Vijfde college complexiteit

5 maart 2019

Selectie is $O(n)$

Insertion sort

Mergesort

Probleem

Gegeven n verschillende getallen, opgeslagen in een array A: $A[1], A[2], \ldots, A[n]$. Laat verder een geheel getal k met $1 \leq k \leq n$ gegeven zijn. Gevraagd de $A[i]$ die groter is dan precies $k-1$ andere $A[j]$'s. M.a.w.: we zoeken het k-de element in grootte (in volgorde van klein naar groot).

Complexiteit

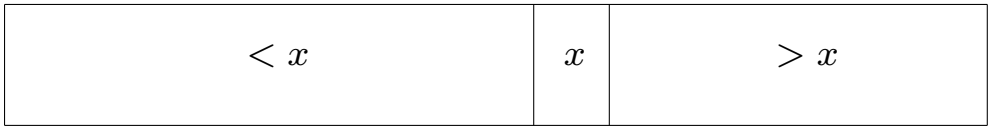
Het selectieprobleem is $O(n)$.

We bewijzen dit door een algoritme te geven dat de k-de in grootte vindt in $O(n)$ vergelijkingen in de worst case*.

*Blum, Floyd, Pratt, Rivest & Tarjan, 1973

Idee:

- Kies (bijvoorbeeld random) een element x uit het array (de pivot)
- Reorganiseer het array als volgt:



- Als het array n elementen bevat kost dit ongeveer n arrayvergelijkingen
- De k-de in grootte zit in het linker óf het rechter gedeelte
- Ga dus met één van beide gedeeltes verder en doe daar weer hetzelfde
- Als je geluk hebt splits je telkens in 2 gelijke stukken en doe je in totaal zo'n $n + \frac{n}{2} + \frac{n}{2^2} + \frac{n}{2^3} + \cdots \frac{n}{2^m} \leq 2n$ vergelijkingen
- Het verwachte aantal vergelijkingen is $O(n)$ (average case), maar de worst case is $\Theta(n^2)$
- Oplossing: kies goede pivots

Algoritme:

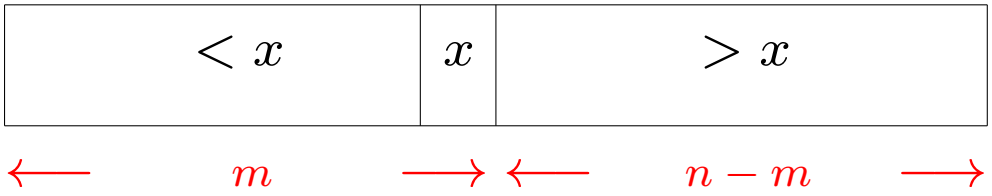
1. Verdeel de getallen in $\lfloor \frac{n}{5} \rfloor$ groepjes van 5 elementen, en 1 groepje met de resterende n mod 5.*

2. Vind de mediaan van elk van de $\lceil \frac{n}{5} \rceil$ groepjes van 5 (of minder), bijvoorbeeld met behulp van Bubblesort of opgave 24.

3. Vind de mediaan x van de in stap 2 gevonden $\lceil \frac{n}{5} \rceil$ medianen: recursie.

*deze 5 is niet zomaar een random gekozen waarde, maar is optimaal

4. Partitioneer (ongeveer zoals bij Quicksort) alle elementen rond x. Stel dat m getallen $\leq x$ zijn en $n-m$ getallen $> x$.



5. Vind de k-de in grootte uit m stuks als $k \leq m$, of de $(k-m)$-de uit $n-m$ als $k > m$: recursie.

De mediaan van ℓ elementen is de $\lceil \frac{\ell}{2} \rceil$-de in grootte.

Na stap 3. van het algoritme geldt:

- Ten minste $3 * (\lceil \frac{1}{2} \lceil \frac{n}{5} \rceil \rceil - 2) \geq \frac{3n}{10} - 6$ elementen zijn groter (respectievelijk kleiner) dan x.
- Dus er zijn hooguit $\frac{7n}{10} + 6$ elementen $\leq x$ (respectievelijk $\geq x$) (*).

