# Zesde college complexiteit

$\pi = 3,14159265358979323846\cdots$

14 maart 2025

## Recurrente betrekkingen

## Mergesort

## Ondergrens sorteren

- Insertion sort
  - worst case: $\frac{1}{2}n(n-1)$;
    average case $\frac{1}{4}n(n-1)$ + lagere orde termen
  - optimaal voor compare-exchange algoritmen met buur-verwisselingen
- inversies
  - paar $(A[i], A[j])$ met $i < j$, maar $A[i] > A[j]$
  - sorteeralgoritme moet alle inversies opheffen
- recurrente betrekkingen

```
int grootste(int A[ ], n) {
    if n = 1 then
        return A[n];
    else
        max := grootste(A, n − 1);
        if A[n] > max then
            max := A[n];
        fi
    fi
    return max;
}
```

werk per aanroep ($n > 1$)

Laat $C(n)$ = aantal arrayvergelijkingen ($A[n] > max$), dan geldt:

$$C(n) = \begin{cases} 0 & n = 1 \\ C(n-1) + 1 & n > 1 \end{cases}$$

Oplossing: $C(n) = n - 1$

Nog een voorbeeld van een recurrente betrekking (opgave 14b):

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

- ▷ vorm een vermoeden voor een gesloten formule voor $T(n)$ door middel van:
  - herhaalde substitutie en afleiden algemene vorm, of:
  - probeer wat termen door te rekenen: zie je een patroon?
- ▷ bewijs ten slotte de formule met volledige inductie

- ander voorbeeld (reeds gezien)

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

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

- het kan vaak helpen om termen los te laten staan, waardoor je een sommatie beter kan herkennen, en/of de regelmaat eerder kunt ontdekken

- volledige inductie:

  (i) *zwak*: neem aan dat het geldt voor $n - 1$*

  (ii) *sterk*: neem aan dat het geldt voor *alle* voorgaande $n$†

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

†met $n = 2^k, k \geq 0$ of $\cdots$

De vorige recurrente betrekking, maar nu voor algemene $n$, dus niet alleen voor tweemachten:

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

Dan geldt: $T(n) \in O(n \lg n)^*$ (en overigens ook $T(n) \in \Theta(n \lg n)$).

Dit kan worden bewezen door met behulp van volledige inductie bijvoorbeeld aan te tonen dat $T(n) \leq 2n \lg n$ voor alle $n \geq 2$.

*voor een algemene stelling, zie bijvoorbeeld de Smoothness Rule in The Design & Analysis of Algorithms van A. Levitin

En ten slotte nog twee voorbeelden om op te lossen.

(i) $T(n) = \begin{cases} 3 & n = 1 \\ T(n-1) + n - 1 & n > 1 \end{cases}$

Oplossing: $T(n) = 3 + \frac{1}{2}n(n-1)$

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

Oplossing: $T(n) = \cdots$

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

- # arrayvergelijkingen $\geq$ # inversies invoerarray

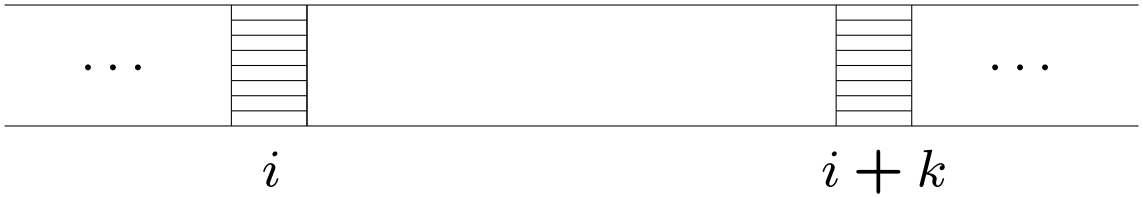
- # arrayvergelijkingen in de worst case $\geq \frac{1}{2}n(n-1)$

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 $A[i+k]$.

Er worden maximaal $2(k-1)+1 = 2k-1$ inversies opgeheven, minimaal 1. Zie werkcollege 7 maart.

