Tentamen Complexiteit
Dinsdag 15 augustus 2006, 10.00-13.00 uur

Opgave 1. (22 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)
Gegeven een binaire boom met hoogte h en b bladeren. Bewijs dat geldt: $h \geq \lceil \lg b \rceil$.
c. (5 punten)
Bewijs met behulp van een beslissingsboomargument dat *elk* algoritme dat van n verschillende waarden het maximum bepaalt m.b.v. arrayvergelijkingen ten minste $\lceil \lg n \rceil$ vergelijkingen moet doen in de worst case.
d. (7 punten)
De ondergrens verkregen in **c.** is niet scherp. Laat zien dat het vinden van het maximum in de worst case altijd ten minste $n - 1$ vergelijkingen kost.

Opgave 2. (23 punten)

Gegeven is een array A dat n verschillende getallen bevat, met $n \geq 0$ en n even. In deze opgave bekijken we eerst een methode om de grootste en de op een na grootste waarde hiervan te bepalen. Vervolgens gebruiken we dat om de rij recursief te sorteren.

a. (6 punten)
Laat $n \geq 2$. We bepalen de grootste en de op een na grootste waarde als volgt. Bepaal eerst (op de gebruikelijke manier) de grootste van de eerste $\frac{n}{2}$ getallen, en de grootste van de laatste $\frac{n}{2}$. Bepaal vervolgens de grootste van deze twee (dat is dan de grootste van alle n getallen), en tenslotte de gevraagde op een na grootste. Leg uit hoe deze in $\frac{n}{2} - 1$ vergelijkingen gevonden kan worden en laat zien dat het algoritme in totaal altijd $\frac{3}{2}n - 2$ vergelijkingen doet.
b. (4 punten)
Is dit algoritme voor het bepalen van de grootste en de op een na grootste optimaal voor wat betreft de worst case?

Bekijk nu het volgende *recursieve* algoritme voor het aflopend sorteren van een array met n verschillende getallen ($n \geq 0$, n even). Bepaal de grootste en de op een na grootste met de methode uit **a**. Verwissel die vervolgens met het voorste en het op een na voorste array-element. Nu staan $A[1]$ en $A[2]$ dus goed. Daarna sorteren we *recursief* de rest van het array. Noem het aantal vergelijkingen dat dit algoritme doet $S(n)$.
c. (3 punten)
Leg uit waarom $S(n)$ voldoet aan de volgende recurrente betrekking:

$$S(n) = \begin{cases} 0 & n = 0 \\ S(n-2) + \frac{3}{2}n - 2 & n \geq 2,\ n \text{ even} \end{cases}$$

d. (10 punten)
Los de recurrente betrekking uit **c.** op door deze herhaald in zichzelf in te vullen en bewijs met behulp van volledige inductie dat de aldus gevonden oplossing inderdaad voldoet. Laat bij het herhaald invullen de termen met n, $n-2$, $n-4$, etc. gewoon staan en veeg de 2-en bij elkaar. Dan is de algemene vorm beter te herkennen. Gebruik tenslotte dat $S(0) = 0$.
Hint bij het uitrekenen: gebruik –met even n– een van de volgende sommaties:

$$\sum_{i=1}^{\frac{n}{2}} i \;=\; \frac{n}{4} \cdot (\frac{n}{2} + 1); \quad \sum_{i=2,\ i \text{ even}}^{n} i \;=\; \frac{n}{2} \cdot (\frac{n}{2} + 1)$$

Opgave 3. (30 punten)

Gegeven een array A ($A[1], A[2], \ldots, A[n]$ met $n \geq 1$) dat n *verschillende* getallen bevat, en een positief geheel getal $1 \leq k < n$. Gevraagd wordt om het k-de element in grootte (dat is hier het op $k-1$ na grootste) te bepalen.
We bekijken hiertoe het volgende algoritme. Eerst worden de laatste k elementen van A oplopend gesorteerd met behulp van een efficiënte versie van *bubblesort*. Vervolgens worden de resterende $n-k$ elementen $A[n-k], A[n-k-1], \ldots, A[1]$ een voor een ingevoegd tussen de laatste k stuks. De laatste k elementen kunnen dus in elke stap variëren, maar ze blijven gesorteerd. Merk op dat de while-loops (4) t/m (11) en (13) t/m (23) gewoon for-loops voorstellen.

```
(1)   rechts := n;
(2)   while rechts > n − k do
(3)       i := rechts − 1; rechts := n − k;
(4)       j := n − k + 1;
(5)       while j ≤ i do
(6)           if A[j] > A[j + 1] then
(7)               wissel(A[j], A[j + 1]);
(8)               rechts := j;
(9)           fi
(10)          j := j + 1;
(11)      od
          // rechts geeft nu aan waar de laatste verwisseling plaatsvond
(12)  od
(13)  j := n − k;
(14)  while j ≥ 1 do
(15)      insert := A[j]; A[j] := A[n − k];
(16)      i := n − k + 1;
          // merk op dat n − k + 1 ≤ n
(17)      while i ≤ n and A[i] < insert do
(18)          A[i − 1] := A[i];
(19)          i := i + 1;
(20)      do
(21)      A[i − 1] := insert;
(22)      j := j − 1;
(23)  od
(24)  return A[n − k + 1];
```

a. (7 punten)
Toon aan dat het *vergelijken* van array-elementen (tweede test uit regel (17)) een goede basisoperatie is voor *deel 2* van het algoritme (regel (13) t/m (24)). Laat hiertoe onder andere zien dat deze vergelijking altijd ten minste $n - k$ keer gedaan wordt. We spreken af dat in regel (17) de tweede test niet wordt gedaan als de eerste test al **False** oplevert.

