Negende college complexiteit

25 maart 2008

Shellsort, Optimaal sorteren, $O(n)$-sorteren

```
(1)   h = n div 2; // h_t = ⌊n/2⌋ dus
(2)   while h > 0 do
(3)       for i := h + 1 to n do
(4)           temp := A[i]; j := i;
(5)           while j − h > 0 do
              // invoegen op de juiste plek
(6)               if temp < A[j − h] then
(7)                   A[j] := A[j − h]; // schuif
(8)                   j := j − h;
(9)               else
(10)                  "exit binnenste while";
(11)              fi
(12)          od
(13)          A[j] := temp; // zet neer
(14)      od
          // de rij is nu h-gesorteerd
(15)      h := h div 2; // oorspronkelijke keuze van Shell: h_i = ⌊h_{i+1}/2⌋
(16)  od
```

Regel (4) t/m (13) is Insertion sort op deelarrays.

Shellsort: voorbeeld

$h_3 = 6, h_2 = 3, h_1 = 1, n = 13$

$$81, 94, 11, 96, 12, 35, 17, 95, 28, 58, 41, 75, 15$$

$\downarrow$ 6-sorteren

$$15, 94, 11, 58, 12, 35, 17, 95, 28, 96, 41, 75, 81$$

$\downarrow$ 3-sorteren

$$15, 12, 11, 17, 41, 28, 58, 94, 35, 81, 95, 75, 96$$

$\downarrow$ 1-sorteren

$$11, 12, 15, 17, 28, 35, 41, 58, 75, 81, 94, 95, 96$$

Insertion sort doet hier **52** vergelijkingen;
Shellsort doet **44** vergelijkingen

De complexiteit van Shellsort (= aantal arrayvergelijkingen) hangt in hoge mate af van de gekozen stapgroottes. De analyse is in het algemeen extreem moeilijk en nog zeer incompleet.

1. Stapgroottes: $h_t = \lfloor \frac{n}{2} \rfloor$, $h_i = \lfloor \frac{h_{i+1}}{2} \rfloor$ voor $i = t-1, \ldots, 1$. Dan $t = \lfloor \lg n \rfloor$. Voor het gemak nemen we $n = 2^k$, dus $t = k$.

   Stelling A. Het aantal arrayvergelijkingen dat Shellsort met deze incrementserie doet is in de worst case $\Omega(n^2)$.

   Bij het bewijs. Het is voldoende om een bad case aan te geven waarvoor het aantal vergelijkingen $\Omega(n^2)$ is.

Stelling B. Het aantal arrayvergelijkingen dat Shellsort met deze increments doet is in de worst case $O(n^2)$.

Gevolg van A en B.
In de worst case doet Shellsort met Shells increments $\Theta(n^2)$ vergelijkingen.

2. Stapgroottes (Hibbard): $2^k-1, 2^{k-1}-1, \ldots, 7, 3, 1$ ($k = \lfloor \lg n \rfloor$). Merk op dat opeenvolgende increments hier relatief priem zijn (volgt uit: $h_{i+1} = 2h_i + 1$).

   Stelling (zonder bewijs). In de worst case doet Shellsort met Hibbards increments $O(n^{\frac{3}{2}})$ vergelijkingen.

3. Het kan nog beter!

Voorbeeld. Na 8-sorteren heeft 3-sorteren i.h.a. veel meer effect dan 4-sorteren.

8-gesorteerd, 36 inversies

$5, 2, 11, 7, 15, 3, 16, 6, 12, 9, 14, 8, 19, 13, 20, 10$

$\downarrow$ 4-sorteren

31 inversies

$5, 2, 11, 6, 12, 3, 14, 7, 15, 9, 16, 8, 19, 13, 20, 10$

8-gesorteerd, 36 inversies

$5, 2, 11, 7, 15, 3, 16, 6, 12, 9, 14, 8, 19, 13, 20, 10$

$\downarrow$ 3-sorteren

7 inversies

$5, 2, 3, 6, 7, 8, 9, 12, 11, 10, 13, 15, 14, 16, 20, 19$

Beter:

- Twee doorgangen: $h_2 \approx 1,72 \cdot \sqrt[3]{n},\ h_1 = 1$.
  Gemiddeld $O(n^{5/3})$

- Heel veel doorgangen: increments van de vorm $2^i \cdot 3^j$:
  $1, 2, 3, 4, 6, 8, 9, 12, 16, 18, 24, 27, 32, 36, 48, 54, 72, 81, \ldots$
  Worst case $O(n(\lg n)^2)$.

- Increments van de vorm $4^{j+1} + 3 \cdot 2^j + 1$:
  $1, 8, 23, 77, 281, 1073, 4193, 16577, \ldots$
  Worst case $O(n^{4/3})$.

- $\ldots$

Voor sorteeralgoritmen gebaseerd op arrayvergelijkingen is bewezen: het aantal vergelijkingen in de worst case is ten minste $\lceil \lg n! \rceil$

Vraag: hoe dicht kan men in de buurt van deze ondergrens komen?

