**Tentamen Algoritmiek**
**Dinsdag 11 juni 2013, 10.00 – 13.00 uur**

Geef een **duidelijke toelichting** bij al je antwoorden.

**Puntenverdeling:** 1: 20; 2: 20; 3: 20; 4: 30; 5: 10; **Veel succes !**

**Opgave 1.** We bekijken het volgende tweepersoonsspel, dat wordt gespeeld door Bilbo en Gollem. We hebben een tabel van $m$ (rijen) bij $n$ (kolommen), waarin bij aanvang $n * m - 1$ willekeurige gehele getallen staan. Een van de vakjes is leeg. Tijdens het spel is steeds één van de vakjes speciaal; aan het begin van het spel is dit het unieke lege vakje, na elke volgende zet is dit weer een ander (leeg) vakje. De twee spelers, Bilbo (B) en Gollem (G), doen om de beurt een zet. We spreken af dat Gollem begint.
In elke beurt kiest de speler die aan de beurt is één getal uit de speciale rij of kolom, aangegeven door het speciale vakje. (Als het speciale vakje coördinaten $X$ en $Y$ heeft, moet dus een getal uit uit rij $X$ of kolom $Y$ gekozen worden.) Dit getal wordt dan uit de tabel verwijderd en opgeteld bij de score van de betreffende speler. Het vakje waaruit het getal weggehaald is wordt het nieuwe speciale vakje. Het spel is afgelopen zodra de huidige speciale rij en kolom geen getal meer bevatten, en er dus geen zet meer mogelijk is. Degene die bij het einde van het spel de hoogste score heeft, heeft gewonnen. Bij gelijke eindscores heeft de speler die niet begonnen is (dat is in dit geval dus Bilbo) gewonnen. Bij aanvang hebben beide spelers score 0.

Hieronder een mogelijk spelverloop, met $m = n = 3$, het speciale vakje aangegeven door `*` en andere lege vakjes met `-`. Onder elke stand staat de huidige score = (scoreG, scoreB).

```
 9   *  12    G      9   -  12    B     9   -  12    G     9   -  12
 8  10  14   --->    8  10  14   --->   8  10  14   --->   *  10  14  (**)
11  12   6          11   *   6          *   -   6          -   -   6
  (0,0)               (12,0)             (12,11)            (20,11)
```

In toestand `(**)` is B aan de beurt en is de score `(20,11)`.

**a.** Wat zijn voor dit spel toestanden en acties (voor algemene $m$ en $n$)?
**b.** Teken de toestand-actie-ruimte voor het geval $m = n = 3$, uitgaande van toestand `(**)`. Geef bij elke eindtoestand aan of deze winnend is voor B of voor G. Bepaal hieruit voor *elke* toestand in je toestand-actie-ruimte voor wie deze winnend is, en ten slotte of `(**)` winnend is voor B of G.
Toestanden waarvan de laatste zetten vastliggen hoef je niet verder uit te werken. Je kunt daar meteen bijzetten wie er wint (met de eindscore). Een voorbeeld:

```
-   -   -
-  10  14
-   -   *
 (32,26)
```

Stel dat G aan de beurt is. Deze moet de 14 pakken; vervolgens moet B de 10 wegnemen. De eindscore wordt dus `(46,36)` en de stand hiernaast is derhalve winnend voor G.

**c.** Stel dat Gollem de volgende gretige strategie gebruikt: pak altijd het/een grootst mogelijke getal. Laat zien dat deze *gretige* strategie voor hem niet hoeft te leiden tot winst. Geef daartoe als tegenvoorbeeld een 2 bij 3 tabel met 5 getallen, en leg uit waarom de gretige strategie voor de beginnende speler (Gollem) niet werkt.

**Opgave 2.** Gegeven is een binaire boom met ingang (ofwel: wortel) `wortel`, die gehele getallen bevat. Hierin is `wortel` een pointer naar een `knoop`, waarbij:

```
struct knoop {
   knoop* links;
   knoop* rechts;
   int waarde;
} // knoop
```

Een binaire boom heeft de heapeigenschap als in *elke* knoop geldt dat de waarde die in die knoop is opgeslagen groter is dan of gelijk is aan de waarde die is opgeslagen in zijn kinderen.

**a.** Schrijf een recursieve C++-functie `bool heap(knoop* wortel)`, die bepaalt of de boom met ingang `wortel` de heapeigenschap heeft.
**b.** Teken de met de rij 51 88 29 66 23 17 75 83 98 70 corresponderende complete binaire boom en breng deze m.b.v. de bottom-up methode *heapify* in hoopstructuur. Laat tussenstappen zien en geef ook de met het eindresultaat corresponderende rij.
**c.** Leg uit hoe het sorteeralgoritme Heapsort werkt. Voer ter illustratie daarbij de eerste drie stappen van Heapsort uit op de hoopstructuur/rij die in **b.** verkregen is. Na afloop van deze drie stappen moeten dus de drie grootste getallen op hun goede plek achteraan staan. Laat ook zien hoe de (complete) rij er na de derde stap uitziet.

