Complexiteit 2023 — college 6

14 maart 2023

Shellsort

Optimaal sorteren

Sorteren in $O(n)$

Vorige keer

- Mergesort
  - verdeel-en-heers, sorteert recursief twee helften en voegt ze weer samen
  - best case, worst case, average case $\in \Theta(n \lg n)$
- Quicksort
  - verdeel-en-heers, verdeelt in twee delen en sorteert deze recursief
  - best case, average case $\in \Theta(n \lg n)$, worst case $\in \Theta(n^2)$
- Worst case en average case sorteren (met arrayvergelijkingen) allebei $\in \Omega(n \lg n)$

Vandaag

- Shellsort
- Optimaal sorteren
- Sorteren in $O(n)$

Inversies

Voor sorteeralgoritmen gebaseerd op arrayvergelijkingen, waarbij per arrayvergelijking hooguit één inversie wordt opgeheven*, geldt:

- # arrayvergelijkingen ≥ # inversies invoerarray
- # arrayvergelijkingen in de worst case ≥ $\frac{1}{2}n(n-1)$

Als je een beter sorteeralgoritme wilt (gebaseerd op arrayvergelijkingen), moet je dus elementen verwisselen die verder van elkaar liggen, zoals gebeurt bij Mergesort, Quicksort en Shellsort (en veel meer).

*zoals bij algoritmen die gebruikmaken van buurverwisselingen, zoals Insertion sort en Bubblesort

Inversies opheffen

Stel dat $A[i]$ en $A[i+k]$ $(k > 0)$ verkeerd om staan en dat we die verwisselen. Hoeveel inversies worden dan ten minste respectievelijk ten hoogste opgeheven?

Situatie:

met $A[i] > A[i+k]$. Verwissel nu $A[i]$ en $A[i+k]$.

Shellsort*

Shellsort was een van de eerste sorteeralgoritmen met een beter dan kwadratische worst case-complexiteit.

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-operaties (compare-exchange, indien nodig).

*Donald Shell, 1959

k-gesorteerd

Definitie

Een rij $A[1], \ldots, A[n]$ 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 is gesorteerd.

De methode

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.

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

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

Voorbeeld

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

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

Algoritme

```
1  h := ⌊n/2⌋;                                          // dus h_t = ⌊n/2⌋
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
               // juist invoegen in deelrijtje
6              if temp < A[j − j] then
7                  A[j] := A[j − h];                    // schuif
8                  j := j − h;
9              else
10                 break;
11         A[j] := temp;                                // zet neer
       // de rij is nu h-gesorteerd
12     h := ⌊h/2⌋;          // oorspronkelijke keuze van Shell
```

Merk op: regels 4 t/m 11 is Insertion sort op deelrijtjes.

Eigenschap

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

Stelling
Als een ℓ-gesorteerd array wordt h-gesorteerd met behulp van vergelijk-verwissel-operaties, dan blijft het ℓ-gesorteerd.

Voorbeeld

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

Complexiteit

De complexiteit van Shellsort (d.w.z. het 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 $t = \lfloor \lg n \rfloor$ rondes. Voor het gemak nemen we $n = 2^k$. Dan: $t = \lg n = k$ en stapgroottes zijn $2^{k-1}, 2^{k-2}, \ldots, 4, 2, 1$.

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

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

Complexiteit

Stelling B
Het aantal arrayvergelijkingen dat Shellsort doet met deze serie stapgroottes 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 Hibbards serie stapgroottes $O(n\sqrt{n})$ vergelijkingen.

3. Het kan nog beter!

Bewijs Hibbard — Schets

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

Bewijsschets. In een ronde met *grote stap* h zijn we snel klaar met Insertion sort: $O(\frac{n^2}{h})$; in een ronde met *kleine stap* gebruiken we dat de rij al is voorgesorteerd en daarmee weinig vergelijkingen doet: $O(nh)$.

Als $n = 2^k$ dan geldt $\sqrt{n} = 2^{k/2}$, halverwege de stapgroottes.

