Complexiteit 2023 — college 5

7 maart 2023

Mergesort

Quicksort

Ondergrens sorteren

Vorige keer

- selectie kan in $O(n)$
- Insertion sort
  - best case: $n - 1$
  - worst case: $\frac{1}{2}n(n-1)$, optimaal voor buurvergelijkingen
  - average case: $\Theta(n^2)$, optimaal in orde van grootte voor buurvergelijkingen
- inversie: paar $(A[i], A[j])$ zodat $i < j$ maar $A[i] > A[j]$, sorteeralgoritmen moeten alle inversies opheffen
- recurrente betrekkingen
  - vorm een vermoeden (herhaalde substitutie of doorrekenen eerste x termen)
  - bewijs met volledige inductie
  - oefenen

Vandaag

- Recurrente betrekkingen (nog even)
- Mergesort
- Quicksort
- Ondergrens sorteren

Recurrente betrekkingen

- het kan soms helpen om termen los te laten staan (om een sommatie te zien)
- volledige inductie:
  - *zwak*: neem aan dat het geldt voor $n - 1$*
  - *sterk*: neem aan dat het geldt voor *alle* voorgaande n†

Voorbeeld: opgave 14b

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

---

*of voor $\frac{n}{2}$ of ...

†zodat $n = 2^k, k \geq 1$ of ...

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 $A[i+k]$.

Verdeel en heers

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

```
1 Sorteer(rij)::
2 |   if de rij heeft meer dan één element then
3 |   |   Verdeel de rij in twee stukken: linkerrij en rechterrij;
4 |   |   Sorteer(linkerrij);
5 |   |   Sorteer(rechterrij);
6 |   |   Combineer linkerrij en rechterrij;
7 |   fi
```

Mergesort stopt het meeste werk in de combineerstap, Quicksort in de verdeelstap. Beide sorteermethoden zijn gebaseerd op arrayvergelijkingen, maar doen niet uitsluitend buurverwisselingen, zoals Insertion sort en Bubblesort.

Mergesort*

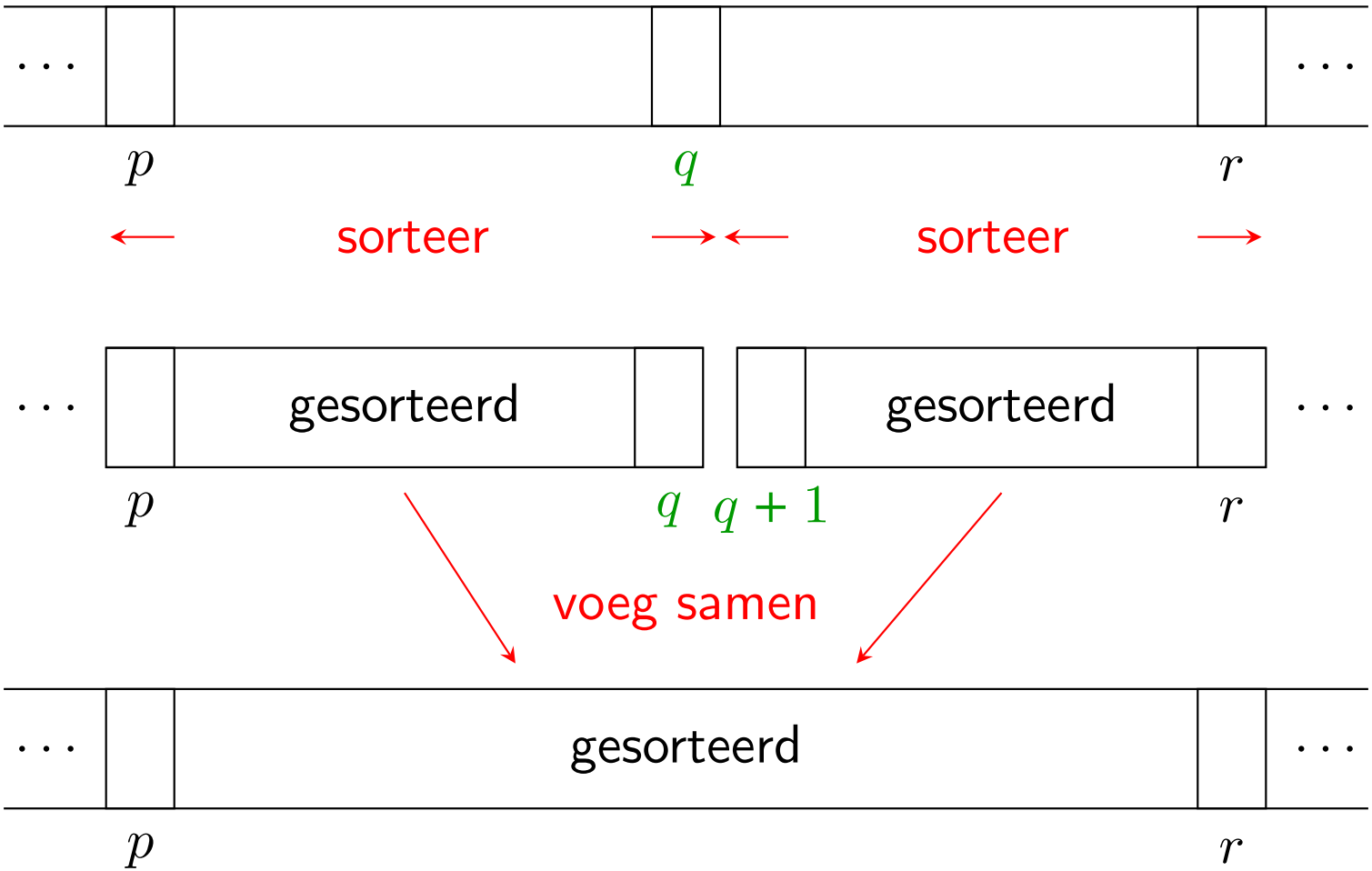
Het (recursieve) Mergesort-algoritme:

```
1 Mergesort(A, p, r)::
  | // sorteert A[p], ..., A[r]
2 | if p < r then
3 | | q := ⌊(p+r)/2⌋;
4 | | Mergesort(A, p, q);                    // verdeel
5 | | Mergesort(A, q + 1, r);                    // en
6 | | Merge(A, p, q, r);          // heers (voeg samen)
7 | fi
```

