# Negende college algoritmiek

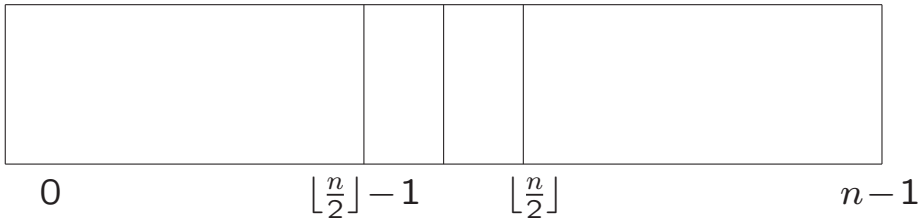
13 april 2026

## Verdeel en Heers

## Even praktisch. . .

- dinsdag 14 april: werkcollege
- Programmeeropdracht 1
  - deadline: 20 april 2026, 23.59 uur
  - woensdag 15 april: practicumbijeenkomst

## Divide and conquer




Mergesort


Quicksort

## Decrease and conquer (decrease by one)


Insertion sort

De sorteermethode Quicksort is, evenals Merge sort, gebaseerd op het verdeel en heers principe. Quicksort wordt in de praktijk veel gebruikt, omdat het (gemiddeld) zeer efficiënt sorteert.

Quicksort($A[l \cdots r]$)::
// sorteert het (sub)array $A[l \cdots r]$ recursief
// uitvoer: $A[l \cdots r]$ oplopend gesorteerd
**if** $l < r$
$s$ :=Partitie($A[l \cdots r]$); // s het splitspunt
Quicksort($A[l \cdots s-1]$);
Quicksort($A[s+1 \cdots r]$);
**fi .**

Voorbeeld: 5 3 1 9 8 2 4 7

## Partitie

Partitie($A[l \cdots r]$) ::

// partitioneert een (sub)array, met $A[l]$ als spil (pivot)

$p := A[l]$;

$i := l; j := r + 1$;

**repeat**

**repeat** $i := i + 1$; **until** $i > r$ **or** $A[i] \geq p$;

**repeat** $j := j - 1$; **until** $A[j] \leq p$;

**if** $i < j$ **then**

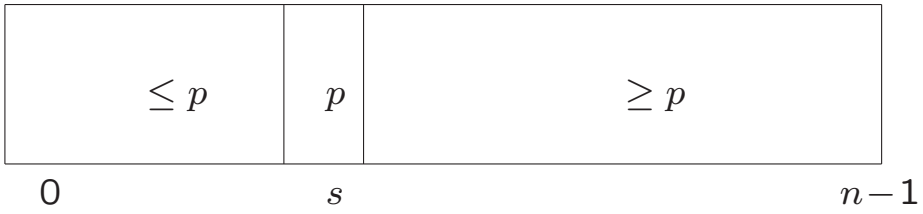
Wissel($A[i], A[j]$);

**if**

**until** $i \geq j$;

Wissel($A[l], A[j]$);

**return** $j$; .



Opgave 1, werkcollege 9:

**Probleem**: reorganiseer de elementen van een gegeven array $A$ zodanig dat alle negatieve elementen voorafgaan aan de positieve. Het algoritme moet lineair zijn en in situ. Hint: vergelijk Partitie.

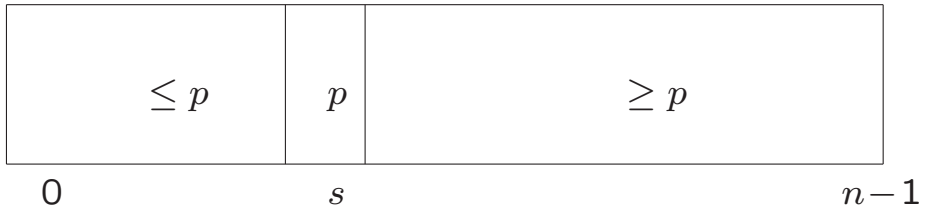
Opgave 2, werkcollege 9:

## Variant

Dutch Flag Problem

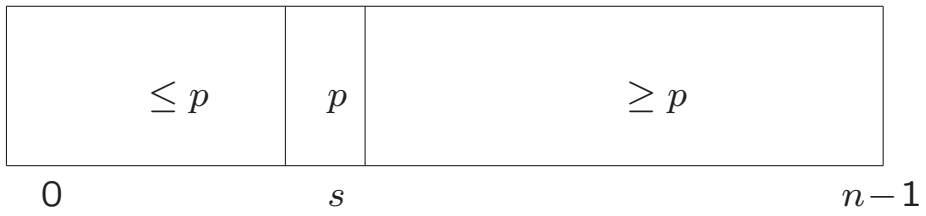
Gegeven een array gevuld met R, W, en B.
Reorganiseer dit array zo dat van links naar rechts eerst alle 'R', dan de 'W' en dan de 'B' komen te staan. Het algoritme moet lineair zijn en in situ (alleen interne verwisselingen). Het mag maar één doorgang door het array maken.