Gevolg: in stap 5. wordt het algoritme derhalve recursief aangeroepen op hooguit $\frac{7n}{10} + 6$ elementen (*).

(*) om preciezer te zijn: $\lceil \frac{7n}{10} \rceil + 6$

Zie opgave 30 voor een voorbeeld.

12	15	11	2	9	5	0	7	3	21	44	40	1	18	20	32	19	35	37	39
13	16	14	8	10	16	6	33	4	27	49	46	52	25	51	34	43	56	72	79
17	23	24	28	29	30	31	36	42	47	50	55	58	60	63	65	66	67	81	83
22	45	38	53	61	41	62	82	54	48	59	57	71	78	64	80	70	76	85	87
96	95	94	86	89	69	68	97	73	92	74	88	99	84	75	90	77	93	98	91

Blauw is de mediaan der medianen; groen is gegarandeerd lager, evenals het linker oranje gedeelte; rood en het rechter oranje stuk zijn gegarandeerd hoger. In het algemeen zijn gegarandeerd ongeveer $\frac{3n}{10}$ elementen uit het array kleiner dan de mediaan der medianen en gegarandeerd ongeveer $\frac{3n}{10}$ elementen groter. In het ergste geval ga je de recursie dus in op de resterende 70% van het array.

Laat $T(n)$ = aantal vergelijkingen in de worst case dat dit (recursieve) algoritme doet.
Onder de aanname dat $T(n)$ stijgend is geldt:

$$T(n) \leq \begin{cases} \texttt{const} & n \leq \ldots \\ T(\lceil \frac{n}{5} \rceil) + T(\lceil \frac{7n}{10} \rceil + 6) + O(n) & n > \ldots \end{cases}$$

$< n$ $\leq dn$ $\uparrow$

als $n > 20$ bijv. 80

De term $O(n)$ komt van stap 1, 2 en 4 samen.

Bewering:

$T(n) \leq cn$ voor geschikte c en $n \geq \cdots$

ofwel: $T(n) \in O(n)$

Opgave 26:

Elk algoritme voor het selectieprobleem dat is gebaseerd op arrayvergelijkingen doet in de worst case ten minste $\lceil \frac{n}{2} \rceil$ van zulke vergelijkingen.

Gevolg:

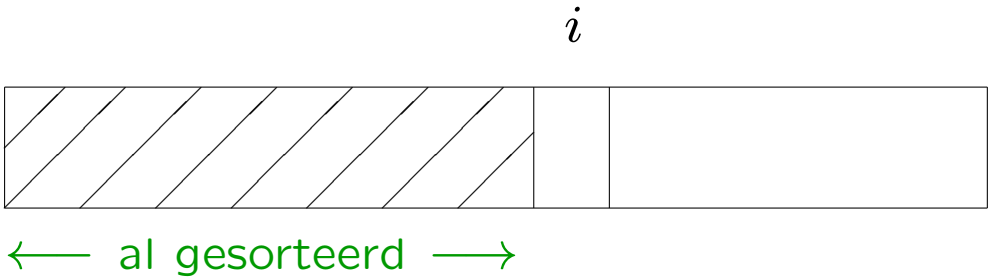
Het selectieprobleem is $\Omega(n)$.

Probleem

Gegeven een rij (array) A met n elementen $A[1], \ldots, A[n]$. Sorteer A oplopend, dus $A[i] \leq A[i+1]$ voor alle i ($<$ als alle $A[i]$ verschillend zijn).

We bekijken eerst sorteeralgoritmen die gebaseerd zijn op het doen van arrayvergelijkingen

Insertion sort



Conceptueel idee Insertion sort: itereer over i en voeg $A[i]$ op de juiste plek in het gesorteerde stuk $A[1] \cdots A[i-1]$ in door herhaald met de linkerbuur te vergelijken en te verwisselen indien nodig.

Het algoritme is gebaseerd op het doen van arrayvergelijkingen ($A[i] < A[j]$).