Aanroep: Mergesort$(A, 1, n)$

*John von Neumann, 1945; Wikipedia: known for [waslijst] + 93 more

Mergesort


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

Samenvoegen

```
1  Merge(A, p, q, r)::
2      i := p; j := q + 1; k := p;
3      while i ≤ q and j ≤ r do
4          if A[i] < A[j] then
5              hulp[k] := A[i]; i := i + 1; k := k + 1;
6          else
7              hulp[k] := A[j]; j := j + 1; k := k + 1;
8          fi
9      od
10     if i > q then                          // eerste helft is op
11         kopieer A[j], ..., A[r] naar hulp;
12     else                                   // tweede helft is op
13         kopieer A[i], ..., A[q] naar hulp;
14     fi
15     kopieer hulp[p], ..., hulp[r] terug naar A;
```

Merge

- 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) worden geschreven die twee gesorteerde rijen A (k elementen) en B (m elementen) samenvoegt tot de gesorteerde rij C ($n = k + m$ elementen)

Merge

- voor het bepalen van de complexiteit 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?

Complexiteit Merge

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. Het aantal vergelijkingen is in dat geval altijd $\Theta(n)$, net als het aantal verplaatsingen van array-elementen. In het algemene geval is dit niet zo (bijvoorbeeld $k = 1$ en $m = n - 1$).

Worst case Mergesort

Zij $T(n)$ het aantal vergelijkingen van Mergesort in de worst case 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: $T(n) = n \lg n - n + 1 \in \Theta(n \lg n)$

Worst case Mergesort

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 de worst case (immers: de ondergrens voor sorteren met arrayvergelijkingen is $\Omega(n \lg n)$). Er is echter extra geheugenruimte nodig ter grootte $\Theta(n)$.

Best case Mergeworst

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

Optimaliteit Merge

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 van 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.

Bewijs

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 $a_i < b_j \Leftrightarrow i < j$:

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

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

Het bewijs van deze bewering gaat uit het ongerijmde.

Sorteren (ondergrens)

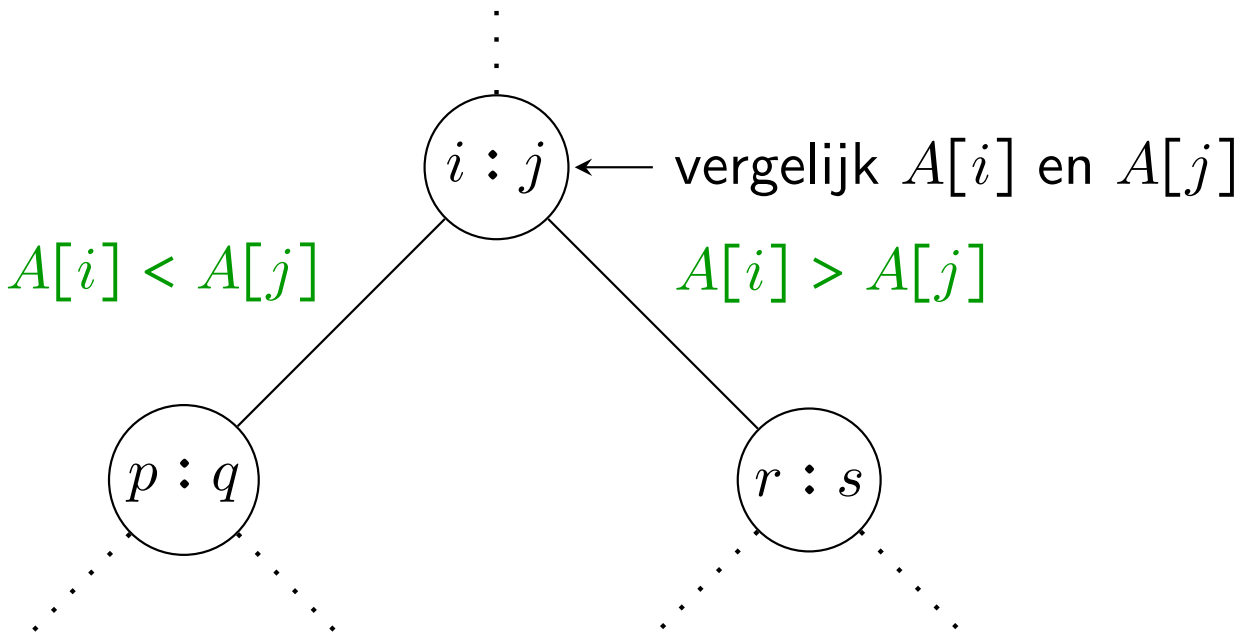
We bekijken sorteeralgoritmen gebaseerd op het doen van vergelijkingen van de vorm $A[i] < A[j]$.