Bewijs Hibbard — Frobenius

Feit. Als a en b geen gemene delers hebben, geldt: alle gehele getallen vanaf $m = (a-1)(b-1)$ kunnen worden uitgedrukt als combinatie $k \times a + \ell \times b$.*

$a = 3, b = 7$: $\quad 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, \ldots$

Gevolg voor Shellsort.
Als het array A zowel is a-gesorteerd als b-gesorteerd, dan geldt dat $A[j] \leq A[i]$ voor alle $j \leq i - m$. Kortom: als we $A[i]$ toevoegen met Insertion sort dan stopt dit proces vóór $A[i-m]$, omdat eerdere array-elementen kleiner zijn.

Dat zijn maximaal $\frac{m}{h}$ vergelijkingen (bij stapgrootte h).

*Frobenius coin problem

Bewijs Hibbard

Laat $n = 2^k$. Hibbard stappen $2^k - 1, 2^{k-1} - 1, \ldots, 7, 3, 1$.

Kleine stappen. Als de huidige stap h is, waren de vorige twee $2h + 1$ en $4h + 3$, met Frobeniuswaarde $m = 2h(4h + 2)$. Dan zijn er maximaal $n \times \frac{2h(4h+2)}{h} \approx 8nh$ vergelijkingen nodig bij een ronde met stap h.

$$\sum_{i=1}^{k/2} 8n(2^i - 1) \leq 8n \sum_{i=1}^{k/2} 2^i \leq 16n \times 2^{t/2} \in O(n\sqrt{n})$$

Grote stappen. Bij stap h maximaal $\frac{n^2}{h}$ vergelijkingen bij Insertion sort.

$$\sum_{i=k/2}^{k} \frac{n^2}{2^i - 1} \leq 2n^2 \sum_{i=t/2}^{t} \frac{1}{2^i} \leq 4n^2 \times \frac{1}{2^{t/2}} \in O(n\sqrt{n})$$

Samen. $O(n\sqrt{n})$

Relatief priem

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

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

Complexiteit

Betere incrementseries

- Twee (!) doorgangen: $h_2 \approx 1{,}72 \times \sqrt[3]{n}$, $h_1 = 1$
  Gemiddeld $O(n^{5/3})$
- Heel veel doorgangen: stapgroottes 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 $O(n(\lg n)^2)$
- Stapgroottes van de vorm $4^{j+1} + 3 \cdot 2^j + 1$:
  $1, 8, 23, 77, 281, 1073, 4193, 16577, \ldots$
  Worst case $O(n^{4/3})$
- . . . (optimale serie stapgroottes is [nog] onbepaald)

Oude tentamenopgave

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 $n/7, n/49, n/343, \ldots, 49, 7, 1$ in de worst case doet $\Omega(n^2)$ is.

Het bewijs gaat analoog aan het bewijs voor Shellsort met Shells rijtje stapgroottes.

Ondergrens

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

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

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

Binary Insertion sort

$M(n)$ is het worst case-aantal vergelijkingen van Mergesort voor een rij met n elementen.
$B(n)$ is het worst case-aantal vergelijkingen van Binary Insertion sort 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 te kunnen invoegen in het deelarray $A[1], \ldots, A[i-1]$ zijn nu in het slechtste geval $\lceil \lg i \rceil$ vergelijkingen nodig*. Dus:

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

*voor het vinden van de juiste positie

Demuth

Er geldt zelfs:

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

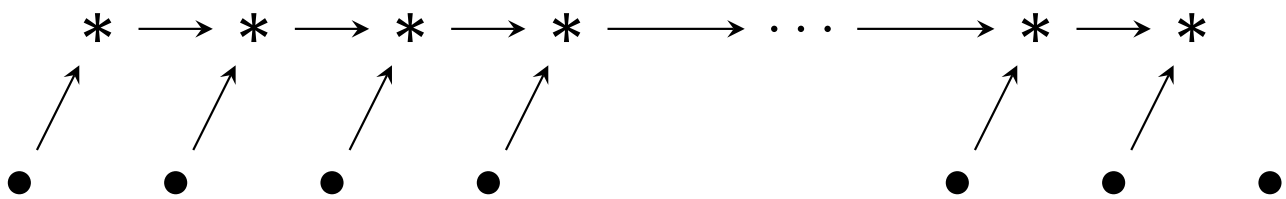
en dat is ook precies het aantal vergelijkingen dat Mergesort doet. Dus $B(n) = M(n)$.