Zie Levitin, opgave 5.2.9.a.




Quicksort

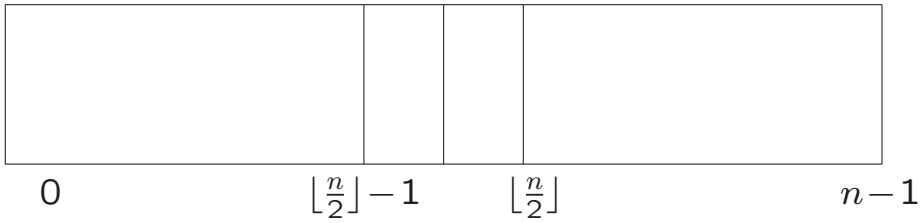
- ruimte complexiteit: . . .
- best case tijd: . . .
  wat zijn best cases?
- worst case tijd: . . .
  wat zijn worst cases?
- aantal worst cases. . .
- average case tijd: . . .




Quicksort

- ruimte complexiteit: $O(\log n)$
- best case tijd: $\Theta(n \log n)$
- worst case tijd: $\Theta(n^2)$
- aantal worst cases: $2^{n-1}$
- average case tijd: $\Theta(n \log n)$

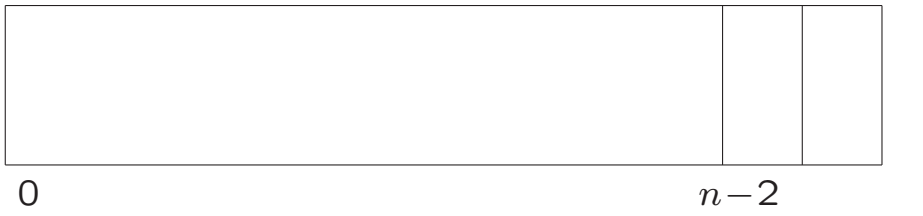
## Divide and conquer




Mergesort


Quicksort

## Decrease and conquer (decrease by one)




Insertion sort

## DECREASE by one & CONQUER

Insertionsort($A[0 \cdots m-1]$)::
  **if** $m > 1$
    Insertionsort($A[0 \cdots m-2]$);
    Voeg $A[m-1]$ op de juiste plek in;
  **fi** .

Invoegen van $A[m-1]$ in het reeds gesorteerde voorstuk $A[0] \cdots A[m-2]$ door van rechts naar links $A[m-1]$ te vergelijken met $A[i]$. Deze recursieve versie komt overeen met de iteratieve versie zoals bij Programmeermethoden behandeld (zie ook Levitin):

$$A[0] \leq A[1] \leq \ldots \leq A[i] \leq A[i+1] \leq \cdots \leq A[m-3] \leq A[m-2] || A[m-1] \cdots$$

kleiner of gelijk $A[m-1]$ ↑ groter dan $A[m-1]$

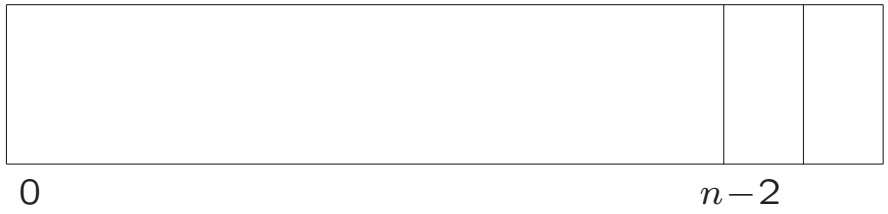
hier invoegen

Voorbeeld: 5 3 1 9 8 2 4 7

Iteratief

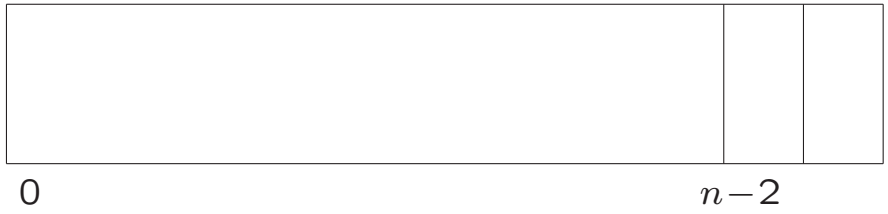
Insertionsort($A[0 \cdots n-1]$)::

```
    for i = 1 to n − 1 // A[0..i − 1] is al gesorteerd
        getal = A[i];
        j = i − 1;
        while j ≥ 0 and getal < A[j]
            A[j + 1] = A[j]; // schuif A[j] naar rechts
            j − −;
        od
        A[j + 1] = getal; // j is 1 'te laag' geworden
    od
```




Insertion sort

- ruimte complexiteit: . . .
- worst case tijd: . . .
  wat zijn worst cases?
- aantal worst cases: . . .
- average case tijd: . . .
- best case tijd: . . .
  wat zijn best cases?