Aannames (z.v.v.a.):

- A bevat n verschillende waarden. (We bepalen immers een ondergrens voor de worst case.)
- het sorteeralgoritme stopt zodra de sortering (onderlinge ordening) is gevonden.

Zo'n algoritme komt overeen (voor elke n) met een beslissingsboom die de series vergelijkingen representeert die het algoritme uitvoert voor elke mogelijke invoer (ter grootte n). Elk pad van de wortel tot een blad komt overeen met een executie van het algoritme.

Beslissingsboom

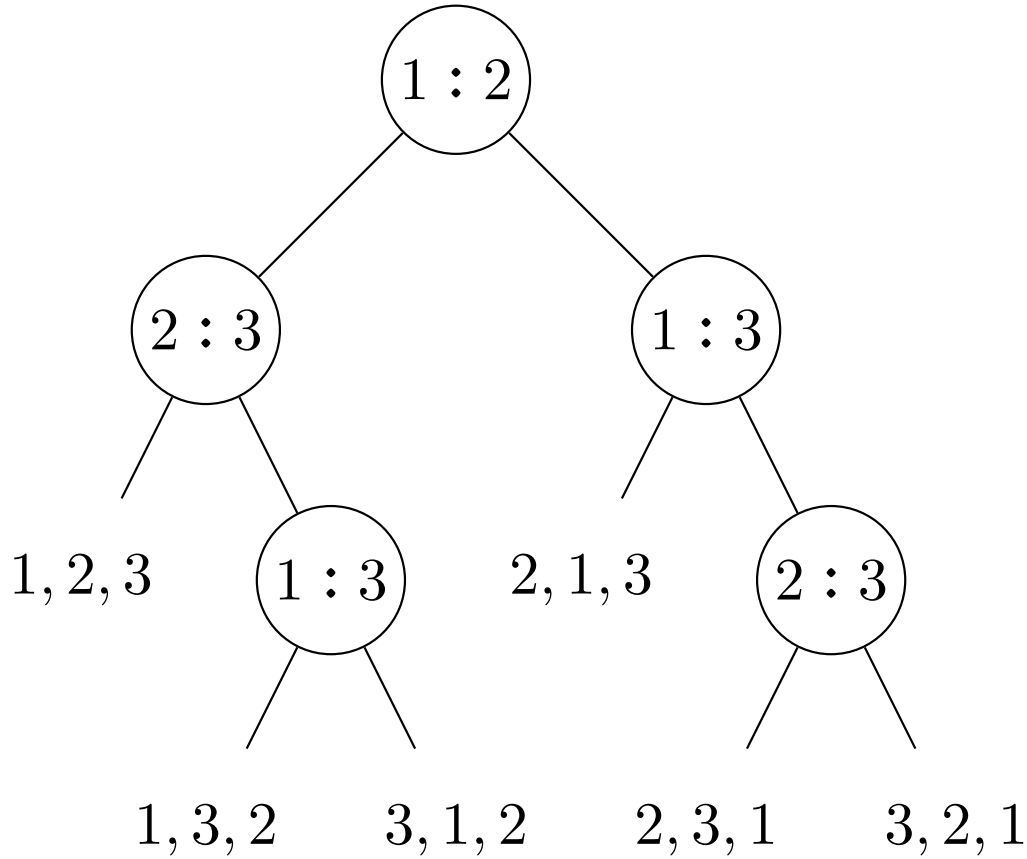


vergelijk vervolgens
$A[p]$ en $A[q]$

vergelijk vervolgens
$A[r]$ met $A[s]$

Beslissingsboom voor algoritmen gebaseerd op **arrayvergelijkingen**

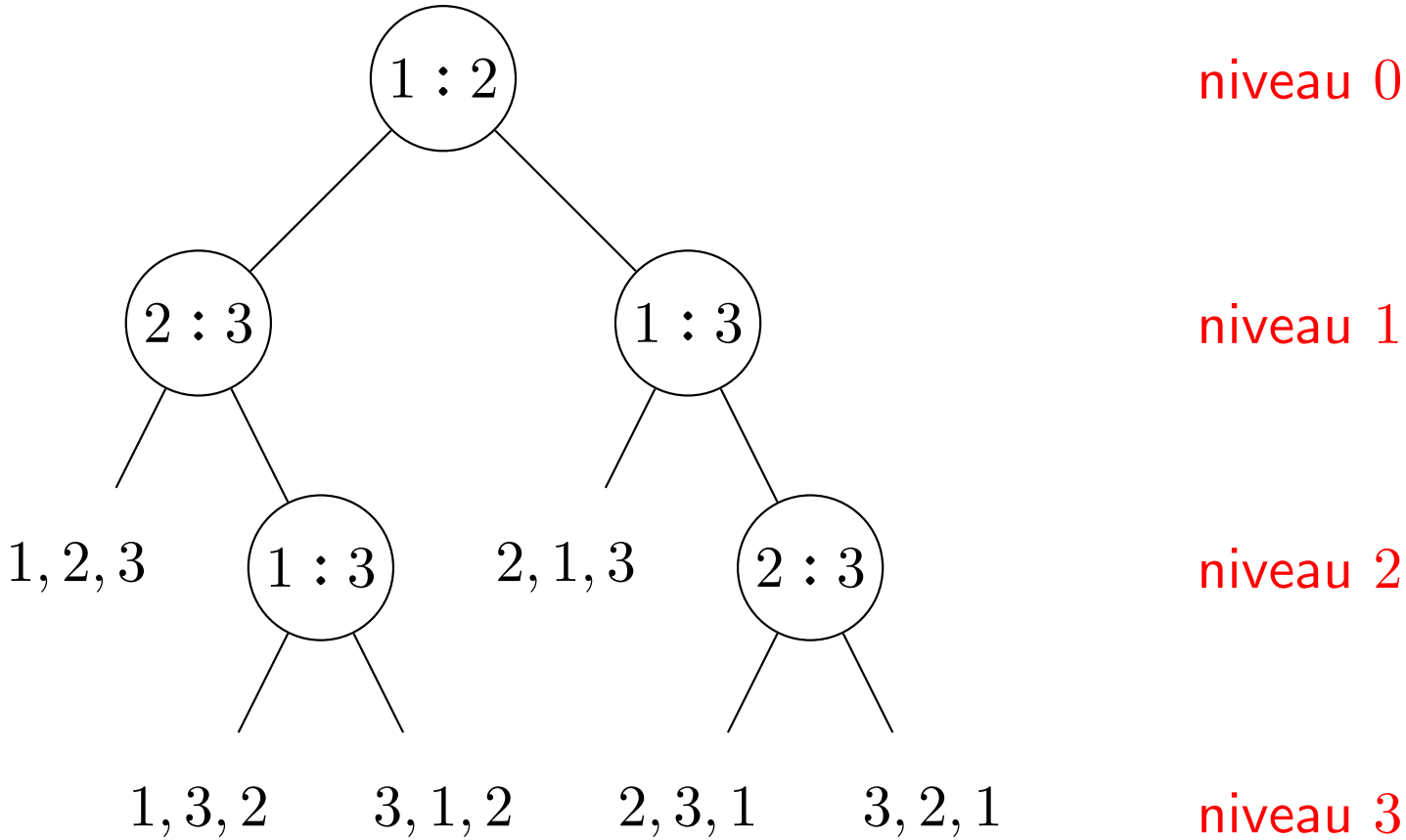
Insertion sort



Beslissingsboom voor Insertion sort met $n = 3$

$2, 3, 1$ betekent: $A[2] < A[3] < A[1]$ (andere bladeren analoog)

Hoogte $\Longleftrightarrow$ worst case



In een beslissingsboom voor algoritmen gebaseerd op arrayvergelijkingen geeft de hoogte van de boom precies het aantal vergelijkingen aan in de worst case.

Bladeren*: maximaal

1.
   - alleen de onderlinge volgorde van de array-elementen wordt onderscheiden; niet de waarde
   - het rijtje $6, 11, 15, 8, 3$ wordt bijvoorbeeld precies zo behandeld door het sorteeralgoritme als het rijtje $2, 4, 5, 3, 1$
   - ze volgen dan ook precies hetzelfde pad in de beslissingsboom
   - er zijn in essentie $n!$ mogelijke te onderscheiden invoeren, die elk één pad volgen in de boom $\Rightarrow$ er zijn maximaal $n!$ bladeren

*algemeen

Bladeren*: minimaal

2.