Opmerking. Analoog kan men laten zien dat de test uit regel (6) een goede basisoperatie is voor deel 1 ((1) t/m (12)). Het aantal vergelijkingen van array-elementen (test regel (6) en tweede test regel (17) samen) is dus een goede maat voor de complexiteit van het algoritme.

b. (4 punten)
Ga na hoe het algoritme werkt op het voorbeeldrijtje $8, 4, 6, 1, 7, 3, 2, 5$ met $n = 8$ en $k = 4$. Geef het aantal arrayvergelijkingen en het aantal verwisselingen dat het algoritme in deel 1 doet, alsmede het aantal arrayvergelijkingen in deel 2. Licht je antwoorden toe.
c. (8 punten)
Hoeveel vergelijkingen (uitgedrukt in n en k) tussen array-elementen doet het algoritme in totaal in het beste geval en voor wat voor invoerrijtjes komt dat voor? Geef uitleg.
d. (7 punten)
Hoeveel vergelijkingen (uitgedrukt in n en k) tussen array-elementen doet het algoritme in het slechtste geval? Geef voor algemene n en k een voorbeeld van een invoerrijtje waarvoor het aantal vergelijkingen maximaal is. Geef vervolgens voor $n = 8$ en $k = 4$ nog een ander voorbeeld van een worst-case invoerrijtje. Geef een korte toelichting.
e. (4 punten)
Als we kijken naar de worst case complexiteit, is dit dan een efficiënt algoritme voor het selectie-probleem of niet? Is het algoritme optimaal? Beantwoord deze vraag voor de gevallen $k = 1$ en $k = n/2$ (n even) en motiveer je antwoord.

Opgave 4. (25 punten)

We bekijken de volgende twee beslissingsproblemen:

SAT: Gegeven een logische formule ϕ in CNF. (Dus ϕ is een AND van clausules waarbij elke clausule de OR van een willekeurig aantal literals is; een literal is een x_i of een $\neg x_i$.) Is er een waardering (waarheidstoekenning) van de in ϕ voorkomende variabelen x_i, zodanig dat ϕ wordt waargemaakt, dat wil zeggen dat per clausule minstens één literal waar is?

DoubleSAT: Gegeven een logische formule ϕ in CNF. Bestaan er *ten minste twee* verschillende waarderingen van de in ϕ voorkomende variabelen x_i die ϕ waarmaken?

a. (10 punten)
Toon aan dat DoubleSAT $\in \mathcal{NP}$ door een *niet-deterministisch polynomiaal* algoritme voor DoubleSAT te geven. Het algoritme heeft dus als invoer een logische formule ϕ, met gebruikte logische variabelen $x_1, x_2, \ldots, x_m$, en moet "ja" opleveren d.e.s.d.a. ϕ 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.

We bekijken de volgende eenvoudige transformatie T van instanties van SAT naar instanties van DoubleSAT.
Gegeven is ϕ, een logische expressie in conjunctieve normaalvorm. Noem de in ϕ voorkomende logische variabelen $x_1, x_2, \ldots, x_n$. We construeren hierbij een logische formule $T(\phi) = \psi$ (eveneens in conjunctieve normaalvorm), waarin behalve de x_i ook een nieuwe logische variabele y voorkomt. Deze ψ ziet er als volgt uit: $\psi = \phi \wedge (y \vee \neg y)$. Het is duidelijk dat de constructie van $T(\phi)$ uit ϕ in polynomiale tijd kan (*).

Klein *voorbeeld*:
$\phi = (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2 \vee x_3) \rightarrow \psi = (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (y \vee \neg y)$

b. (8 punten)
Toon aan dat geldt: ϕ is een ja-instantie van SAT $\iff$ $T(\phi)$ is een ja-instantie van DoubleSAT. Ofwel: er is een waardering van de $x_1, \ldots, x_n$ die ϕ waarmaakt $\iff$ er zijn ten minste twee waarderingen van de $x_1, \ldots, x_n, y$ die ψ waarmaken.
Samen met (*) volgt hier dan uit dat SAT $\leq_P$ DoubleSAT.
c. (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.)
d. (5 punten)
Welke van de twee volgende beweringen is waar en welke niet (waarom/waarom niet)? Indien niet, geef dan aan wat je nog zou moeten weten opdat de bewering wel waar is. Motiveer je antwoord en formuleer duidelijk gebruikte stellingen.
(i) Als gegeven is dat SAT $\in \mathcal{NPC}$, dan volgt uit **a.** en **b.** dat DoubleSAT NP-volledig is.
(ii) Als gegeven is dat SAT $\in \mathcal{NP}$ en DoubleSAT $\in \mathcal{NPC}$, dan volgt uit **b.** dat SAT NP-volledig is.

Veel succes!