Vraag: kan het nog beter?
Antwoord: ja!
Voorbeeld: 5 elementen kunnen worden gesorteerd met 7 vergelijkingen (Howard Demuth, 1956) (opgave 25b)

De methode van Demuth kan worden gegeneraliseerd tot het algoritme Merge Insertion sort (Lester Ford en Selmer Johnson, 1959).

Merge Insertion sort

1. vergelijk de n elementen twee aan twee
2. sorteer de $\lfloor \frac{n}{2} \rfloor$ winnaars $*$ (dus de grootsten) recursief. Dit levert iets op als:



Hierin betekent $*_1 \to *_2$ dat $*_1 < *_2$ en $\bullet \to *$ dat $\bullet < *$.

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

Aantal vergelijkingen

Het aantal vergelijkingen dat Merge Insertion sort doet is voor $n \leq 15$ en $n = 20, 21, 22$ gelijk aan de theoretische ondergrens van $\lceil \lg n! \rceil$.

Merge Insertion sort sorteert bijvoorbeeld:

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

Optimaal?

Voor kleine n (zoals $n = 12$) is het mogelijk om, via exhaustive search, de echte ondergrens *empirisch* te bepalen: probeer alle mogelijke combinaties van vergelijken 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 (Schulte Mönting, 1981) dat één vergelijking minder nodig heeft (200) dan Merge Insertion sort (201).

Voorkennis

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 voorkennis over de invoer.

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

Counting sort

Invoer: een array $A = A[1], \ldots, A[n]$
Aanname: $\forall i : 1 \leq A[i] \leq k$ voor een of andere constante 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 worden gebruikt om X meteen op de juiste positie te zetten in het uitvoerarray. Onder bovenstaande aanname over de invoer kan dat zonder (array)vergelijkingen in $O(n)$ stappen. Pas dit idee aan voor de situatie waarin waarden vaker kunnen voorkomen in A.

Het algoritme

```
1  for i := 1 to k do
2  |  C[i] := 0;                                  // initialisatie
3  od
4  for j := 1 to n do
5  |  C[A[j]] := C[A[j]] + 1;
   |  // telt hoe vaak de waarde A[j] voorkomt in A
6  od
7  for i := 2 to k do
8  |  C[i] := C[i] + C[i − 1];
9  od
   // C[i] bevat nu het aantal getallen ≤ i uit A
10 for j := n down to 1 do                        // !!
11 |  B[C[A[j]]] := A[j];
12 |  C[A[j]] := C[A[j]] − 1;
13 od
```

Voorbeeld

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

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

$C = 2\ 0\ 2\ 3\ 0\ 1$ $\qquad \leftarrow$ na tweede for

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

$C = 2\ 2\ 4\ 7\ 7\ 8$ $\qquad \leftarrow$ na derde for

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

De complexiteit

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

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

Counting sort werkt voor invoerrijtjes waarvan de elementen zijn begrensd (bijvoorbeeld tussen 1 en k).