   - sorteren komt neer op het vinden van de oplopende ordening
   - er zijn dus $n!$ verschillende eindantwoorden (ordeningen) mogelijk
   - een sorteeralgoritme moet die allemaal kunnen vinden
   - de bijbehorende beslissingsboom moet dus minstens $n!$ bladeren hebben
3. conclusie: een beslissingsboom corresponderend met een sorteeralgoritme gebaseerd op arrayvergelijkingen, heeft precies $n!$ bladeren (waar n het aantal elementen is).

*voor sorteren

Ondergrens sorteren

Stelling[*]
Het aantal vergelijkingen in de worst case is voor elk algoritme dat sorteert middels arrayvergelijkingen ten minste $\lceil \lg b \rceil$ (dus $\Omega(n \lg n)$).

Stelling[†]
Het aantal vergelijkingen in de average case is voor elk algoritme dat sorteert middels arrayvergelijkingen $\Omega(n \lg n)$. Dit onder de aanname dat alle $n!$ mogelijke volgordes als invoerrijtje even waarschijnlijk zijn.
De stelling volgt direct uit de resultaten op de volgende sheet.

---

[*] om dit te bewijzen heb je alleen nodig dat het aantal bladeren $\geq n!$ is

[†] hiervoor gebruik je dat het aantal bladeren precies gelijk is aan $n!$

Intermezzo

Gegeven een binaire boom B met b bladeren.

Definitie De externe padlengte E van B is de som van de lengtes van alle paden van de wortel naar een blad:

$$E = \sum_{\text{bladeren}} (\text{lengte pad wortel} \rightarrow \text{blad})$$

Lemma Zij E de externe padlengte van B. Dan geldt:

$$E \geq b \times (\lceil \lg b \rceil - 1)$$

Gevolg De gemiddelde lengte van een pad van de wortel naar een blad is $\frac{E}{b} \geq \lceil \lg b \rceil - 1$.