Insertion sort

- ruimte complexiteit: iteratief $O(1)$
- worst case tijd: $\Theta(n^2)$
- aantal worst cases: 1 of $2^{n-1}$
- average case tijd: $\Theta(n^2)$
- best case tijd: $\Theta(n)$

Mergesort:

- worst case complexiteit: $\Theta(n \log n)$
- extra geheugen: $O(n)$

Quicksort:

- worst case complexiteit: $\Theta(n^2)$ voor (o.a.) het reeds gesorteerde rijtje
- average case complexiteit: $\Theta(n \log n)$
- extra geheugen: $O(\log n)$

Insertion sort:

- worst case/average case complexiteit: $\Theta(n^2)$
- extra geheugen: in situ

Vermenigvuldiging van grote integers:

Het voor de hand liggende algoritme gebruikt voor de vermenigvuldiging van twee getallen bestaande uit $n$-cijfers (digits) $n^2$ digit-vermenigvuldigingen

Voorbeeld (n=2): $12 * 34 = \ldots$

Vermenigvuldiging van grote integers:

Het voor de hand liggende algoritme gebruikt voor de vermenigvuldiging van twee getallen bestaande uit $n$-cijfers (digits) $n^2$ digit-vermenigvuldigingen Het kan echter op magische wijze beter (althans voor zeer grote getallen) via divide and conquer. Gebruik een generalisatie van de volgende truc (met $n = 2$):

$$c = a * b = (a_1 10^1 + a_0) * (b_1 10^1 + b_0) = c_2 10^2 + c_1 10^1 + c_0$$

$$c_2 = a_1 * b_1$$

$$c_0 = a_0 * b_0$$

$$c_1 = (a_1 + a_0) * (b_1 + b_0) - (c_2 + c_0)$$

Voor $n = 2$ zijn hier dus 3 i.p.v. 4 digit-vermenigvuldigingen gebruikt!

Voorbeeld $n = 8$:

$$87593264 * 49367251 =$$

$$(8759 \cdot 10^4 + 3264) * (4936 \cdot 10^4 + 7251) = c_2 10^8 + c_1 10^4 + c_0$$

$$c_2 = 8759 * 4936$$

$$c_0 = 3264 * 7251$$

$$c_1 = 8759 * 7251 + 3264 * 4936 =$$

$$(8759 + 3264) * (4936 + 7251) - (c_2 + c_0)$$

Voorbeeld $n = 8$:

$$87593264 * 49367251 =$$
$$(8759 \cdot 10^4 + 3264) * (4936 \cdot 10^4 + 7251) = c_2 10^8 + c_1 10^4 + c_0$$

$$c_2 = 8759 * 4936$$
$$c_0 = 3264 * 7251$$
$$c_1 = 8759 * 7251 + 3264 * 4936 =$$
$$(8759 + 3264) * (4936 + 7251) - (c_2 + c_0)$$

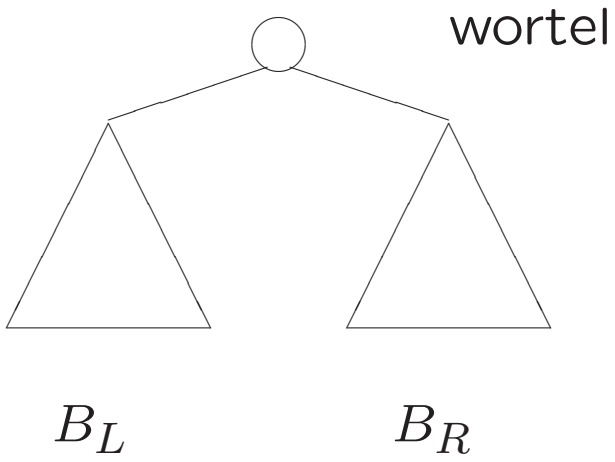
Het vermenigvuldigen van twee getallen bestaande uit $n = 2^k$ bits is zo teruggebracht tot 3 keer hetzelfde probleem voor $n/2 = 2^{k-1}$. Als $M(n)$ het aantal digitvermenigvuldigingen is voor $n = 2^k$, dan voldoet $M(n)$ aan:

$$M(n) = 3 * M(n/2) \text{ als } n > 1; M(1) = 1,$$

en vinden we: $M(n) = n^{\lg 3} < n^{\lg 4} = n^2$.