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

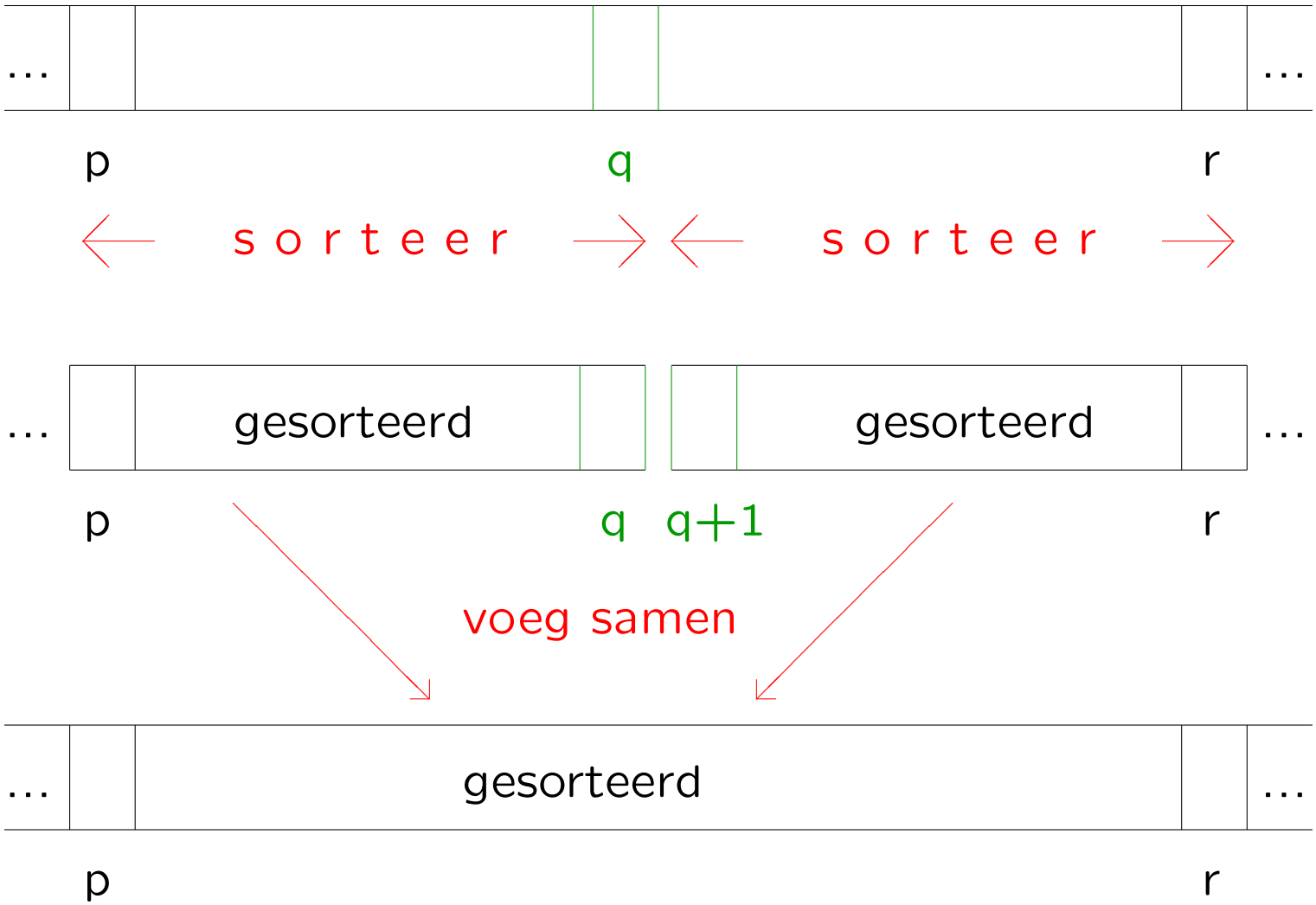
Het (recursieve) Mergesort* algoritme:

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

```
if p < r then
    q := ⌊(p+r)/2⌋;
    MergeSort(A, p, q);          // verdeel
    MergeSort(A, q + 1, r);      // en
    Merge(A, p, q, r);           // heers (voeg samen)
fi
```

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

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


...
...
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 $A$ (met $k$ elementen) en $B$ (met $m$ elementen) samenvoegt tot een gesorteerde rij $C$ met $n = k + m$ 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?

