de ene en c in de andere).

c. (10 punten)
Toon aan dat geldt: $< S, t >$ is een ja-instantie van SUM $\Longleftrightarrow$ $S^* = T(< S, t >)$ is een ja-instantie van Part. Ofwel: S bevat een deelverzameling met som der elementen gelijk aan $t \Longleftrightarrow S^*$ is op te splitsen in twee disjuncte deelverzamelingen waarbij de som der elementen van beide even groot is.
Bij "$\Longleftarrow$" kun je **b.** gebruiken.
d. (5 punten)
Uit bovenstaande volgt dat SUM $\leq_P$ Part. Laat zien dat het omgekeerde ook geldt, dat wil zeggen: Part $\leq_P$ SUM. Gebruik een eenvoudig argument om dit aan te tonen.
e. (2 punten)
Wanneer is een beslissingsprobleem P NP-hard? En wanneer NP-volledig? (Geef de definitie van NP-hard/ NP-volledigheid; de definitie van NP hoeft niet te worden gegeven.)
f. (4 punten)
Stel dat we weten dat SUM NP-volledig is. Welke van onderstaande twee beweringen is dan waar en waarom?

(i) Omdat bovendien Part $\in \mathcal{NP}$ en Part $\leq_P$ SUM is Part NP-volledig
(ii) Omdat bovendien Part $\in \mathcal{NP}$ en SUM $\leq_P$ Part is Part NP-volledig

Veel succes!