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

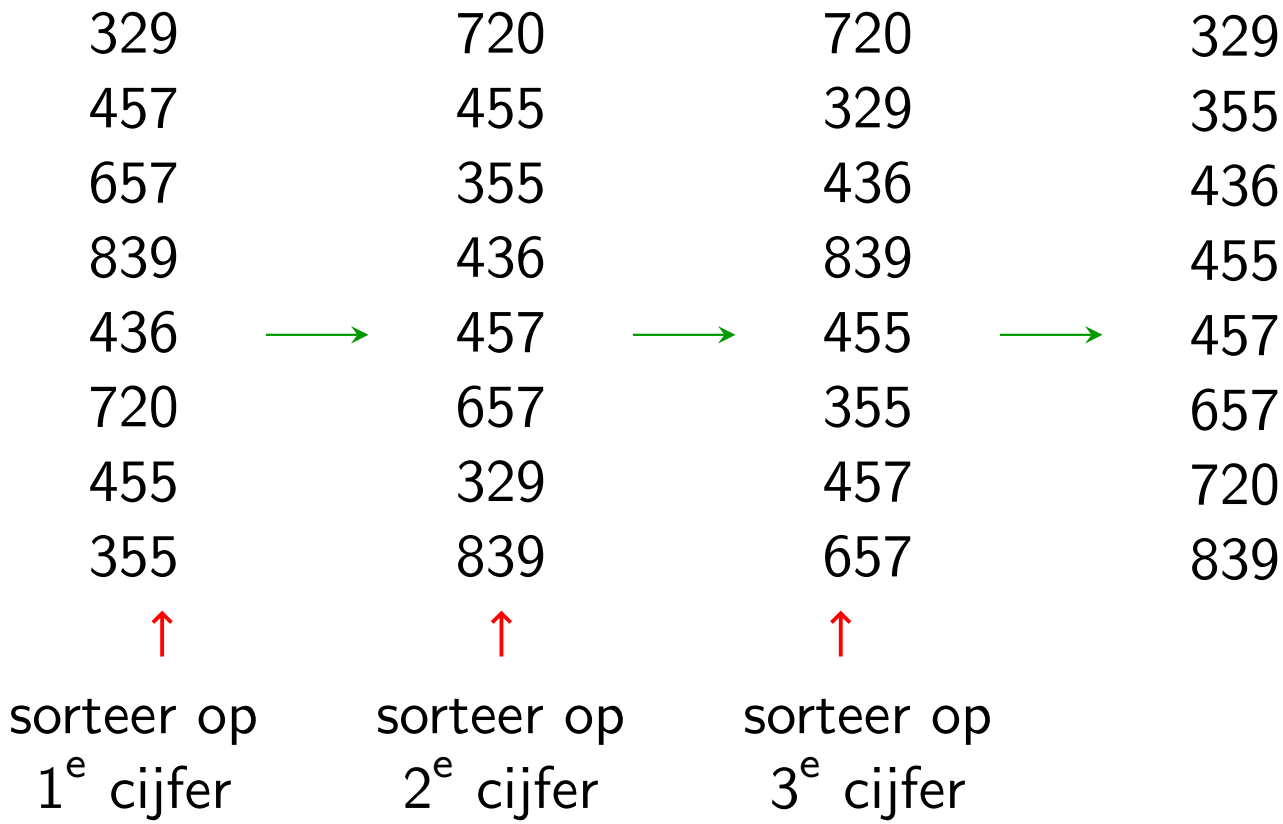
Radix sort

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 $k = 10$).