Veronderstel dat we met behulp van Merge twee gesorteerde (deel)rijtjes van respectievelijk $k$ en $m$ elementen (waarbij $k + m = n$) 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 arrayvergelijkingen is in dat geval namelijk altijd $\Theta(n)$ (waarom?), 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 arrayvergelijkingen in de worst case dat Mergesort doet 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 met behulp van array-vergelijkingen is in $\Omega(n \lg n)$). Er is echter extra geheugenruimte nodig ter grootte $\Theta(n)$.

Zij $B(n)$ = aantal arrayvergelijkingen 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: $B(n) = \frac{n}{2} \lg n \in \Theta(n \lg n)^*$.

*zie opgave 36

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.

## 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 $a_i < b_j \Leftrightarrow i < j$:

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

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 verder uit het ongerijmde.

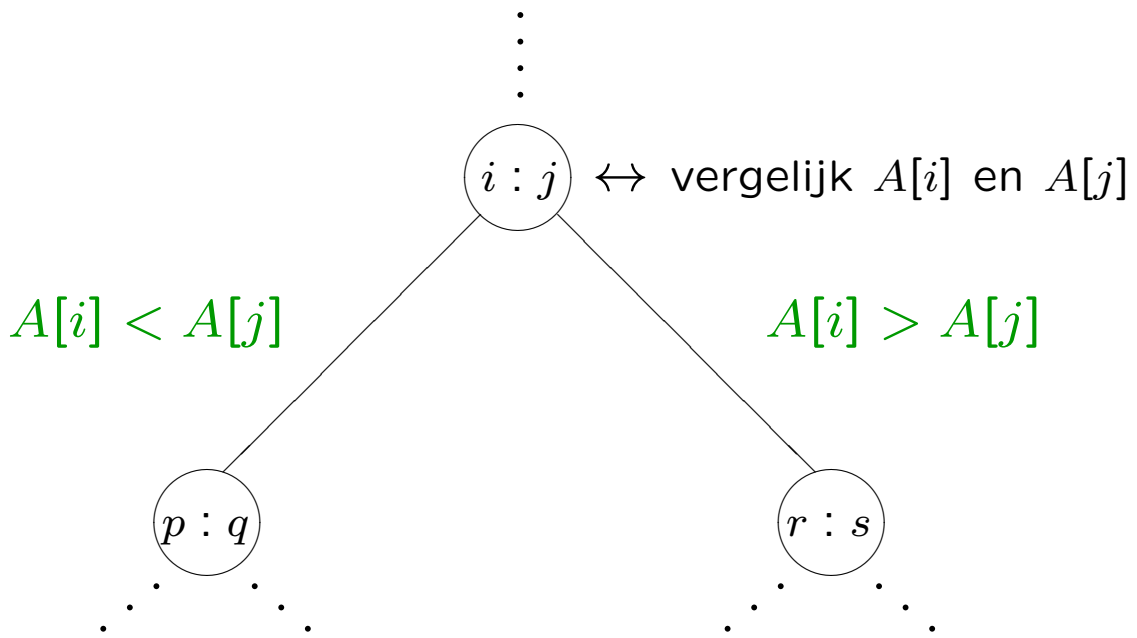
*We geven alleen het idee van het bewijs van 1. Punt 2. geheel analoog.

