Opgaven Complexiteit dinsdag 19.2.2019

Opgave 1. Bekijk het volgende probleem: gegeven een reëel getal x en een geheel positief getal n, gevraagd x^n.
a. Geef de complexiteit van het meest voor de hand liggende algoritme hiervoor. Wat is de basisoperatie in dit geval en waarom?
b. Idem, nu voor het algoritme dat x herhaald kwadrateert en het antwoord uit de tussenresultaten samenstelt: $x^{13} = x^8 * x^4 * x$, waarbij overigens x^2 niet gebruikt wordt.
c. Is het antwoord van **b.** optimaal? Hint: bekijk x^{15}.

Opgave 2. Deze opgave gaat over het vermenigvuldigen van matrices.
a. Bereken het matrixproduct

$$\begin{pmatrix} 1 & 3 \\ 5 & 7 \end{pmatrix} \begin{pmatrix} 8 & 4 \\ 6 & 2 \end{pmatrix}$$

op de gebruikelijke manier (zie ook hieronder). Hoeveel $*$'s en hoeveel $+/-$'s heb je hiervoor nodig gehad?
Nu het algemene geval: gegeven twee $n \times n$-matrices A en B (met reële getallen). Bepaal het matrixproduct van deze twee matrices. Bekijk het "gebruikelijke" algoritme: dit algoritme berekent de componenten c_{ij} van de productmatrix AB via de formule $c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$ voor $1 \leq i, j \leq n$.
b. Hoeveel vermenigvuldigingen $(*)$ en hoeveel optellingen $(+/-)$ doet het "gebruikelijke" algoritme, uitgedrukt in n?
c. Dezelfde vraag, maar nu uitgedrukt in het aantal array-elementen, ook een mogelijke maat voor de grootte van de invoer.
d. Een vermenigvuldiging is i.h.a. tijdrovender dan een optelling. Dus er wordt voorgesteld om als basisoperatie het vermenigvuldigen van twee getallen te nemen. Is dit inderdaad altijd een goede maat voor de complexiteit? Hint: denk aan het geval dat de matrices allemaal gehele getallen bevatten.
e. Laat zien: de *complexiteit* van elk algoritme voor het vermenigvuldigen van twee n bij n matrices is altijd minstens $\Omega(n^2)$.
f. Er zit een ordeverschil tussen de complexiteit van het "gebruikelijke" algoritme en de ondergrens uit **e.** Betekent dit dat het algoritme niet optimaal is of dat de ondergrens niet scherp genoeg is?

Opgave 4. Gegeven een oplopend gesorteerd array A ($A[1]$, ..., $A[n]$) dat n gehele getallen bevat. Gegeven is verder een geheel getal X. We gebruiken een aangepaste versie van het zoekalgoritme uit de vorige opgave. Net als daar is het vergelijken van X met array-elementen ($A[index] < X$) een goede basisoperatie.

```
(1)  index := 1;
(2)  while index ≤ n and A[index] < X do
(3)      index := index + 1;
(4)  od
(5)  if index ≤ n and X = A[index] then
(6)      return index;
(7)  else
(8)      return −1;
(9)  fi
```

a. Wat is het aantal vergelijkingen in het beste geval en voor welke invoer komt dat voor?
b. Wat is het aantal vergelijkingen in het slechtste geval en voor welke invoer komt dat voor? Vergelijk dit met ongeordend zoeken (opgave 3).
c. Wat is het aantal vergelijkingen in het gemiddelde geval? Laat ook zien dat dit $\Theta(\frac{n}{2})$ is. Neem hierbij aan dat (1) als X in A voorkomt alle posities in het array even waarschijnlijk zijn en (2) als X niet in A zit alle "gaten" tussen de opeenvolgende array-elementen even waarschijnlijk zijn. Laat q de kans zijn dat X in A zit.

Opgave 6. *Bubblesort.* Bekijk het volgende algoritme om een array A met n gehele getallen op plekken 1 t/m n oplopend te sorteren:

```
(1)  Boven := n; Gewisseld := True;
(2)  while Gewisseld do
(3)      Boven := Boven − 1; Gewisseld := False;
(4)      for j := 1 to Boven do
(5)          if A[j] > A[j + 1] then
(6)              Wissel(A[j], A[j + 1]); Gewisseld := True fi od
(7)  od
```

Hierbij is *Wissel* uiteraard een functie die de inhoud van zijn argumenten verwisselt.
a. Wat is hier de worst case, en hoeveel arrayvergelijkingen $A[j] > A[j+1]$ doet Bubblesort in dit geval? Neem aan dat het array een permutatie van $\{1, 2, \ldots, n\}$ bevat.
b. Idem voor de best case.
c. Is het aantal arrayvergelijkingen een goede basisoperatie? Waarom (niet)?
d. We passen het algoritme aan door onder- en bovengrens van de for-loop variabel(er) te maken, en correct te updaten. Het idee is om te lopen vanaf de plek waar in de vorige ronde voor het eerst gewisseld werd (*Onder*) tot de plek waar in de vorige ronde voor het laatst gewisseld werd (*Boven*). Veranderen best case en worst case hiermee (essentieel)?

Opgave 7. *Insertion sort.*

```
(1)  for i := 2 to n do
         // nu A[i] op de juiste plek in A[1] ... A[i − 1] invoegen
(2)      x := A[i];
(3)      j := i − 1;
(4)      while j > 0 and A[j] > x do
(5)          A[j + 1] := A[j];
(6)          j := j − 1;
(7)      od
(8)      A[j + 1] := x;
(9)  od
```

Opgaven afkomstig uit dictaat:
www.liacs.leidenuniv.nl/~graafjmde/COMP/

a. Laat zien dat het aantal arrayvergelijkingen (tweede test regel (4)) een goede maat is voor de complexiteit.
b. Hoeveel arrayvergelijkingen doet Insertion sort in de worst case?
c. Geef *alle* worst case gevallen voor Insertion sort (dus niet alleen de omgekeerd gesorteerde rij). Hoeveel zijn dat er? Zonder beperking der algemeenheid kun je volstaan met alleen invoerrijtjes met n verschillende waarden, zeg 1 t/m n te bekijken.