Een binaire boom $B$ wordt recursief gedefinieerd als ofwel leeg, ofwel bestaande uit een knoop (de wortel) en twee disjuncte subbomen $B_L$ en $B_R$ die beide ook weer een binaire boom zijn: de linkersubboom en de rechtersubboom.

$B =$




Bij (veel) problemen met binaire bomen ligt oplossen via divide & conquer dus voor de hand.

De hoogte van een binaire boom is het hoogste niveau dat voorkomt, waarbij de wortel per definitie op niveau 0 zit.

Voor de hoogte van een binaire boom $B$ geldt dus:

$$\text{hoogte}(B) = 1 + \texttt{max}\ \{\ \text{hoogte}(B_L),\ \text{hoogte}(B_R)\ \}$$

Verdeel en heers algoritme in C++:

```
int hoogte( knoop * root ) {
  if ( root == nullptr )      // lege boom
    return -1;
  else
    return ( 1 + max( hoogte( root->links ), hoogte( root->rechts ) ) );
}  // hoogte
```

## DECREASE by one & CONQUER

Insertionsort($A[0 \cdots m-1]$)::
    **if** $m > 1$
        Insertionsort($A[0 \cdots m-2]$);
        Voeg $A[m-1]$ op de juiste plek in;
    **fi** .

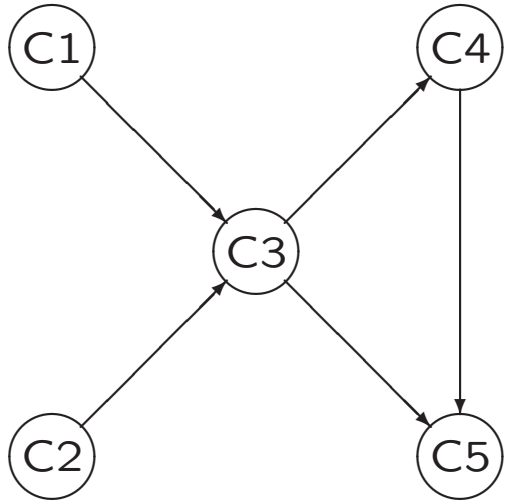
Invoegen van $A[m-1]$ in het reeds gesorteerde voorstuk $A[0] \cdots A[m-2]$ door van rechts naar links $A[m-1]$ te vergelijken met $A[i]$. Deze recursieve versie komt overeen met de iteratieve versie zoals bij Programmeermethoden behandeld (zie ook Levitin):

$$A[0] \leq A[1] \leq \ldots \leq A[i] \leq A[i+1] \leq \cdots \leq A[m-3] \leq A[m-2] || A[m-1] \cdots$$

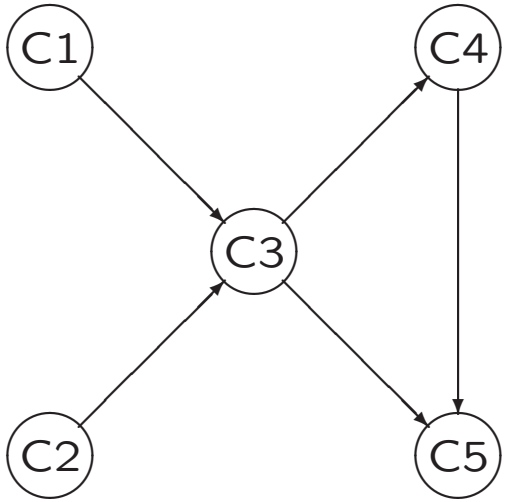
kleiner of gelijk $A[m-1]$ ↑ groter dan $A[m-1]$