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

Aannames (z.b.d.a.):

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

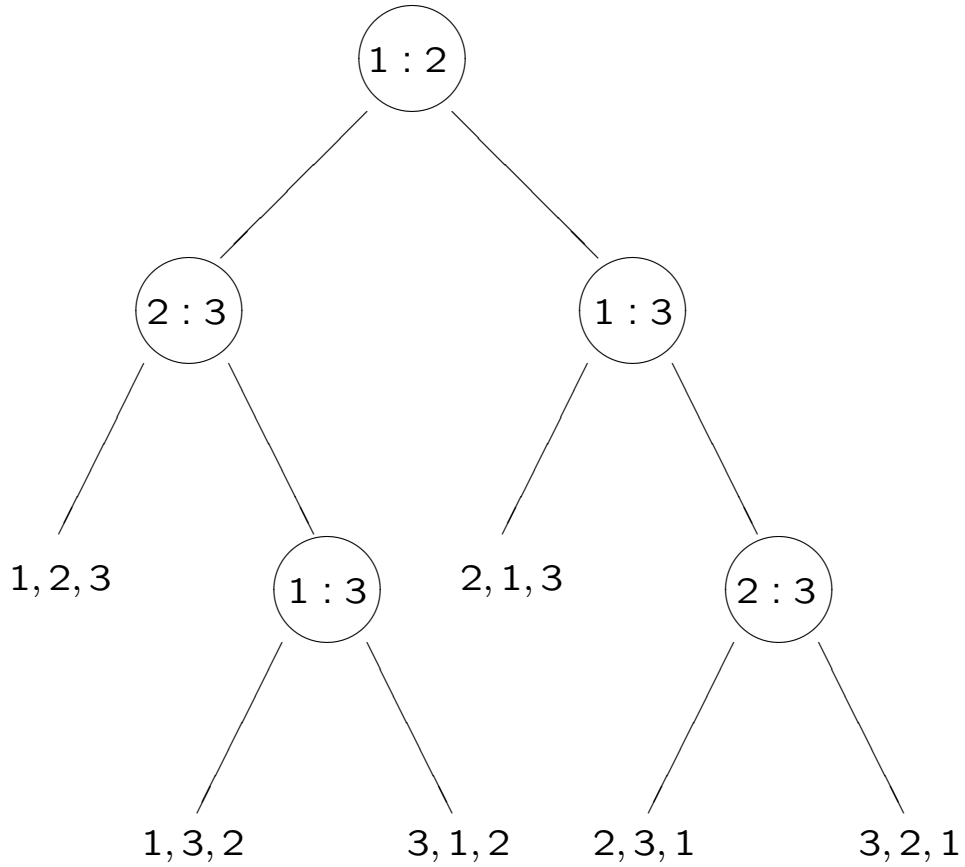
Zo'n algoritme correspondeert (voor elke $n$) met een beslissingsboom die de series vergelijkingen representeert die het algoritme uitvoert voor elke mogelijke invoer (ter grootte $n$). Elk pad van de wortel tot een blad correspondeert met een executie van het algoritme. In de bladeren (externe knopen) staat het eindresultaat, dus de gevonden sortering/ordening.