(1) **for** $i := 2$ **to** n **do**

// nu $A[i]$ op de juiste plek in $A[1] \ldots A[i-1]$ 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**

- Het aantal arrayvergelijkingen is een goede maat voor de complexiteit.

- Insertion sort doet eigenlijk steeds compare-exchange operaties: vergelijk en verwissel (indien nodig). Deze zijn hier vermomd als verschuivingen, waarna pas in de laatste stap $A[i]$ daadwerkelijk wordt neergezet.

- De verwisselingen zijn steeds buurverwisselingen.

We tellen het aantal vergelijkingen $A[j] > x$ $(= A[i])$.

1. Worst case: $W(n) = \sum_{i=2}^{n}(i-1) = \frac{1}{2}n(n-1)$
2. Best case: $B(n) = \sum_{i=2}^{n} 1 = n-1$
3. Average case (*): $A(n) = \frac{1}{4}n(n-1) + n - \sum_{i=1}^{n} \frac{1}{i} \in \Theta(n^2)$

(*) onder de aanname dat alle $A[i]$'s verschillend zijn en dat alle $n!$ permutaties (ordeningen) van $A[1]$ t/m $A[n]$ even waarschijnlijk zijn. We middelen dan over alle mogelijke permutaties en dat zijn in essentie alle mogelijke invoerrijtjes.

Definitie: een inversie van de permutatie $A[1], A[2], \ldots, A[n]$ is een paar $(A[i], A[j])$ waarvoor $i < j$ en $A[i] > A[j]$. M.a.w.: een inversie is een paar $(A[i], A[j])$ dat verkeerd om staat.

Merk op: *elk* sorteeralgoritme moet *alle* aanwezige inversies opheffen.

Verder: als een sorteeralgoritme altijd hooguit één inversie opheft per arrayvergelijking, dan is het aantal vergelijkingen dat wordt gedaan om $A[1], \ldots, A[n]$ te sorteren *ten minste* het aantal inversies van A.

Bovendien: een buurverwisseling (zoals bij Insertion sort) heft altijd precies één inversie op (indien de buurelementen verkeerd om staan).

Stelling

Het maximale aantal inversies dat kan voorkomen in een rijtje van n verschillende waarden is $\binom{n}{2} = \frac{1}{2}n(n-1)$. Dit treedt op bij een omgekeerd (= aflopend) gesorteerd rijtje.

Gevolg

Elk sorteeralgoritme (gebaseerd op arrayvergelijkingen) dat hooguit één inversie opheft per vergelijking doet ten minste $\frac{1}{2}n(n-1)$ vergelijkingen in de worst case.

Conclusie: Insertion sort is optimaal voor wat betreft de worst case, binnen de klasse van algoritmen gebaseerd op het doen van arrayvergelijkingen, waarbij per vergelijking hooguit één inversie wordt opgeheven (bijvoorbeeld via buurverwisselingen).

Stelling

Het *gemiddeld* aantal inversies in een permutatie van n verschillende waarden (bijvoorbeeld de getallen 1 t/m n) is $\frac{1}{4}n(n-1)$. Dit onder de aanname dat alle $n!$ permutaties even waarschijnlijk zijn.

Gevolg

Elk algoritme dat sorteert met behulp van arrayvergelijkingen en dat per vergelijking ten hoogste één inversie opheft, moet ten minste $\frac{1}{4}n(n-1)$ vergelijkingen doen in de average case.

Conclusie: Insertion sort doet gemiddeld $\frac{1}{4}n(n-1)+n-1-\sum_{i=2}^{n}\frac{1}{i}$ vergelijkingen, **dus** Insertion sort is in de average case in orde van grootte optimaal (binnen de betreffende klasse van algoritmen), namelijk $\Theta(n^2)$.