hier invoegen

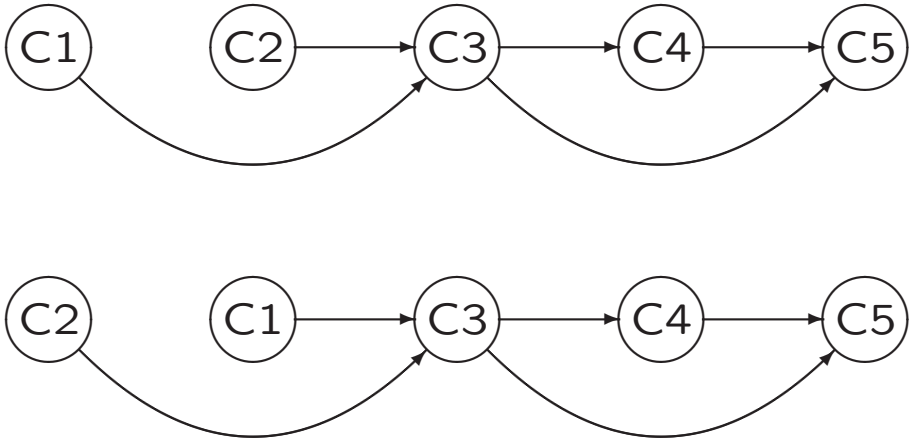
Gerichte grafen, b.v. met benodigde voorkennis voor cursussen:



Gerichte grafen, b.v. met benodigde voorkennis voor cursussen:




Mogelijke volgordes



Gerichte grafen

- eis aan graaf
- algoritme
- implementatie
- efficiëntie
- ander algoritme
- efficiëntie
- effect algoritmes als graaf cyclisch is

Gerichte grafen

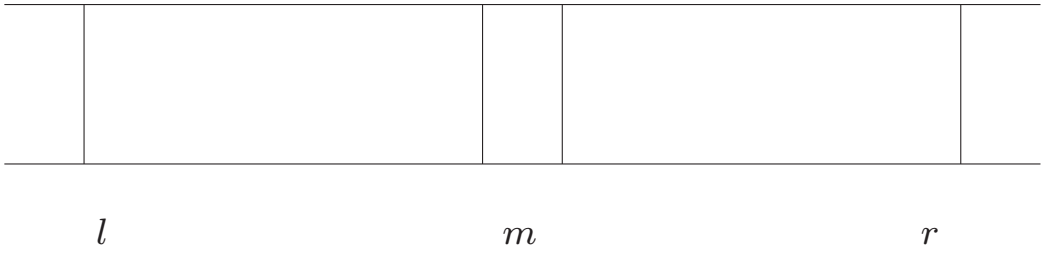
- eis aan graaf: acyclisch
- algoritme: source-removal
- implementatie: aantal inkomende takken per knoop bijhouden
- efficiëntie: $\Theta(n+m)$
- ander algoritme: DFS
- efficiëntie: $\Theta(n+m)$
- effect algoritmes als graaf cyclisch is: verschillend

Zie paragraaf 4.2 van boek + antwoorden werkcollege 9.

Binair zoeken is een voorbeeld van decrease and conquer, decrease by a constant factor. Hier de iteratieve versie.

// invoer: oplopend gesorteerd array $A[0..n-1]$, te zoeken waarde $K$
// uitvoer: positie van $K$ in $A$ (-1 als $K$ niet in $A$ zit)

$l := 0; r := n-1;$
**while** $l \leq r$ **do**
  $m := \lfloor \frac{l+r}{2} \rfloor;$
  **if** $K = A[m]$ **then** // gevonden
    **return** $m$;
  **else if** $K < A[m]$ **then** // links verder zoeken
    $r = m - 1;$
  **else** // rechts verder zoeken
    $l = m + 1;$ **fi**
  **fi**
**od**
**return** $-1$;




altijd *links óf rechts* verdergaan

## Fake coin probleem

Gegeven $n$ identiek uitziende munten. Eén ervan is vals. Bekend is dat de valse munt lichter is dan de andere. Tevens is een balans beschikbaar. Bepaal door weging de valse munt.

Decrease by a constant factor: verdeel de munten in twee stapels van $\lfloor \frac{n}{2} \rfloor$ en —indien $n$ oneven— een losse munt. Als de twee stapels even zwaar zijn (best case) is de losse munt de valse. Zo niet, dan bevindt de valse zich in de lichtste van de twee stapels.

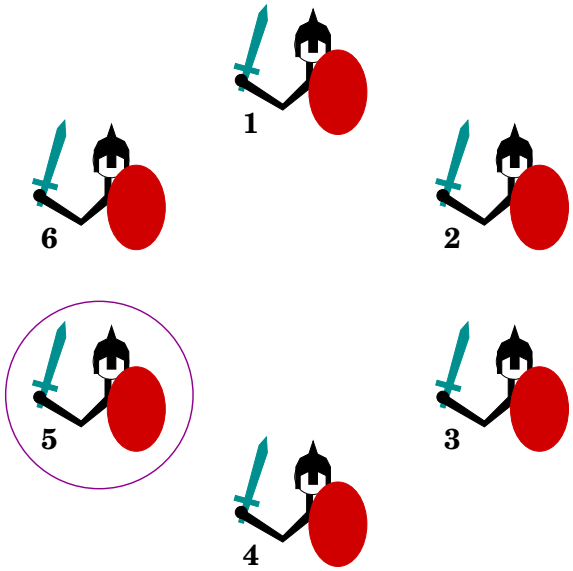
Recurrente betrekking voor het aantal wegingen dat nodig is in de worst case om de valse munt te ontdekken:

$$W(n) = W(\lfloor \tfrac{n}{2} \rfloor) + 1 \text{ als } n > 1,\ W(1) = 0$$

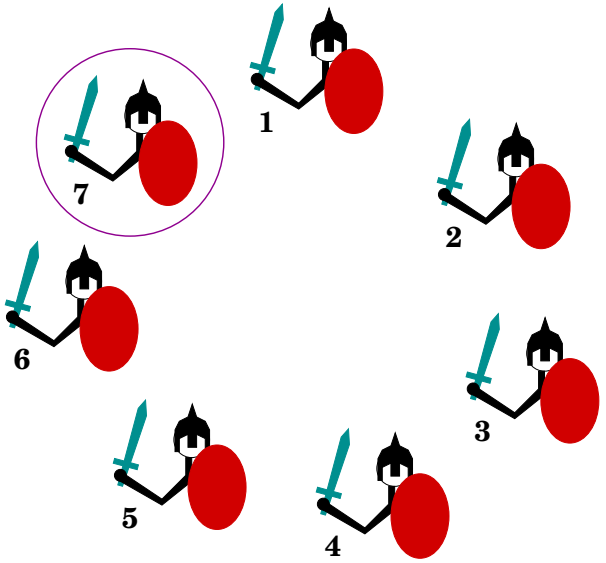
Oplossing: $W(n) = \lfloor \lg n \rfloor$.

Zie Levitin excercise 4.5.10 voor een efficiënter decrease by a constant factor algoritme.

Josephus probleem: gegeven $n$ personen, genummerd 1 t/m $n$, die in een cirkel staan. Elimineer, te beginnen bij persoon 2, telkens elke tweede persoon, totdat er nog maar één persoon, $J(n)$, over is. Bepaal wie deze overlevende is.




Josephus 6




Josephus 7

Decrease by a constant factor: maak één doorgang door de cirkel. Er zijn dan nog $\lfloor \frac{n}{2} \rfloor$ overlevenden in de cirkel over, dus hetzelfde probleem maar gehalveerd.

Recurrente betrekking:

$$\begin{cases} J(1) & = & 1 \\ J(2k) & = & 2J(k) - 1 \\ J(2k+1) & = & 2J(k) + 1 \end{cases}$$

Oplossing:

| $n$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $J(n)$ | 1 | 1 | 3 | 1 | 3 | 5 | 7 | 1 | 3 | 5 | 7 | 9 | 11 | 13 | 15 | 1 |

Variable-size decrease: de reductie in grootte is variabel,
dus kan in elke stap anders zijn

**Voorbeelden:**

- Algoritme van Euclides
  Dit is gebaseerd op: ggd($m$, $n$) = ggd($n$, $m$ **mod** $n$)
- Flipping pancakes (Levitin, excercise 4.5.12)

- Binaire zoekbomen

**Probleem**:

*Gegeven*: twee niet-negatieve gehele getallen $m$ en $n$ (niet beide nul).

*Vraag*: wat is de grootste gemeenschappelijke deler, genoteerd als ggd($m$,$n$), van $m$ en $n$?

**Voorbeelden**:

ggd(60,24) = 12;
ggd(25,0) = 25;
ggd(200, 441) = 1;
ggd(588,495) = 3.

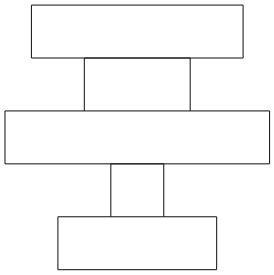
Het algoritme van Euclides is gebaseerd op het herhaald gebruiken van