Tabel:

n	2	3	4	5	6	7	8	9	10	11	12	...	21
$\lceil \lg n! \rceil$	1	3	5	7	10	13	16	19	22	26	29	...	66
M(n)	1	3	5	8	11	14	17	21	25	29	33	...	74
B(n)	1	3	5	8	11	14	17	21	25	29	33	...	74

M(n) = aantal vergelijkingen dat Mergesort doet in de worst case voor een rij met n elementen.
B(n)= aantal vergelijkingen dat Binary Insertion sort in de worst case doet voor een rij met n elementen.
Merk op dat $\lceil \lg 1! \rceil$ = M(1) = B(1)= 0.

Binary Insertion sort werkt als Insertion sort, maar gebruikt voor het zoeken van de plek waar $A[i]$ moet komen binair zoeken in plaats van lineair zoeken. (Zie ook opgave 31.)

Om $A[i]$ op de juiste plek in te voegen in $A[1], \ldots, A[i-1]$ zijn nu in het slechtste geval $\lceil \lg i \rceil$ vergelijkingen nodig. Dus:

$$B(n) = \Sigma_{i=2}^{n} \lceil \lg i \rceil \geq \lceil \Sigma_{i=2}^{n} \lg i \rceil = \lceil \lg n! \rceil$$

Er geldt zelfs:

$$\Sigma_{i=2}^{n} \lceil \lg i \rceil = n\lceil \lg n \rceil - 2^{\lceil \lg n \rceil} + 1,$$

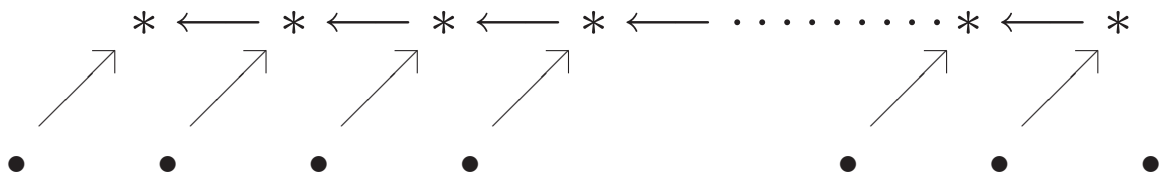
en dat is ook precies het aantal vergelijkingen dat Mergesort doet. Dus $B(n) = M(n)$.

Vraag: Kan het nog beter?
Antwoord: Ja !
Voorbeeld: 5 elementen kunnen met 7 vergelijkingen gesorteerd worden (Demuth, 1956).

De methode van Demuth kan gegeneraliseerd worden tot het algoritme Merge Insertion sort (Ford & Johnson).

1. Vergelijk de n elementen twee aan twee.

2. Sorteer de $\lfloor \frac{n}{2} \rfloor$ winnaars $*$ (de grootsten dus) recursief. Dit levert iets op als:



3. Voeg nu de $\lfloor \frac{n}{2} \rfloor$ verliezers (en de losse waarde als n oneven is) op de juiste plek in **via een of andere handige volgorde**.

Merge Insertion sort sorteert bijvoorbeeld:

- 21 elementen in 66 vergelijkingen (optimaal)
- 12 elementen in 30 vergelijkingen (optimaal)
- 10 elementen in 22 vergelijkingen (optimaal)

Vraag: is Merge Insertion sort optimaal?
Antwoord: nee, bijvoorbeeld $n = 47$.

Invoer: een array A met n getallen $A[1], \ldots, A[n]$.
Aanname: elke $A[i]$ is een geheel getal tussen 1 en k (voor zekere k).
Uitvoer: een array B, voorstellende het array A oplopend gesorteerd.

Het basisidee (als alle $A[j]$'s verschillen): bepaal voor elke $X = A[j]$ het aantal array-elementen kleiner dan X. Deze informatie kan dan gebruikt worden om X meteen op de juiste positie in het uitvoerarray te zetten. Pas dit idee aan voor de situatie waarin sommige waarden meer dan eens in A voorkomen.

```
for i := 1 to k do
    C[i] := 0;        // initialisatie
od
for j := 1 to n do
    C[A[j]] := C[A[j]] + 1;
    // telt aantal keer dat A[j] in A voorkomt
od
for i := 2 to k do
    C[i] := C[i] + C[i − 1];
od
// C[i] bevat nu het aantal getallen ≤ i uit A
for j := n downto 1 do      // !!
    B[C[A[j]]] := A[j];
    C[A[j]] := C[A[j]] − 1;
od
```

Voorbeeld

A	$=$	3	6	4	1	3	4	1	4	$n = 8$
C	$=$	0	0	0	0	0	0			$k = 6$
C	$=$	2	0	2	3	0	1			$\longleftarrow$ na 2^e for

$C[i] =$ aantal keer dat i in A voorkomt

C	$=$	2	2	4	7	7	8			$\longleftarrow$ na 3^e for

$C[i] =$ aantal array-elementen $\leq i$

De complexiteit van Counting sort wordt bepaald door het aantal toekenningen aan array-elementen, en dat zijn er $O(n + k)$. Indien $k = O(n)$ is de complexiteit dus $O(n)$.

Extra geheugenruimte is ook $O(n + k)$.