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

- # arrayvergelijkingen $\geq$ # inversies invoerarray

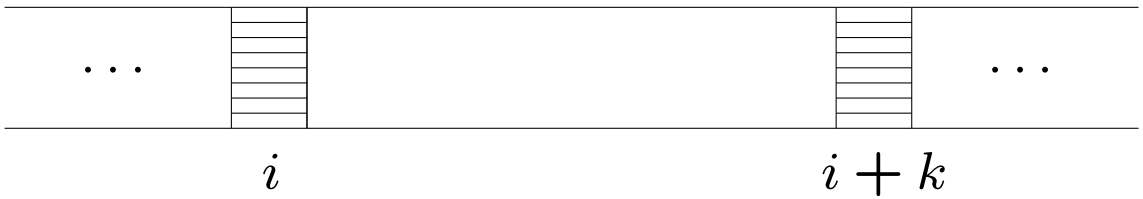
- # arrayvergelijkingen in de worst case $\geq \frac{1}{2}n(n-1)$

Als je een beter sorteeralgoritme (gebaseerd is op arrayvergelijkingen) wilt, moet je elementen verwisselen die verder van elkaar liggen, zoals Mergesort, Quicksort, Shellsort.

*dit is het geval bij algoritmen die gebruikmaken van buurverwisselingen, zoals Insertion sort en Bubblesort

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 $A[i+k]$.

Mergesort en Quicksort zijn sorteermethoden die allebei gebaseerd zijn op de verdeel-en-heers strategie:

```
Sorteer(rij)::
    if ( de rij heeft meer dan één element ) then
        Verdeel de rij in twee stukken: linkerrij en rechterrij;
        Sorteer(linkerrij);
        Sorteer(rechterrij);
        Combineer linkerrij en rechterrij;
    fi .
```

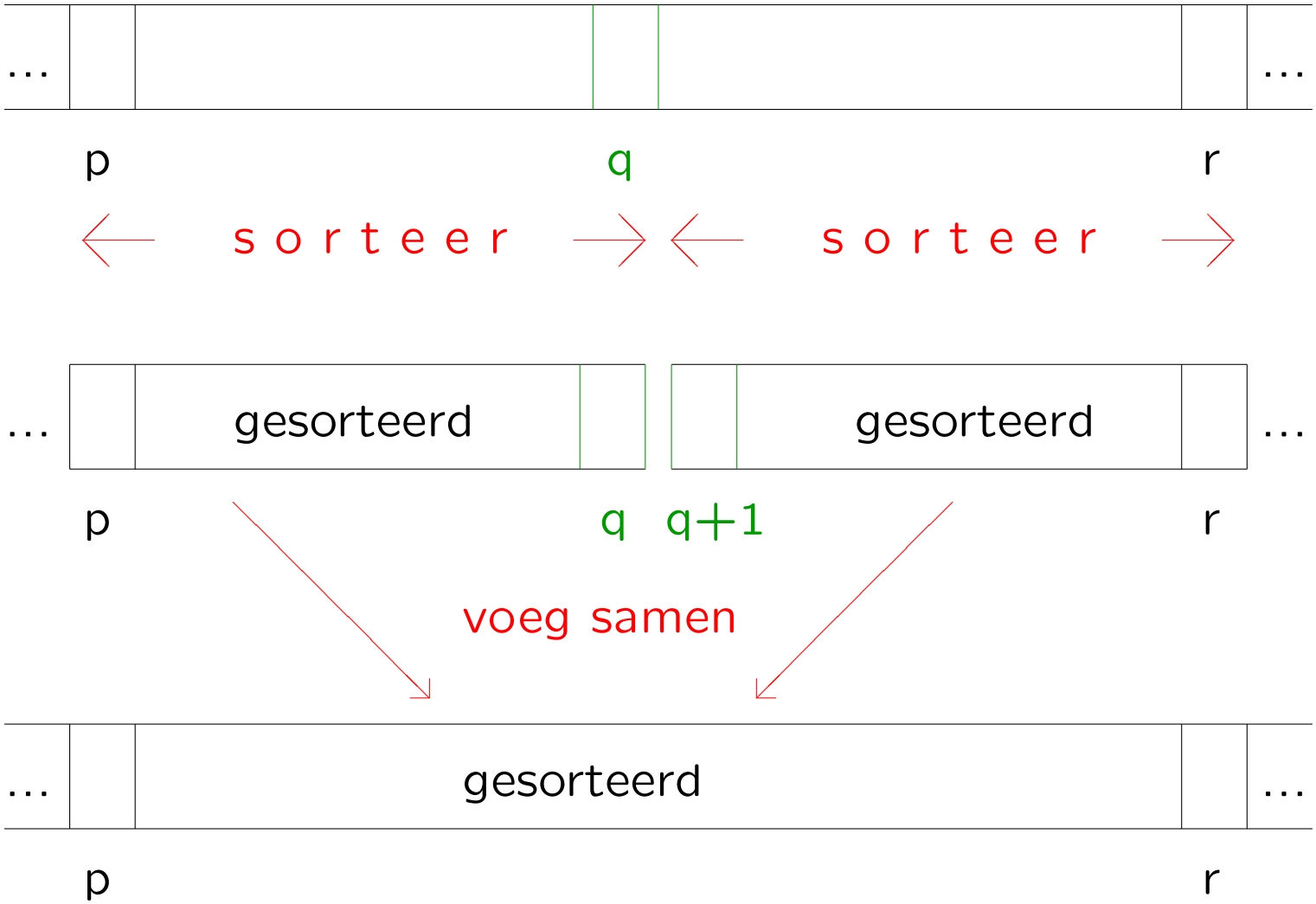
Mergesort stopt het meeste werk in de Combineer-stap, Quicksort in de Verdeel-stap. Beide sorteermethoden zijn gebaseerd op arrayvergelijkingen, maar doen geen buurverwisselingen, zoals Insertion sort en Bubblesort.

