Zevende college complexiteit

26 maart 2019

Shellsort, optimaal sorteren

Merge insertion sort, $O(n)$-sorteren

Shellsort (Shell 1959)

Shellsort* was een van de eerste sorteeralgoritmen met worst case complexiteit minder dan kwadratisch.

Shellsort sorteert in elke ronde deelrijtjes. In het begin veel korte rijtjes, waarbij de elementen uit een rijtje ver van elkaar liggen. Later weinig lange rijtjes, waarbij de elementen uit een rijtje dicht bij elkaar liggen. De rij wordt zo als het ware voorgesorteerd. In de laatste ronde wordt de rij dan als geheel gesorteerd. Shellsort sorteert met behulp van vergelijk-verwissel / compare-exchange (indien nodig) operaties.

*Donald Shell

Definitie

Een rij $A[1], A[2], \ldots, A[n]$ heet k-gesorteerd als geldt: $A[i] \leq A[i+k]$ voor elke $i = 1, 2, \ldots, n-k$.

Voorbeeld: Het rijtje

$$5, 8, 24, 13, 7, 18, 31, 19, 44, 63, 82, 29$$

is 4-gesorteerd (en overigens ook 6-gesorteerd).

Merk op: 1-gesorteerd = gesorteerd.

Shellsort gebruikt een rijtje stapgroottes (increments) $h_t, h_{t-1}, \ldots, h_2, h_1 = 1$. De rij A met n elementen wordt gesorteerd door achtereenvolgens subrijen te sorteren van elementen die telkens op afstand h_i van elkaar liggen. Met andere woorden: A wordt h_i-gesorteerd voor $i = t, \ldots, 1$. Aangezien $h_1 = 1$ sorteert Shellsort correct.

Merk op: bij het k-sorteren worden k deelrijtjes van elk maximaal $\lceil \frac{n}{k} \rceil$ elementen gesorteerd.

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

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

↓ 6-sorteren

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

↓ 3-sorteren

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

↓ 1-sorteren

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

Insertion sort op het voorbeeldrijtje: 52 vergelijkingen.

Shellsort met Insertion sort: 44 vergelijkingen:

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

\# inversies $= 41$ $\downarrow$ 6-sorteren: 8 vergelijkingen

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

\# inversies $= 25$ $\downarrow$ 3-sorteren: 15 vergelijkingen

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

\# inversies $= 11$ $\downarrow$ 1-sorteren: 21 vergelijkingen

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

```
(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 in je eigen deelrijtje
(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.

Een zeer belangrijke eigenschap van Shellsort is de volgende (zonder bewijs).

Stelling

Als een ℓ-gesorteerd array h-gesorteerd wordt met behulp van compare-exchanges, dan blijft het ℓ-gesorteerd.

Voorbeeld met $n = 12$ en incrementrijtje $6, 4, 3, 2, 1$:

$7, 19, 24, 13, 31, 8, 82, 18, 44, 63, 5, 29$	
$\downarrow$ 6-sorteren	
$7, 18, 24, 13, 5, 8, 82, 19, 44, 63, 31, 29$	6-gesorteerd
$\downarrow$ 4-sorteren	
$5, 8, 24, 13, 7, 18, 31, 19, 44, 63, 82, 29$	4-, 6-gesorteerd
$\downarrow$ 3-sorteren	
$5, 7, 18, 13, 8, 24, 31, 19, 29, 63, 82, 44$	3-, 4-, 6-gesorteerd
$\downarrow$ 2-sorteren	
$5, 7, 8, 13, 18, 19, 29, 24, 31, 44, 82, 63$	2-, 3-, 4-, 6-gesorteerd
$\downarrow$ 1-sorteren	
$5, 7, 8, 13, 18, 19, 24, 29, 31, 44, 63, 82$	1-, 2-, 3-, 4-, 6-gesorteerd

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 aantal rondes $t = \lfloor \lg n \rfloor$. Voor het gemak nemen we $n = 2^k$. Nu $t = \lg n = k$ en stapgroottes: $2^{k-1}, 2^{k-2}, \ldots, 4, 2, 1$.

   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. Zie verder college.

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

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

2. Stapgroottes (Hibbard): $2^k - 1, 2^{k-1} - 1, \ldots, 7, 3, 1$ ($k = \lfloor \lg n \rfloor$).

   Stelling. In de worst case doet Shellsort met Hibbard's increments $O(n^{\frac{3}{2}})$, ofwel $O(n\sqrt{n})$ vergelijkingen.

3. Het kan nog beter!

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. In de worst case doet Shellsort met Hibbards increments $O(n\sqrt{n})$ arrayvergelijkingen.

Bewijsschets: Je kunt inzien dat je bij stapgrootte h_i voor elk van de $n - h_i$ getallen maar met hoogstens $8h_i + 4$ getallen hoeft te vergelijken*: met $h_i = 3$, $h_{i+1} = 7$ en $h_{i+2} = 15$ weet je bijvoorbeeld al dat $A[100] \leq A[107] \leq A[114] \leq A[129]$. Dat is dus $O(nh_i)$ werk. We gebruiken dit voor $h_i < \sqrt{n}$. Voor $h_i > \sqrt{n}$ als voorheen: $O(n^2/h_i)$. Totaal geeft dat uiteindelijk $O(n\sqrt{n})$.

*Dit volgt uit de speciale eigenschappen van de h_i.

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