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

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

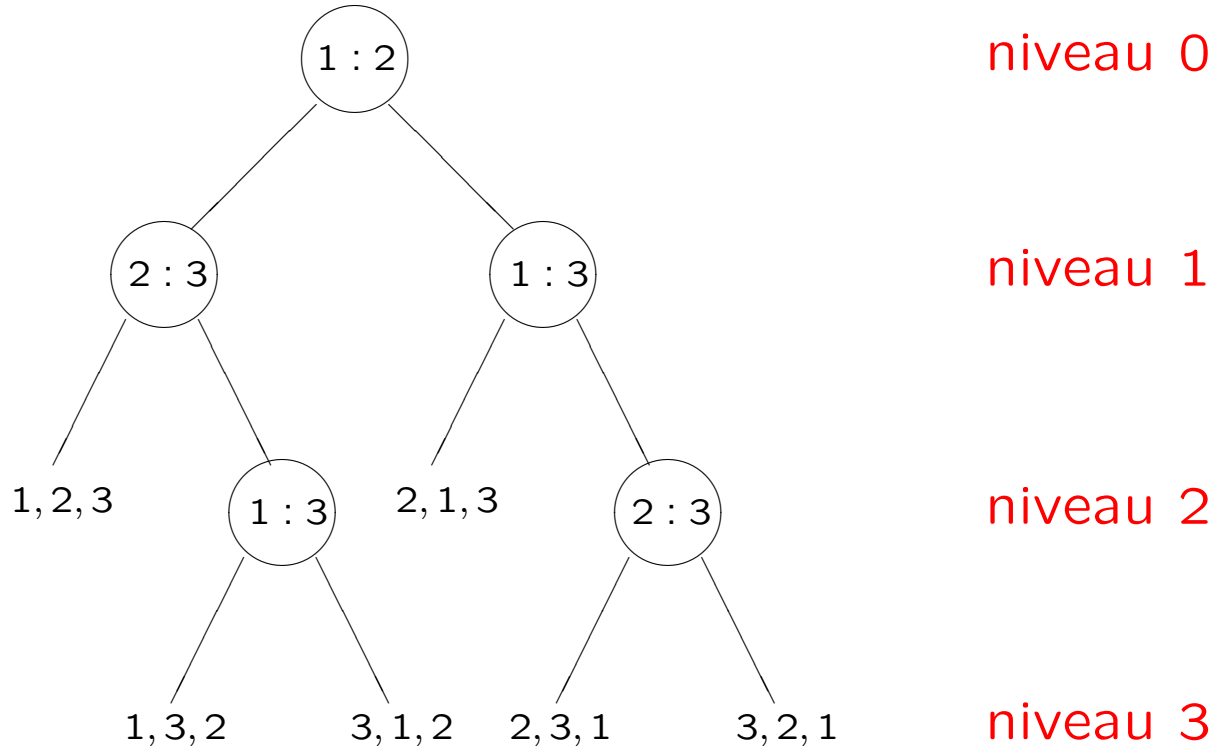
Beslissingsboom voor algoritmen gebaseerd op **arrayvergelijkingen:** beschrijft de werking op elke mogelijke invoer




Beslissingsboom voor Insertion sort met $n = 3$

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

## Hoogte $\leftrightarrow$ worst case




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

## 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 mogelijk, want er zijn $n!$ verschillende ordeningen

   - 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 (met $n$ het aantal array-elementen)

*voor *sorteren*

Stelling*

Het aantal arrayvergelijkingen in de worst case is voor elk algoritme dat sorteert middels arrayvergelijkingen ten minste $\lceil \lg n! \rceil$ (dus $\Omega(n \lg n)$).

Stelling†

Het aantal arrayvergelijkingen 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 van 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 gelijk is aan $n!$

Gegeven een binaire boom $\mathcal{B}$ met $b$ bladeren.

Definitie. De externe padlengte $E$ van $\mathcal{B}$ is de som van de lengtes van alle paden van de wortel naar een blad:

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

Lemma. Zij $E$ de externe padlengte van $\mathcal{B}$. Dan geldt:

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

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

Enige observaties:

- de externe padlengte is minimaal voor binaire bomen waarvoor alle knopen (inclusief de bladeren) allemaal zo dicht mogelijk bij de wortel zitten; de bladeren zitten dan allemaal op niveau $h-1$ of $h$, met $h$ de hoogte van de boom.

- de lengte van een pad van de wortel naar een blad is dan altijd $\geq h-1$.

- voor zo'n boom hebben we ook: $h \geq \lceil \lg b \rceil$.

- de lengte van een pad van de wortel naar een blad in een beslissingsboom is precies het aantal vergelijkingen dat het algoritme doet voor de bijbehorende invoer.

- **Volgende college:** vrijdag 21 maart, 11.00 – 12.45,
  Gorlaeusgebouw zaal BW.0.40

- **Straks werkcollege:** vrijdag 14 maart, 13.15 – 15.00,
  Gorlaeusgebouw zalen BW.0.29 en BW.0.30
  Opgaven: 15, 16, opg. 2 tentamen juni 2024, 34, 35, 36

- **Tweede huiswerkopgave:**
  komt volgende week

- **Website:**
  liacs.leidenuniv.nl/~graafjmde/COMP/