$$\text{ggd}(m, n) = \text{ggd}(n, m \bmod n),$$

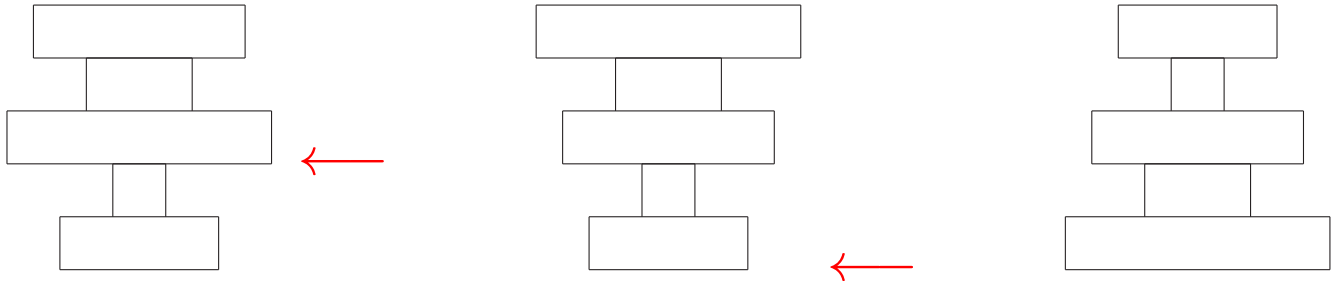
totdat de tweede parameter nul wordt.

**Voorbeeld**: ggd(60,24) = ggd(24, 12) = ggd(12, 0) = 12

**Probleem:** Gegeven een stapel van $n$ pannenkoeken, allemaal verschillend in grootte. Verder is alleen een spatel beschikbaar, die je onder een pannenkoek kan schuiven, waarna je de hele stapel daarbovenop in één keer kan omdraaien. De bedoeling is om uiteindelijk alle pannenkoeken bovenop elkaar te krijgen in volgorde van grootte (de grootste onderop).



**Probleem:** Gegeven een stapel van $n$ pannenkoeken, allemaal verschillend in grootte. Verder is alleen een spatel beschikbaar, die je onder een pannenkoek kan schuiven, waarna je de hele stapel daarbovenop in één keer kan omdraaien. De bedoeling is om uiteindelijk alle pannenkoeken bovenop elkaar te krijgen in volgorde van grootte (de grootste onderop).



# BOUNDS FOR SORTING BY PREFIX REVERSAL

William H. GATES
*Microsoft, Albuquerque, New Mexico*

Christos H. PAPADIMITRIOU*†
*Department of Electrical Engineering, University of California, Berkeley, CA 94720, U.S.A.*

Received 18 January 1978
Revised 28 August 1978

For a permutation $\sigma$ of the integers from 1 to $n$, let $f(\sigma)$ be the smallest number of prefix reversals that will transform $\sigma$ to the identity permutation, and let $f(n)$ be the largest such $f(\sigma)$ for all $\sigma$ in (the symmetric group) $S_n$. We show that $f(n) \leqslant (5n+5)/3$, and that $f(n) \geqslant 17n/16$ for $n$ a multiple of 16. If, furthermore, each integer is required to participate in an even number of reversed prefixes, the corresponding function $g(n)$ is shown to obey $3n/2-1 \leqslant g(n) \leqslant 2n+3$.

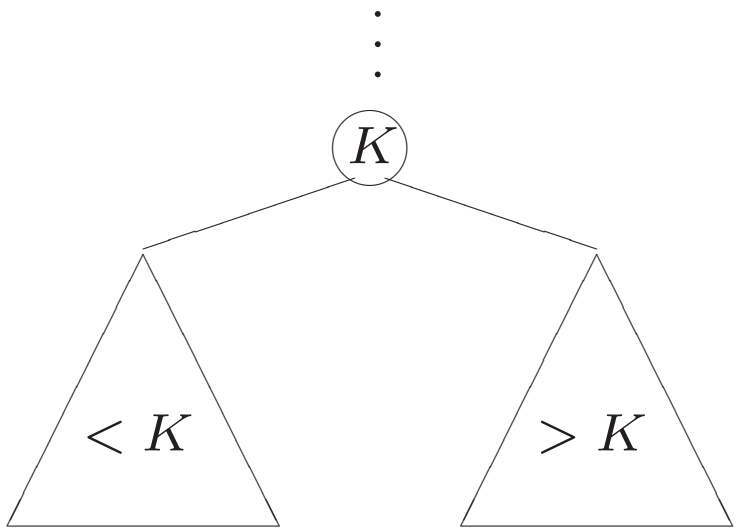
## 1. Introduction

We introduce our problem by the following quotation from [1]

> The chef in our place is sloppy, and when he prepares a stack of pancakes they come out all different sizes. Therefore, when I deliver them to a customer, on the way to the table I rearrange them (so that the smallest winds up on top, and so on, down to the largest at the bottom) by grabbing several from the top and flipping them over, repeating this (varying the number I flip) as many times as necessary. If there are $n$ pancakes, what is the maximum number of flips (as a function $f(n)$ of $n$) that I will ever have to use to rearrange them?