Verdeel en heers

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

```
1 Sorteer(rij)::
2 |   if de rij heeft meer dan één element then
3 |   |   Verdeel de rij in twee stukken: linkerrij en rechterrij;
4 |   |   Sorteer(linkerrij);
5 |   |   Sorteer(rechterrij);
6 |   |   Combineer linkerrij en rechterrij;
7 |   fi
```

Mergesort stopt het meeste werk in de combineerstap, Quicksort in de verdeelstap. Beide sorteermethoden zijn gebaseerd op arrayvergelijkingen, maar doen niet uitsluitend buurverwisselingen, zoals Insertion sort en Bubblesort.

Quicksort*

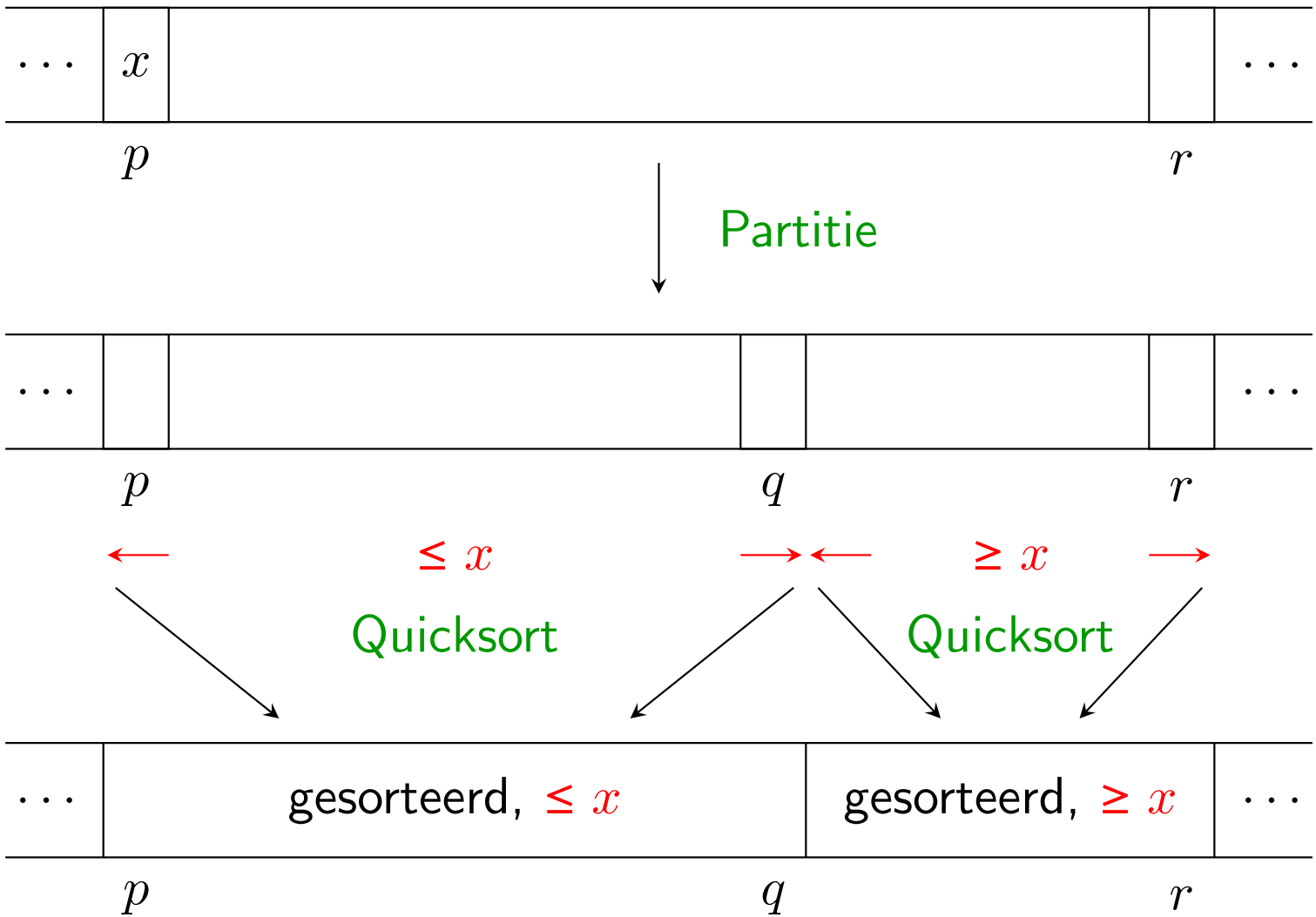
```
1 Quicksort(A, p, r)::
  |  // sorteert A[p], ..., A[r]
2 |  if p < r then
3 |  |  q := Partitie(A, p, r);
4 |  |  Quicksort(A, p, q);
5 |  |  Quicksort(A, q + 1, r);
6 |  fi
```

Aanroep: Quicksort(A, 1, n)

- recursief
- alleen interne verwisselingen
- geen extra geheugenruimte
- in de praktijk een van de snelste