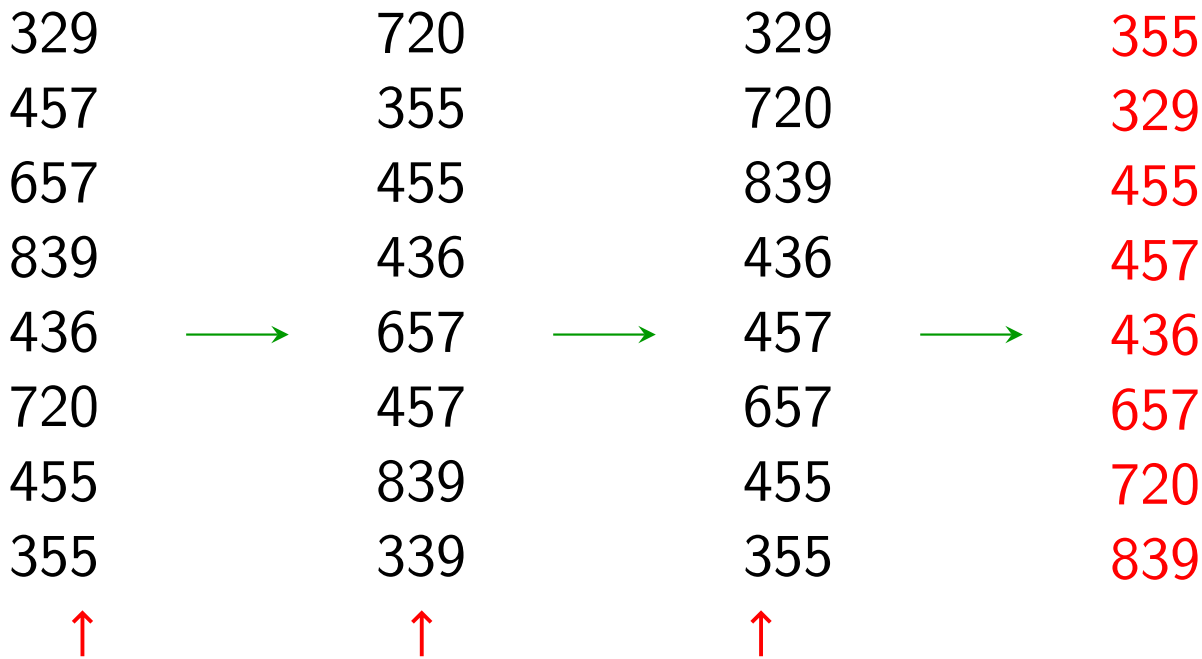
De getallen worden gesorteerd door ze achtereenvolgens op het i^{de} cijfer te sorteren, te beginnen bij het minst significante cijfer. Het gebruik van een stabiele sorteermethode voor het sorteren op het i^{de} cijfer is essentieel.

```
1 Radixsort(A, d)::
2 |   for i := 1 to d do
  |   |   // minst significante cijfer eerst, dus van
  |   |      rechts naar links
3 |   |   sorteer A op het i^de cijfer;
  |   |   // met een stabiele (!!) sorteermethode
4 |   od
```

Voorbeeld


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

Niet stabiel



Gebruikte sorteermethode is hier Mergesort.

Complexiteit

- 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).

Om te onthouden van vandaag

- Shellsort
  - sorteert deelrijtjes in rondes a.d.h.v. een serie (afnemende) stapgroottes: k-sorteren
  - complexiteit sterk afhankelijk van serie stapgroottes; Shell-rijtje $O(n^2)$ en Hibbard $O(n\sqrt{n})$
- Binary Insertion sort
- Merge Insertion sort: bereikt de theoretische ondergrens voor een aantal kleine n, maar nog steeds niet optimaal
- Counting sort: tellen en in de juiste volgorde overschrijven; $\Theta(n + k)$, of $O(n)$ als $k \in O(n)$.
- Stabiel sorteren: behoudt volgorde gelijke elementen
- Radix sort: sorteert cijfer voor cijfer, van minst significant (rechts) naar meest significant (links); stabiel intern sorteeralgoritme nodig; $\Theta(dk + dn)$, of $O(n)$ als d constant en $k \in O(n)$.

(Werk)college

Volgende college: dinsdag 21 maart, 9u00–10u45, zaal C3 (Gorlaeus)

Werkcollege: zodadelijk van 11u00 tot 12u45, computerzalen 302–304 en 303 (dus niet 306–308) (Snellius)
Opgaven uit het dictaat: 39 (minus Heapsort), 40, 41, 42, 43

Huiswerk

Huiswerk 1

Is nagekeken, cijfers en feedback zijn beschikbaar op Brightspace.
Het (einde van het) werkcollege is een goed moment voor vragen.