Het (recursieve) Mergesort algoritme:

MergeSort(A, p, r)::
// sorteert $A[p], \ldots, A[r]$

if $p < r$ **then**
$q := \lfloor \frac{p+r}{2} \rfloor$;
MergeSort(A, p, q); verdeel
MergeSort$(A, q+1, r)$; en
Merge(A, p, q, r); heers (voeg samen)
fi

Aanroep: MergeSort$(A, 1, n)$.


...
...
p
q
r
sorteer
sorteer
...
gesorteerd
gesorteerd
...
p
q q+1
r
voeg samen
...
gesorteerd
...
p
r

Merge(A, p, q, r)::

$i := p$; $j := q + 1$; $k := p$;
while $i \leq q$ **and** $j \leq r$ **do**
  if $A[i] < A[j]$ **then**
    `hulp`$[k] := A[i]$; $i := i + 1$; $k := k + 1$;
  else
    `hulp`$[k] := A[j]$; $j := j + 1$; $k := k + 1$;
  fi
od
if $i > q$ **then** // eerste helft is op
  kopieer $A[j], \ldots, A[r]$ naar `hulp`;
else // tweede helft is op
  kopieer $A[i], \ldots, A[q]$ naar `hulp`;
fi
kopieer `hulp`$[p], \ldots,$`hulp`$[r]$ terug naar A;

- Merge(A,p,q,r) voegt de reeds gesorteerde deelrijtjes $A[p], \ldots, A[q]$ en $A[q+1], \ldots, A[r]$ samen tot een gesorteerd stuk $A[p], \ldots, A[r]$

- `hulp` is een hulparray ter grootte n (net als A)

- Geheel analoog kan een functie Merge(A, B, C, k, m) geschreven worden die twee gesorteerde rijen A (k elementen) en B (m elementen) samenvoegt tot de gesorteerde rij C ($n = k + m$ elementen)

- Voor het bepalen van de complexiteit van Merge tellen we het aantal vergelijkingen van de vorm: $A[i] < A[j]$

- Er worden altijd $2n$ verplaatsingen van array-elementen gedaan

- Is het aantal arrayvergelijkingen hier wel een goede maat voor de complexiteit?