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

Betere incrementseries:

- 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 $O(n^{4/3})$

- $\cdots$ (optimale serie stapgroottes is [nog] onbepaald)

Opgave

Neem aan dat n een macht van 7 is, dus $n = 7^k$ met $k \geq 0$ geheel.
Bewijs nu dat het aantal vergelijkingen dat Shellsort met stapgroottes $n/7$, $n/49$, $n/343$, . . . , 49, 7, 1 in de worst case doet $\Omega(n^2)$ is.

Het bewijs gaat analoog aan het bewijs voor Shellsort met Shells rijtje stapgroottes (=increments).

Voor sorteeralgoritmen gebaseerd op arrayvergelijkingen is bewezen: het aantal arrayvergelijkingen* 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

*voor het vinden van de juiste ordening

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 34.)

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

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

*voor het vinden van de juiste positie

Er geldt zelfs:

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

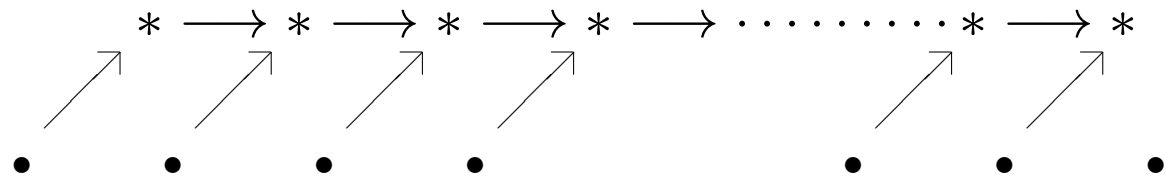
en dat is ook precies het aantal vergelijkingen dat Mergesort doet. Dus $\mathrm{B}(n) = \mathrm{M}(n)$.

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, 1959).

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:



Hierin betekent $*_1 \longrightarrow *_2$ dat $*_1 < *_2$ en $\bullet \longrightarrow *$ dat $\bullet < *$

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 listige volgorde**.

Het aantal vergelijkingen dat Merge Insertion sort doet is voor $n \leq 11$ en $n = 20, 21$ gelijk aan de theoretische ondergrens $\lceil \lg n! \rceil$ voor sorteren.

Merge Insertion sort sorteert bijvoorbeeld:

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

Voor kleine n (zoals $n = 12$) is het mogelijk om, via exhaustive computer search, de echte ondergrens *empirisch* te bepalen: probeer alle mogelijke combinaties van vergelijkingen op alle permutaties van n getallen en concludeer dat ℓ vergelijkingen niet voldoende zijn. De ondergrens is dan dus $\geq \ell + 1$.
Inmiddels is zo ook aangetoond dat Merge Insertion sort optimaal is voor $n = 13, 14, 15, 22$ (Peczarski, 2004/2006).

Vraag: is Merge Insertion sort optimaal?

Antwoord: nee, bijvoorbeeld voor $n = 47$ is een algoritme bekend (het Schulte Mönting algoritme) dat één vergelijking minder nodig heeft (200 om 201) dan Merge Insertion sort (Schulte Mönting, 1981).

Als je helemaal niets weet over het array A is de enige manier om A correct te ordenen het stellen van vragen van de vorm $A[i] < A[j]$. In dat geval geldt de theoretische ondergrens $\lceil \lg n! \rceil \in \Theta(n \lg n)$. Soms kun je echter sneller sorteren dan $\Theta(n \lg n)$ (namelijk in $O(n)$ tijd) door handig gebruik te maken van wat je al weet over de invoer.