*Tony Hoare, 1962; Hoare-logica, CSP (en veel meer); Turing Award 1980

Quicksort


$\cdots$
x
$\cdots$
p
r
Partitie
$\cdots$
$\cdots$
p
q
r
$\leq x$
$\geq x$
Quicksort
Quicksort
$\cdots$
gesorteerd, $\leq x$
gesorteerd, $\geq x$
$\cdots$
p
q
r

Partitie

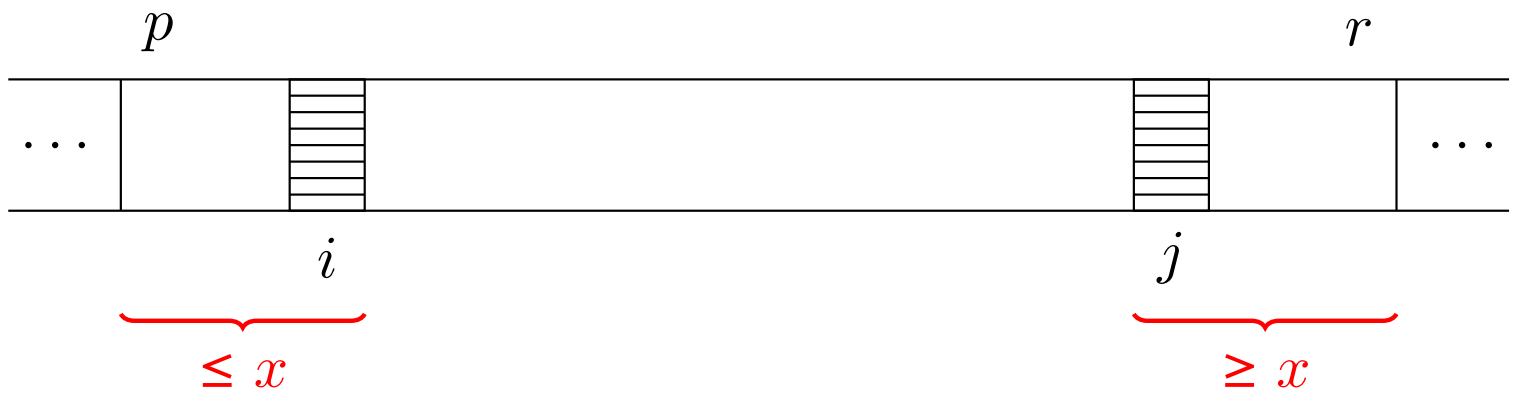
```
1  Partitie(A, p, r)::
       // (*) hier komt nog wat
2      x := A[p]; i := p − 1; j := r + 1;
3      while i < j do
4          j := j − 1;                    // loop met j naar links
5          while A[j] > x do
6              j := j − 1;
7          od                  // tot je een waarde A[j] ≤ x vindt
8          i := i + 1;                    // loop met i naar rechts
9          while A[i] < x do
10             i := i + 1;
11         od                  // tot je een waarde A[i] ≥ x vindt
12         if i < j then
13             wissel(A[i], A[j]);
14         fi  // A[i] en A[j] staan nu in het goede stuk
15     od
16     return j;                                  // dit wordt dus q
```

Opmerkingen bij Partitie

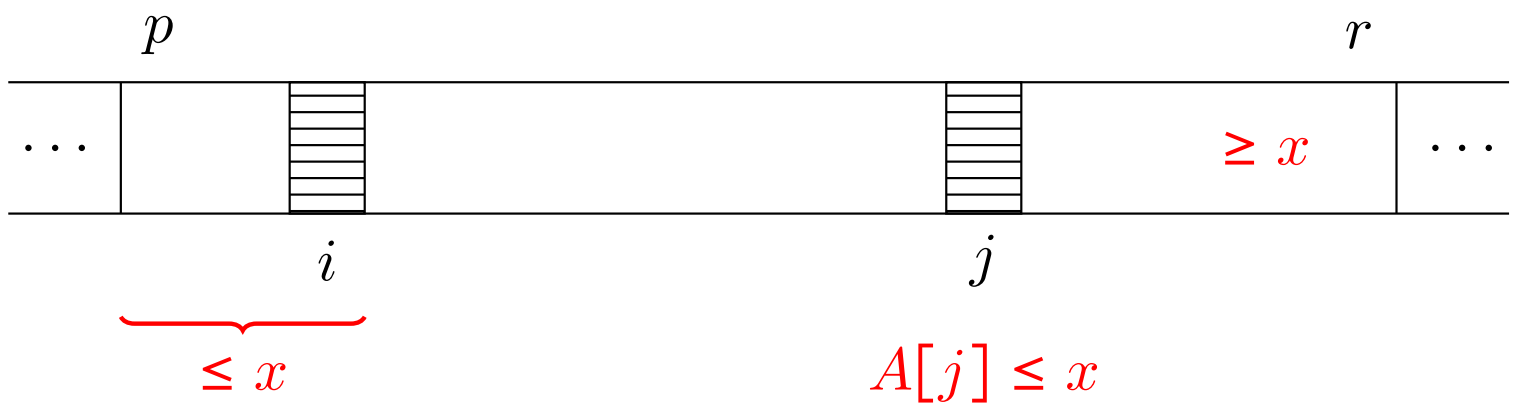
1. in ronde 1 stopt i op positie p en j op een positie $\geq p$
2. de *basisoperatie* is het vergelijken van array-elementen: $A[j] > x$ en $A[i] < x$
3. partitie stopt uiteindelijk met $i = j$ of $i = j + 1$
4. na afloop is altijd $p \leq j \leq r - 1$, dus $p \leq q \leq r - 1$. Quicksort wordt dus echt recursief aangeroepen op kleinere rijtjes.
5. elk array-element wordt precies één keer vergeleken met x, behalve $A[q]$ (twee keer) en eventueel $A[q + 1]$ (soms twee)
6. Partitie doet altijd $\Theta(m)$ vergelijken, namelijk $m + 1$ of $m + 2$, met m het aantal elementen van het (deel)array $A[p], \ldots, A[r]$
7. er worden elementen verwisseld die ver uit elkaar kunnen liggen. Per vergelijking worden dus wellicht > 1 inversies opgeheven.