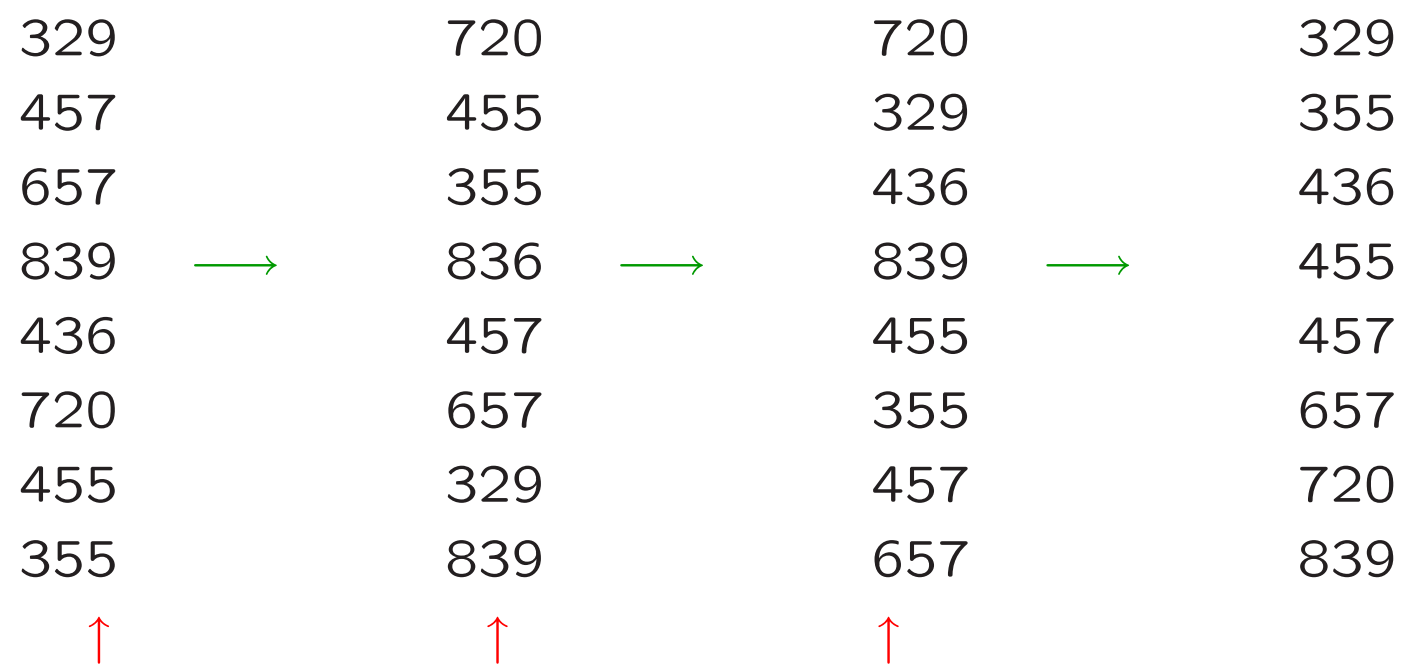
Counting sort is een stabiele sorteermethode, dat wil zeggen: gelijke waarden uit het invoerarray A komen in precies dezelfde volgorde in het uitvoerarray B te staan als ze in A stonden.

De sorteermethode Radix sort sorteert n getallen, elk van d cijfers, waarbij elk cijfer een waarde heeft tussen 0 en $k-1$ (bijvoorbeeld $k = 2$, of $k = 10$).
De getallen worden gesorteerd door ze achtereenvolgens op het i-de cijfer te sorteren, te beginnen bij het minst significante cijfer. Het gebruik van een stabiele sorteermethode voor het sorteren op het i-de cijfer is essentieel.

Radixsort(A, d):

for $i := 1$ **to** d **do**
// minst significante cijfer eerst, dus van rechts naar links
sorteer A op het i-de cijfer
// met een stabiele (!!) methode
od

Voorbeeld


329
457
657
839
436
720
455
355
sorteer op
1e cijfer
720
455
355
836
457
657
329
839
sorteer op
2e cijfer
720
329
436
839
455
355
457
657
sorteer op
3e cijfer
329
355
436
455
457
657
720
839

Indien de gebruikte sorteermethode voor het sorteren per cijfer niet stabiel is kan het fout gaan:

329		720		720		355
457		455		329		329
657		355		436		457
839	$\longrightarrow$	436	$\longrightarrow$	839	$\longrightarrow$	436
436		457		455		455
720		657		355		657
455		329		457		720
355		839		657		839
$\uparrow$		$\uparrow$		$\uparrow$		

- Als sorteermethode per cijfer kunnen we Counting sort gebruiken, aangezien de cijfers tussen 0 en $k-1$ zitten.

- In dat geval kost elke ronde $O(k+n)$ stappen. In totaal (d rondes) dus $O(dk+dn)$. En dat is $O(n)$ als d een constante is en $k = O(n)$.

- Een nadeel van deze methode is dat er net als bij Counting sort $O(n+k)$ extra geheugenruimte nodig is.