**Opgave 3.** Gegeven een array $A = A[0], A[1], \ldots, A[n-1]$ dat $n$ ($\geq 2$) verschillende gehele getallen bevat. Neem aan dat $n$ een 2-macht is. We zoeken de *grootste* index $j$ waarvoor geldt dat $A[j] < j$. Voor $9, 4, 1, 7, 11, 13, 3, 5, 8$ is deze index gelijk aan 7. Als zo'n index niet bestaat moeten de te schrijven algoritmen $-1$ opleveren.
We zullen het probleem eerst oplossen met brute force en daarna met verdeel en heers.

**a.** Schrijf een eenvoudig brute force algoritme in C++ dat deze index oplevert.
**b.** Geef een divide-and-conquer algoritme in C++ voor het probleem. Het array dient hiervoor in twee gelijke delen te worden verdeeld. Schrijf hiertoe een *recursieve* C++-functie `int grootste2(int A[ ], int links, int rechts)` die het probleem oplost voor het deelarray $A$[`links`], ..., `A`[`rechts`] ter lengte een 2-macht.
**c.** We veronderstellen nu dat het array oplopend gesorteerd is. Geef een decrease-by-half algoritme in C++ voor het bepalen van de gevraagde index. Schrijf hiertoe een *recursieve* C++-functie `int grootste3(int A[ ], int links, int rechts)`. Leg uit waarom je algoritme werkt, dus waarom je telkens maar één van beide helften hoeft te bekijken. Gebruik daarbij wat gegeven is over het array (verschillende gehele getallen, oplopend gesorteerd).

*Bonus* (5 punten). Vergelijk de complexiteit van de drie methodes uit **a.**, **b.** en **c.**. Welke is het efficiëntst, welke het minst efficiënt en in welke gevallen (worst case, best case)? Licht je antwoord toe.

**Opgave 4.** We hebben een driehoek bestaande uit $n$ rijen, genummerd 1 t/m $n$, gevuld met positieve gehele getallen. Deze driehoek met getallen kan weergegeven worden als een $n$ bij $n$ array $D$, waarvan alleen de linkeronderdriehoek gevuld is. Gevraagd wordt de grootste som die je kunt krijgen door vanuit de top (dus $D[1][1]$) naar beneden te lopen naar het onderste niveau, waarbij je vanaf een positie in de driehoek alleen naar de twee posities er schuin onder kunt lopen. In het array $D$ betekent dit dat je vanuit $D[k][\ell]$ alleen $D[k+1][\ell]$ en $D[k+1][\ell+1]$ kunt bereiken.

Een voorbeeld met $n = 4$:

```
          2                          2
       5     4          ofwel:     5   4
    3     4     7                  3   4   7
 1     6     9     6                1   6   9   6
```

Een mogelijke wandeling van de top naar beneden gaat via 2, 5, 4 en 6, en levert als som 17 op. De optimale route loopt via 2, 4, 7 en 9 (in het array $D$ dus via posities $(1, 1)$, $(2, 2)$, $(3, 3)$ en $(4, 3)$) met som 22.

**a.** Geef in woorden of in pseudocode een exhaustive search algoritme voor dit probleem met algemene $n$. Hoeveel complete routes controleert je algoritme?
**b.** Leg, aan de hand van de Fibonacci-getallen, uit wat dynamisch programmeren inhoudt, wat het verschil is tussen top-down en bottom-up dynamisch programmeren, en wat (bij de berekening van de Fibonacci-getallen) het grote voordeel is van dynamisch programmeren in vergelijking met gewoon recursie.

We gaan bovenstaand probleem oplossen met bottom-up dynamisch programmeren. Hierbij gebruiken we een tweedimensionaal array $S$. Laat $S[i][j]$ de maximale som voorstellen die je kunt krijgen door van positie $(i, j)$ naar rij $n$ te lopen.

**c.** Leg uit waarom de volgende recurrente betrekking geldt voor $S$, met $1 \leq j \leq i \leq n$:

$$S[i][j] = \begin{cases} D[n][j] & \text{als } i = n \text{ en } 1 \leq j \leq i \\ \\ D[i][j] + \max\{S[i+1][j], S[i+1][j+1]\} & \text{als } 1 \leq j \leq i < n \end{cases}$$

**d.** Schrijf een algoritme in pseudocode (of in C++) dat, gebruikmakend van het array $S$ en de recurrente betrekking uit **c.**, de gevraagde maximale som berekent. Geef duidelijk aan in welke volgorde je het array $S$ vult. Vul vervolgens het array $S$ voor het voorbeeld.
**e.** Geef in woorden aan hoe je, behalve de maximale som zelf, ook de/een route via welke je die maximale som bereikt, kunt bepalen. Voor het voorbeeld zou dit de route $(1, 1)$, $(2, 2)$, $(3, 3)$, $(4, 3)$ moeten opleveren.

**Opgave 5.** Het algoritme van Dijkstra bepaalt voor gewogen grafen de (lengtes van) kortste paden vanuit een gegeven knoop naar alle andere knopen.
Pas het algoritme van Dijkstra toe op onderstaande graaf, beginnend in knoop 1. Geef voor elke stap van het algoritme duidelijk aan welke knoop erbij wordt gekozen in $U$ (= verzameling knopen waarvan de kortste afstand vanaf 1 bekend is) en welke labels door die keuze veranderen en hoe.
Geef ook de uiteindelijke boom van kortste paden met daarin de bijbehorende lengtes van de kortste paden vanaf 1 !