Werking Partitie

1. Na een volledige ronde:

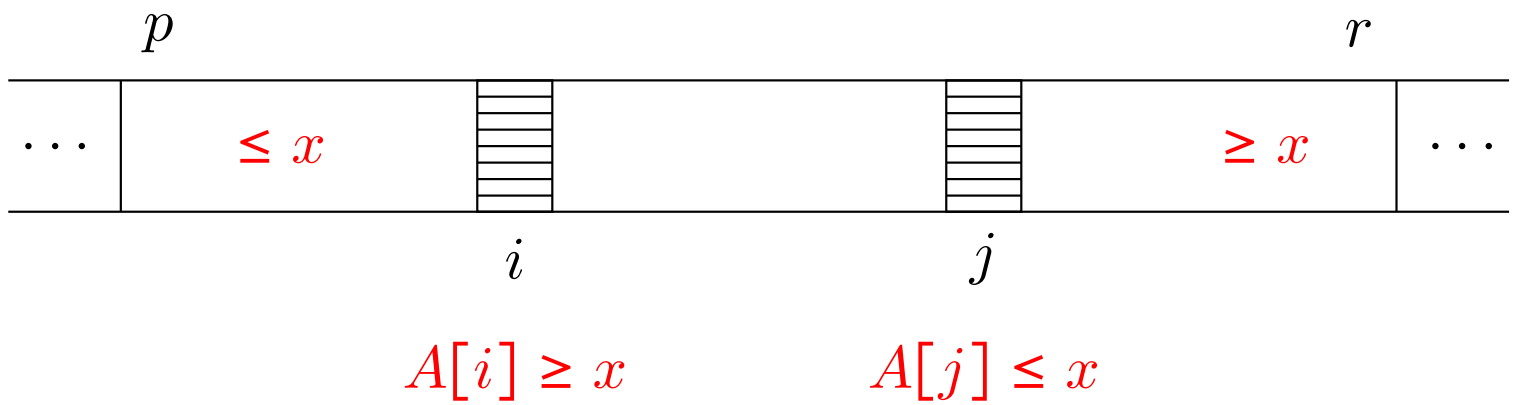


2. Na de j-loop:

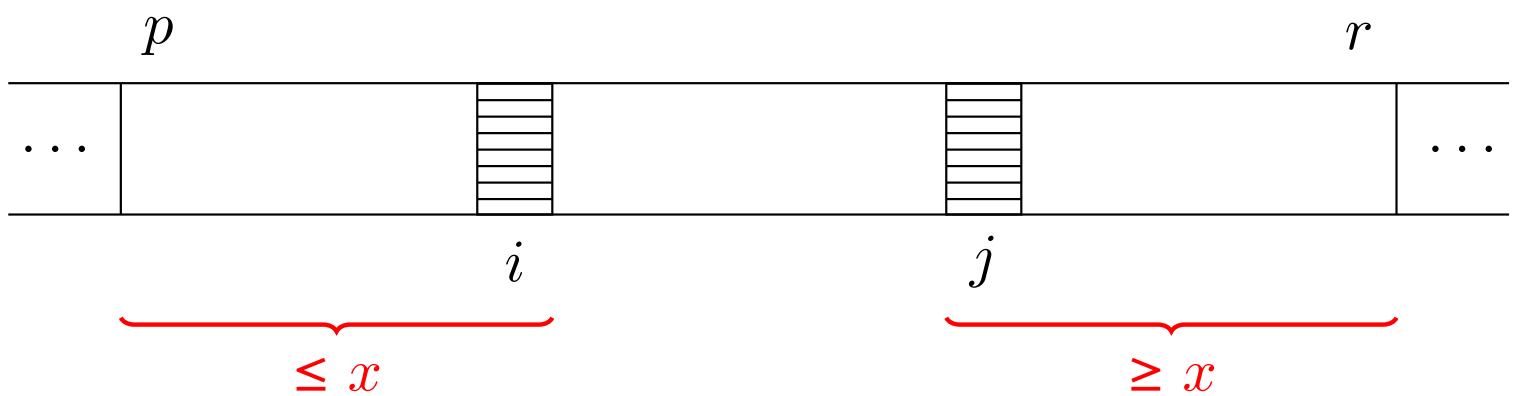


Werking Partitie

3. Na de i-loop, vóór de verwisseling:



4. Na de verwisseling:



Vragen bij Partitie

- wat gebeurt er bij gelijke array-elementen? Bijvoorbeeld: $5, 3, 5, 5, 6, 5, 7$ of $5, 6, 3, 4, 3, 8, 3$ of een array bestaande uit allemaal dezelfde elementen.
- wat gebeurt er met een rijtje met allemaal verschillende elementen? Bijvoorbeeld: $3, 2, 7, 4, 6, 8, 5, 1$, of een reeds gesorteerd rijtje of een omgekeerd gesorteerd rijtje.
- zie ook opgaven 40 en 41 uit het dictaat.