Dit is niet in strijd met de theoretische ondergrens: die geldt immers voor sorteeralgoritmen die *alle* mogelijke inputs kunnen sorteren (en gebaseerd zijn op arrayvergelijkingen).

We bekijken ter afsluiting van het gedeelte over sorteren twee methoden die efficiënt sorteren als de invoer aan zekere voorwaarden voldoet: Counting sort en Radix sort.

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. Onder bovenstaande aanname over de invoer kan dat zonder (array)vergelijkingen in $O(n)$ stappen. 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 de waarde $A[j]$ 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 $\leq i$ uit A
for $j := n$ **downto** 1 **do** // !!
    $B[C[A[j]]] := A[j]$;
    $C[A[j]] := C[A[j]] - 1$;
od

$A = 3\ 6\ 4\ 1\ 3\ 4\ 1\ 4$ $n = 8$

$C = 0\ 0\ 0\ 0\ 0\ 0$ $k = 6$

$C = 2\ 0\ 2\ 3\ 0\ 1$ $\longleftarrow$ 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 $\Theta(n+k)$. Indien $k \in O(n)$ is de complexiteit dus $O(n)$.

Extra geheugenruimte is ook $\Theta(n+k)$.

Counting sort werkt voor invoerarrays waarvan de array-elementen begrensd zijn (tussen 1 en k liggen bijv.)

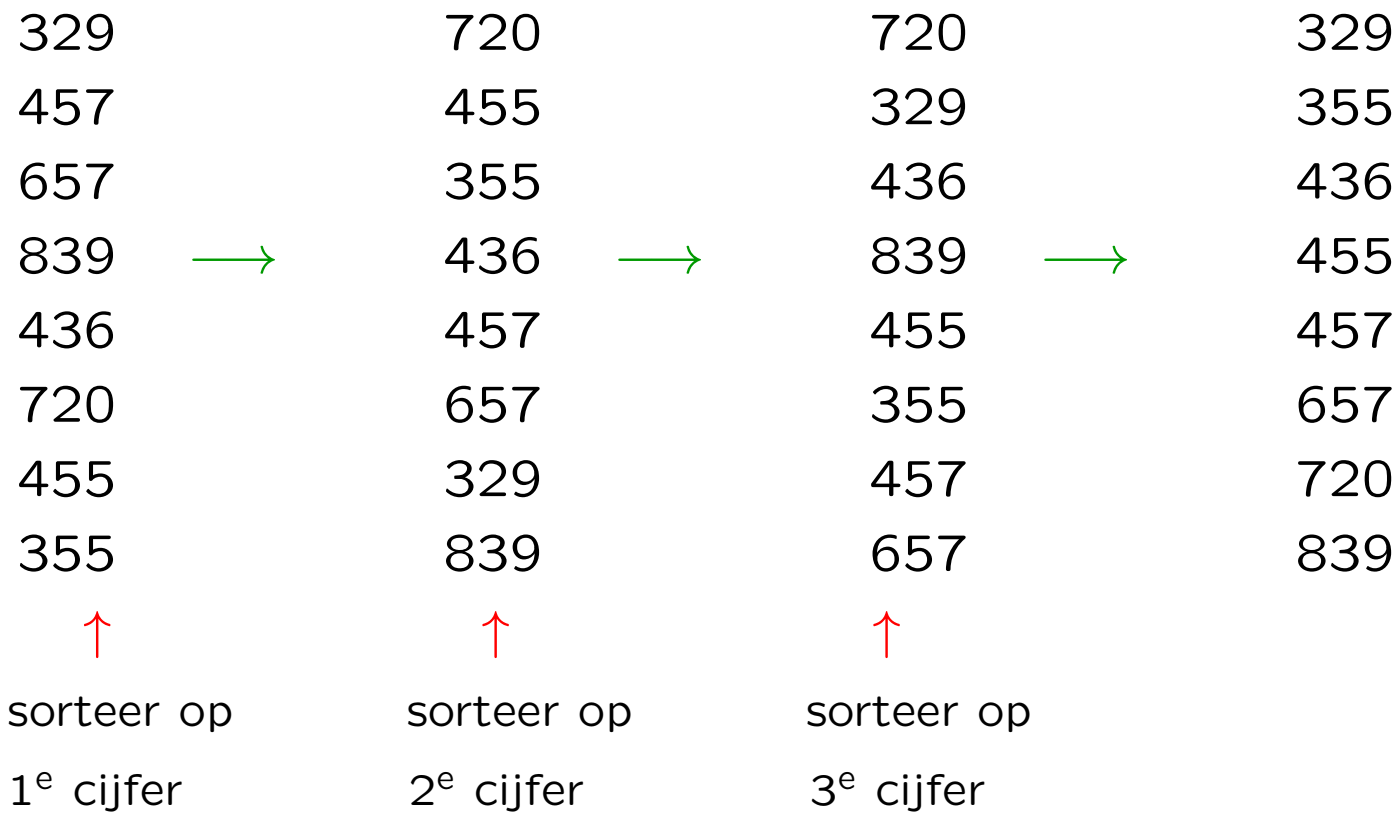
Counting sort is een stabiele sorteermethode, dat wil zeggen: gelijke waarden uit het invoerarray A komen in precies dezelfde volgorde in het uitvoerarray B te staan als ze in A stonden.

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.

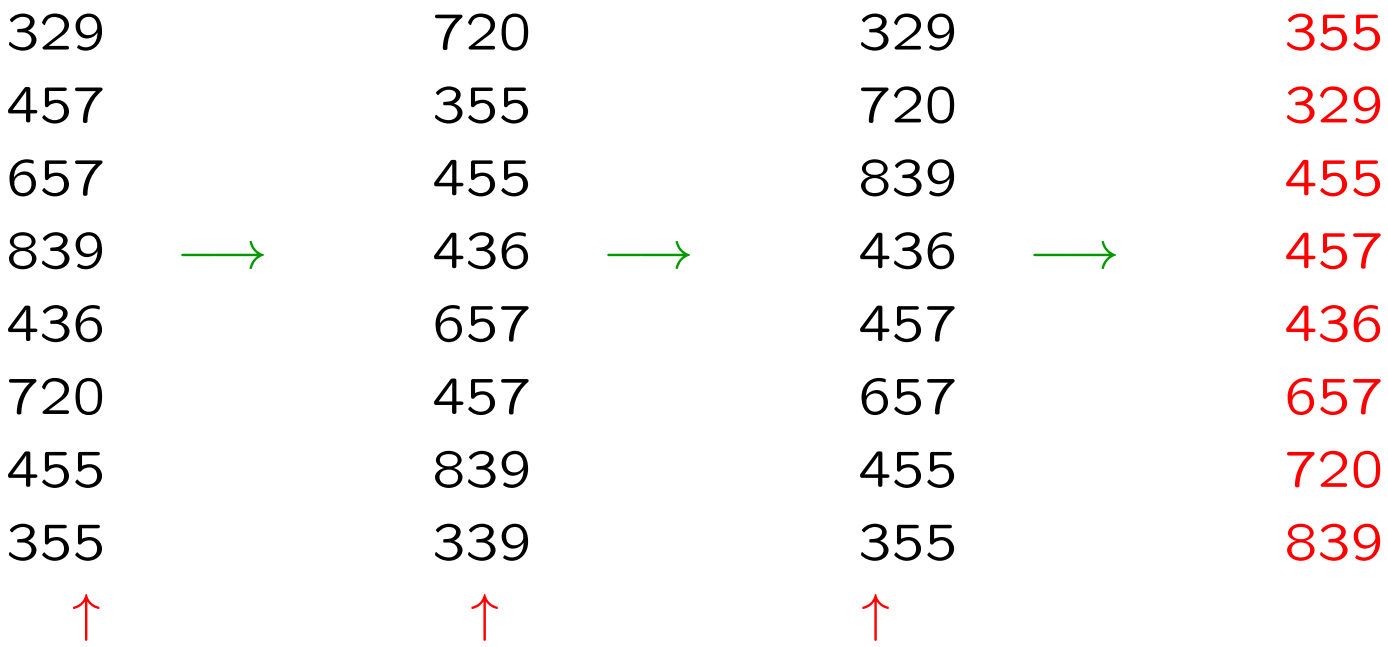
Radixsort(A, d):

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
436
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:



Gebruikte sorteermethode per cijfer is hier Mergesort

- Als sorteermethode per cijfer kunnen we Counting sort gebruiken, aangezien de cijfers tussen 0 en $k-1$ zitten.
- Bovendien is Counting sort *stabiel* (zie eerder).
- Als we Counting sort gebruiken kost elke ronde $\Theta(k+n)$ stappen. In totaal (d rondes) dus $\Theta(dk+dn)$. En dat is $O(n)$ als d een constante is en $k \in O(n)$.
- Een nadeel van deze methode is dat er net als bij Counting sort $\Theta(n+k)$ extra geheugenruimte nodig is.
- Radix sort kun je bijvoorbeeld ook gebruiken om woorden alfabetisch te sorteren (met $k = 26$ voor het Nederlandse alfabet).

- Volgende college:
  dinsdag 2 april, 11.00 – 12.45, zaal 174

- Eerstvolgende werkcollege:
  dinsdag 26 maart, 13.30 – 15.15, zaal 174
  Opgaven 23, 24, 26, 27, "extra"

- Tweede huiswerkopgave:

  * deadline: dinsdag 2 april; LaTeX; print → college

  * www.liacs.leidenuniv.nl/~graafjmde/COMP/