Stel dat we met behulp van Merge twee gesorteerde rijtjes van respectievelijk k en m elementen (met $k+m = n$) samenvoegen tot één gesorteerde rij. Dan geldt:

1. Het aantal vergelijkingen in de worst case is $n - 1$

2. Het aantal vergelijkingen in de best case is $\min\{k, m\}$

Let op: *binnen Mergesort* is het aantal vergelijkingen een goede maat voor de complexiteit. Immers het aantal vergelijkingen is in dat geval altijd $\Theta(n)$, evenals het aantal verplaatsingen van array-elementen. In het algemene geval is dit niet zo (bijvoorbeeld $k = 1$ en $m = n - 1$).

Zij $T(n) =$ aantal vergelijkingen in de worst case van Mergesort op n elementen, met $n = 2^k$.

Dan geldt:

$$T(n) = \begin{cases} 0 & n = 1 \\ 2T(\frac{n}{2}) + n - 1 & n = 2^k > 1 \end{cases}$$

Oplossing: $T(n) = n \lg n - n + 1 \in \Theta(n \lg n)$

Als n geen tweemacht is, wordt de recurrente betrekking:

$$T(n) = \begin{cases} 0 & n = 1 \\ T(\lfloor \frac{n}{2} \rfloor) + T(\lceil \frac{n}{2} \rceil) + n - 1 & n > 1 \end{cases}$$

Dan geldt eveneens: $T(n) \in \Theta(n \lg n)$.
Je kunt zelfs bewijzen: $T(n) = n\lceil \lg n \rceil - 2^{\lceil \lg n \rceil} + 1$

Mergesort is in orde van grootte optimaal voor wat betreft de worst case (immers: de ondergrens voor sorteren via arrayvergelijkingen is $\Omega(n \lg n)$). Er is echter extra geheugenruimte ter grootte $\Theta(n)$ nodig.

Zij $B(n) =$ aantal vergelijkingen in de best case, met $n = 2^k$.
Dan geldt:

$$B(n) = \begin{cases} 0 & n = 1 \\ 2B(\frac{n}{2}) + \frac{n}{2} & n = 2^k > 1 \end{cases}$$

Oplossing: $B(n) = \frac{n}{2} \lg n \in \Theta(n \lg n)$.

Stelling

1. *Elk* algoritme gebaseerd op arrayvergelijkingen dat twee gesorteerde arrays (rijen) van lengte m samenvoegt tot één gesorteerd array, doet in het slechtste geval ten minste $2m-1$ van zulke vergelijkingen.
   Voor $m = \frac{n}{2}$ (n even) is dit dus ten minste $n-1$.

2. Voor het samenvoegen van twee rijtjes ter lengte $m-1$ respectievelijk m is dat ten minste $2m-2$.
   Voor $m = \lceil \frac{n}{2} \rceil$ (n oneven) is dit ten minste $n-1$.

Gevolg. Binnen de klasse van samenvoegalgoritmen gebaseerd op arrayvergelijkingen is het beschreven Merge-algoritme optimaal, althans voor twee ongeveer even lange rijtjes.

We geven een klasse van invoerrijtjes waarop *elk* samenvoegalgoritme (gebaseerd op arrayvergelijkingen) *ten minste* $2m-1$ vergelijkingen moet doen. Dat bewijst dan de stelling.

Kies stijgende rijtjes $A = (a_1, a_2, \ldots, a_m)$ en $B = (b_1, b_2, \ldots, b_m)$ zó dat alle a_i en b_j verschillend zijn en $a_i < b_j \leftrightarrow i < j$:

$$b_1 < a_1 < b_2 < \cdots < a_{i-1} < b_i < a_i < b_{i+1} < \cdots < b_m < a_m$$

Dan *moet* elk samenvoegalgoritme a_i met b_i vergelijken ($i = 1, 2, \ldots, m$) en a_i met b_{i+1} ($i = 1, 2, \ldots, m-1$).

Het bewijs van deze bewering gaat uit het ongerijmde.

- Volgende college:
  dinsdag 19 maart, 11.00 – 12.45, zaal 174

- Werkcollege:
  dinsdag 5 maart, 13.30 – 15.15, zaal 174
  Opgaven 20, 22, 18, 14, 15

- Huiswerkopgave 1:

  * deadline: vrijdag 8 maart; LaTeX, print → kamer 159

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