Bad case

Bad case voor Quicksort:

Op een reeds gesorteerd rijtje (!) bestaande uit verschillende waardes, bijvoorbeeld $1, 2, \ldots, n$, doet Quicksort

$$\sum_{k=3}^{n+1} k = \frac{1}{2} n(n+3) - 2 \in \Theta(n^2)$$

vergelijkingen. Partitie deelt het array telkens zeer onevenwichtig in tweeën.

Good case

Good case voor Quicksort (met $n = 2^k$):

$$B(n) = \begin{cases} 0 & n = 1 \\ 3 & n = 2 \\ 2B(\frac{n}{2}) + \Theta(n) & n = 2^k, n > 2 \end{cases}$$

Het aantal vergelijkingen $B(n)$ is dan $\Theta(n \lg n)$. Dit komt voor als Partitie het array bij elke aanroep precies in tweeën deelt. Dit is zelfs het *beste* geval voor Quicksort.

Worst case

Laat $W(n)$ het aantal vergelijkingen voorstellen dat Quicksort doet in de worst case. Dan voldoet W aan*:

$$W(n) = \begin{cases} 0 & n = 1 \\ \max_{1 \leq q \leq n-1}(W(q) + W(n-q)) + \Theta(n) & n > 1 \end{cases}$$

Er geldt dus dat $W(n) \leq dn^2$ voor zekere $d > 0$, en vanaf zekere n_0. Dus: $W(n) \in O(n^2)$.

Samen met de bad case hebben we dus:

Stelling
Quicksort doet $\Theta(n^2)$ vergelijkingen in de worst case.

*i.p.v. $\Theta(n)$ kun je in de recurrente betrekking ook $\leq n + 2$ zetten

Spil (pivot)

De keuze van de spil (x dus) heeft grote invloed op de complexiteit van Quicksort. Standaard het eerste array-element ($A[p]$) kiezen als spil is een slechte keuze.

Voeg ter verbetering op plek (*) in Partitie toe:

Kies een slim array-element en wissel dat met $A[p]$.

Slim kan zijn: kies een willekeurig* array-element. De worst case blijft dan uiteraard $\Theta(n^2)$, maar onder de aanname dat alle $A[i]$ verschillend zijn hebben we nu:

Stelling
Quicksort doet $O(n \lg n)$ vergelijkingen in de average case.

*Randomised Quicksort

Randomised Quicksort

```
1  Partitie(A, p, r)::
2      wissel willekeurig array-element met A[p];
3      x := A[p]; i := p − 1; j := r + 1;
4      while i < j do
5          j := j − 1;                    // loop met j naar links
6          while A[j] > x do
7              j := j − 1;
8          od                  // tot je een waarde A[j] ≤ x vindt
9          i := i + 1;                    // loop met i naar rechts
10         while A[i] < x do
11             i := i + 1;
12         od                  // tot je een waarde A[i] ≥ x vindt
13         if i < j then
14             wissel(A[i], A[j]);
15         fi  // A[i] en A[j] staan nu in het goede stuk
16     od
17     return j;                                  // dit wordt dus q
```

Sorteermethoden

Algoritme	worst case	average case	extra ruimte
Insertion sort	$\frac{1}{2}n^2$	$\Theta(n^2)$	"in place" (d.w.z. geen)
Quicksort	$\frac{1}{2}n^2$	$\Theta(n \lg n)$	proportioneel aan $\lg n$
Mergesort	$n \lg n$	$\Theta(n \lg n)$	proportioneel aan n
Heapsort	$2n \lg n$	$\Theta(n \lg n)$	in place

Heapsort: zie dictaat, opgave 38

Vergelijken

Vergelijking van verschillende sorteeralgoritmen
(average case; tijd in seconden)

n	Insertion $O(n^2)$	Shellsort $O(n^{7/6})$	Heapsort $O(n \lg n)$	Quicksort $O(n \lg n)$	Quicksort* $O(n \lg n)$
10	0,00044	0,00041	0,00057	0,00052	0,00046
100	0,00675	0,00171	0,00420	0,00284	0,00244
1000	0,59564	0,02927	0,05565	0,03153	0,02587
10 000	58,864	0,42998	0,71650	0,36765	0,31532
100 000	—	5,7298	8,8591	4,2298	3,5882
1 000 000	—	71,164	104,68	47,065	41,282

*geoptimaliseerd

Om te onthouden van vandaag

- Recurrente betrekkingen: oefenen (ook de bewijzen!)
- 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 $\in \Omega(n \lg n)$

(Werk)college

Volgende college: dinsdag 14 maart, 9u00–10u45, zaal C1 (Gorlaeus)

Werkcollege: zodadelijk van 11u00 tot 12u45, computerzalen 302–304, 306–308 en 303 (Snellius)
Opgaven uit het dictaat: 15, 16, 31, 23, 24, 18

Huiswerk

Huiswerk 2

Verschijnt vanmiddag op de website.

Inleveren via Brightspace, uiterlijk maandag**ochtend** 27 maart, **11u59**.