In this paper we derive upper and lower bounds for $f(n)$. Certain bounds were already known. For example, consider any stack of pancakes. An *adjacency* in

Een binaire zoekboom is een binaire boom waarbij voor elke knoop geldt dat de waarde in die knoop groter is dan alle waarden in zijn linkersubboom, en kleiner dan alle waarden in zijn rechtersubboom.



Bij het zoeken naar een waarde in een gewone binaire boom (bijv. WLR) moeten in het slechtste geval alle $n$ knopen bekeken worden.
Zoeken in een binaire zoekboom is i.h.a. efficiënter: in het slechtste geval worden $h+1$ knopen bekeken, met $h$ de hoogte van de boom.

```
knoop* zoeken(knoop* root, int getal) {
  if ( root == nullptr )      // lege boom
    return nullptr;
  else
    if ( root->info == getal )       // gevonden!
      return root;
    else
      if ( getal < root->info )
        return zoeken(root->links, getal);
      else
        return zoeken(root->rechts, getal);
}  // zoeken
```

Quicksort($A[l \cdots r]$)::
// sorteert het (sub)array $A[l \cdots r]$ recursief
// uitvoer: $A[l \cdots r]$ oplopend gesorteerd
**if** $l < r$
$s$ :=Partitie($A[l \cdots r]$); // s het splitspunt
Quicksort($A[l \cdots s-1]$);
Quicksort($A[s+1 \cdots r]$);
**fi .**

Voorbeeld: 5 3 1 9 8 2 4 7

## Partitie

Partitie($A[l \cdots r]$) ::

// partitioneert een (sub)array, met $A[l]$ als spil (pivot)

$p := A[l]$;

$i := l; j := r + 1$;

**repeat**

**repeat** $i := i + 1$; **until** $i > r$ **or** $A[i] \geq p$;

**repeat** $j := j - 1$; **until** $A[j] \leq p$;

**if** $i < j$ **then**

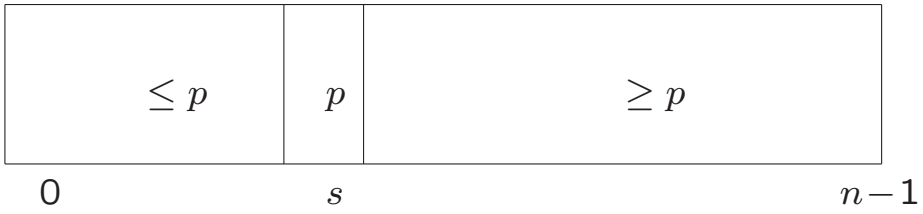
Wissel($A[i], A[j]$);

**if**

**until** $i \geq j$;

Wissel($A[l], A[j]$);

**return** $j$; .



Bekijk en vergelijk vier verschillende oplossingsmethoden voor het berekenen van $a^n$:

1. Brute force: gebaseerd op de definitie, $a^n = \overbrace{a * \cdots * a}^{n\times}$
2. Divide and conquer: gebaseerd op $a^n = a^{\lfloor \frac{n}{2} \rfloor} * a^{\lceil \frac{n}{2} \rceil}$
3. Decrease by one: gebaseerd op $a^n = a^{n-1} * a$
4. Decrease by a constant factor: ...

Bekijk en vergelijk vier verschillende oplossingsmethoden voor het berekenen van $a^n$:

1. Brute force: gebaseerd op de definitie, $a^n = \overbrace{a * \cdots * a}^{n\times}$

2. Divide and conquer: gebaseerd op $a^n = a^{\lfloor \frac{n}{2} \rfloor} * a^{\lceil \frac{n}{2} \rceil}$

3. Decrease by one: gebaseerd op $a^n = a^{n-1} * a$

4. Decrease by a constant factor: gebaseerd op

$$a^n = \begin{cases} (a^{\frac{n}{2}})^2 & \text{als } n \text{ even is} \\ (a^{\frac{n-1}{2}})^2 * a & \text{als } n \text{ oneven is} \end{cases}$$

- **Lezen/leren bij dit college:**
  5.2, 5.3, 4.inl, 4.1, 4.2, 4.4 (geen Russian peasant),
  4.5 (inl. + Binary Search Tree), slides;

- **Werkcollege** verdeel en heers:
  dinsdag 14 april 2026, 11.00–12.45, DM.1.09

- **Opgaven:**
  zie `https://liacs.leidenuniv.nl/~vlietrvan1/algoritmiek/`

- **Practicumbijeenkomst** programmeeropdracht 1:
  woensdag 15 april 2026, 15.15–17.00, DM.0.09, DM.0.13

- **Volgend college:**
  maandag 20 april 2026, 11.00-12.45