Enkele uitwerkingen van opgaven bij werkcollege 3 Algoritmiek

4. (a) Toestand: een collectie rechthoeken bestaande uit damstenen (of X-en), elk maximaal m bij n groot. Ook is van belang wie er aan de beurt is. Alternatief: een m bij n-bord waarbij elk vakje leeg is of een damsteen (X) bevat. De damstenen vormen rechthoeken. (In de begintoestand hebben we één rechthoek met $m * n$ damstenen, in de eindtoestand hebben we geen enkele steen meer over.)

   Een actie is het weghalen van een rij uit één der rechthoeken door H als deze aan de beurt is, resp. het weghalen van een kolom uit één der rechthoeken door V als deze aan de beurt is. Tevens het omklappen van de beurt.

   (b) Zie het bijgevoegde bestand tad2014-03.pdf. Het spel is winnend voor V, die de eerste (of laatste) kolom moet weghalen en in zijn volgende zet de middelste kolom (zie in het plaatje de dubbele lijnen).

   (c) Uit de toestand-actie-ruimte uit (b) zagen we al dat het geval 1 bij 3 winnend is voor V als die aan de beurt is. Hij neemt de middelste steen weg en wint in zijn volgende zet.

   Bekijk nu het geval 1 bij 5. Wil V winnen, dan moet hij in elk geval twee rijen overlaten (dus geen randkolom pakken). Als hij de tweede van links (of rechts, dat is symmetrisch) wegneem, blijven en een rij van 1 en een rij van 3 liggen. H kan maar één van de twee rijen weghalen. Er resteert dus X of XXX voor V. Beide gevallen zijn winnend voor V. Dus het geval 1 bij 5 is winnend voor V.

   Nu het algemene geval. Om te winnen moet V de rij splitsen. Merk verder op dat als V een zet doet, er in totaal een even aantal stenen blijft liggen. V kan dus altijd zorgen dat hij splitst in twee oneven stapels. Hij kan dit bijvoorbeeld doen door de de tweede kolom van links weg te halen, waarbij er een rij van één steen en een rij van $n-2$ stenen overblijven, beide van oneven lengte. H moet nu een van de twee rijen wegnemen, en er resteert weer een (kleinere) oneven rij met V aan de beurt. Als V deze strategie herhaalt zal hij uiteindelijk uitkomen op een rij van 1 steen, en wint.

   (d) Als n even is kan V ofwel de rij met één korter maken (en vervolgens haalt H die in zijn geheel weg, en wint dus meteen), of hij splitst de rij in een even en een oneven rij. Immers als hij een kolom weghaalt blijven er $n-1$ stenen liggen, en dat is oneven. Als H in dit geval de oneven rij in zijn geheel weghaalt blijft er een even rij liggen met V aan de beurt. Indien H deze strategie herhaalt komt V uiteindelijk uit op een 1 bij 2 rij, en dan verliest hij.

6. The basic operation is the assignment to `diff[i][j]`. It is performed $n \times n = n^2$ times. This is quadratic in the matrix order n. It is, however, linear in the total number of elements in the input matrices, which is $2n^2$, and hence linear in the input size.

7. The seven functions in the order of increasing order of growth:

   $$2\log_2(n+50)^5, \quad (\ln n)^3, \quad \sqrt{n}, \quad 0.05n^{10}+3n^3+1 \quad 3^{2n}, \quad 3^{3n}, \quad (n^2+3)!$$

   To see why:

   - Consider what happens to the functions $2\log_2(n+50)^5$, $(\ln n)^3$ and $\sqrt{n}$, when you replace n by n^2. The first function grows roughly with a factor 2, the

second function with a factor 8, the third function with a factor $\sqrt{n}$ (from $\sqrt{n}$ to n).

- Observe that $\sqrt{n} = n^{\frac{1}{2}}$, which grows much slower than n^{10}.
- Observe that $3^{2n} = 9^n$ and that $3^{3n} = 27^n$.
- Observe that $(n^2+3)!$ is the product of n^2+3 factors, most of which are larger than n.

8. (a) This algorithm computes the difference between the last number in array A that is greater than 100, and the last number in array A that is smaller than 100. If there is no number in A greater (smaller) than 100, we use the default value 0 instead. In fact, the variable `sumgreater` may be read as `somegreater`, (and similar for `sumless`), as it a quite arbitary choice from all numbers in A that are greater than 100.

   (b) The basic operation is one of the two comparisons between $A[i]$ and `val`. Both comparisons serve equally well as basic operation.

   (c) The basic operation is executed n times.

   (d) The efficiency class is $\Theta(n)$, i.e., linear in the input size.

   (e) An improvement could be to perform the second comparison only if the condition `(A[i]>val)` does not hold:

```
if (A[i]>val)
  sumgreater = A[i];
else
  if (A[i]<val)
    sumless = A[i];
```

   Indeed, if `(A[i]>val)` holds, then `(A[i]<val)` certainly does not hold, so it is no use testing this. With this improvement, the efficiency class remains $\Theta(n)$, since the first comparison is still performed for every element $A[i]$. In fact, if all elements $A[i]$ happen to be smaller than 100, then also the second comparison is still performed for every element $A[i]$.

   A more significant improvement would be to search the array A in reverse order, starting from $A[n-1]$, and to stop as soon as you find an element greater than `val`. This element is assigned to `sumgreater`. Repeat this search, this time stopping as soon as you find an element smaller than `val`, which then is assigned to `sumless`. Of course, you should also stop a search after having compared $A[0]$.

   In the worst case, this improved algorithm is still in $\Theta(n)$. In the best case, both searches stop after 1 or 2 comparisons, and thus in $\Theta(1)$ time. Assuming that the elements $A[i]$ have a constant (positive) probability of being larger than 100 and a constant (positive) probability of being smaller than 100, the average number of iterations of both searches is hardly dependent on n. Hence, the average case complexity is also in $\Theta(1)$.