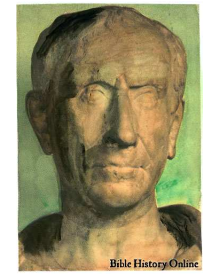
Zevende college algoritmiek

1 april 2016

Verdeel en Heers



Divide and Conquer

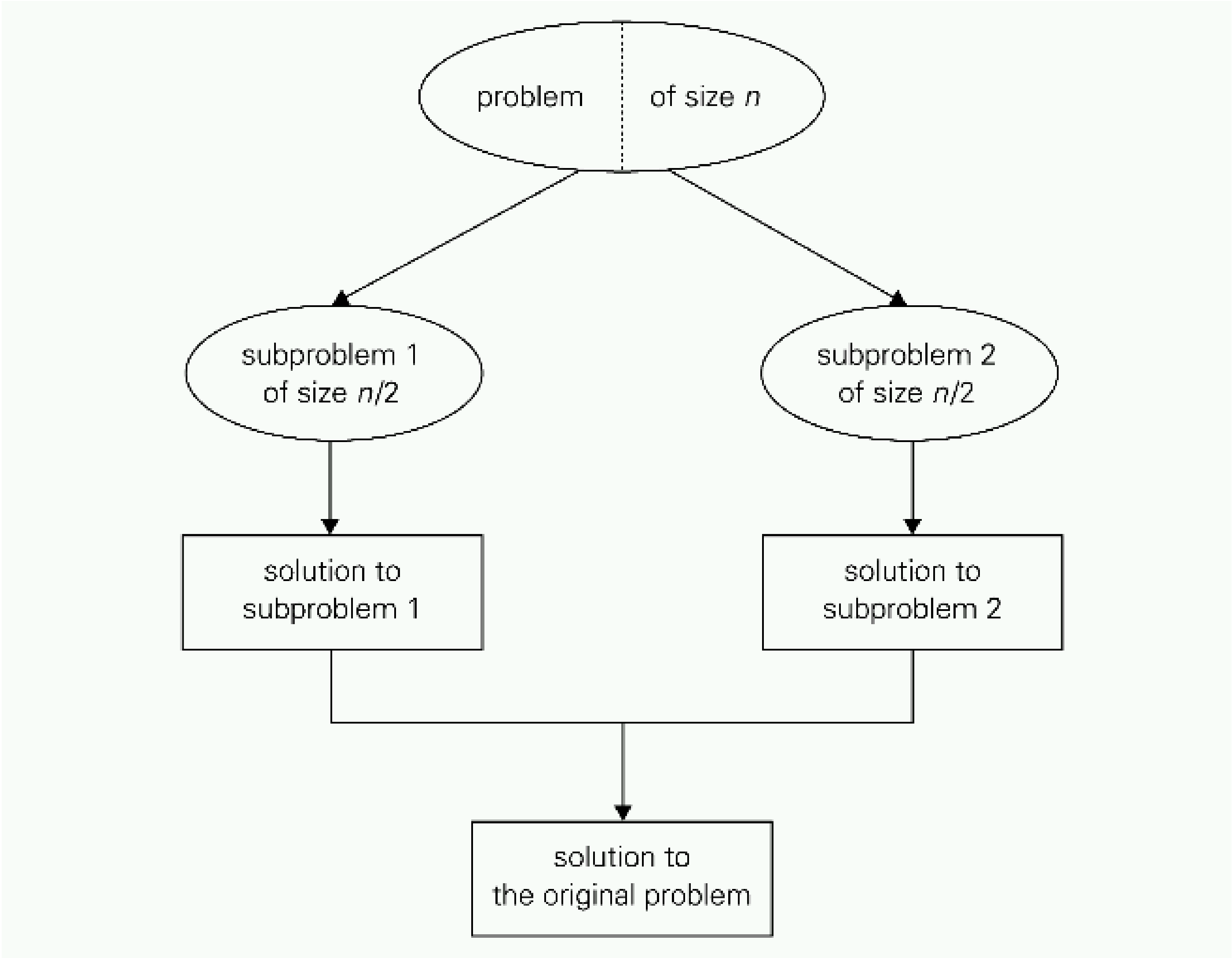
1. Verdeel een instantie van het probleem in twee (of meer) kleinere instanties
2. Los de kleinere instanties op: meestal recursief
3. Combineer deze twee (of meer) oplossingen tot een oplossing van de oorspronkelijke (grotere) instantie

Opmerking: meestal wordt een probleeminstantie in twee ongeveer gelijke delen verdeeld.

Schematisch (1)

Verdeel en heers

(vaak: verdeel in twee gelijke delen)



Decrease and Conquer

1. Reduceer een instantie van het probleem tot een kleinere instantie van hetzelfde probleem

2. Los de kleinere instantie op: vaak recursief

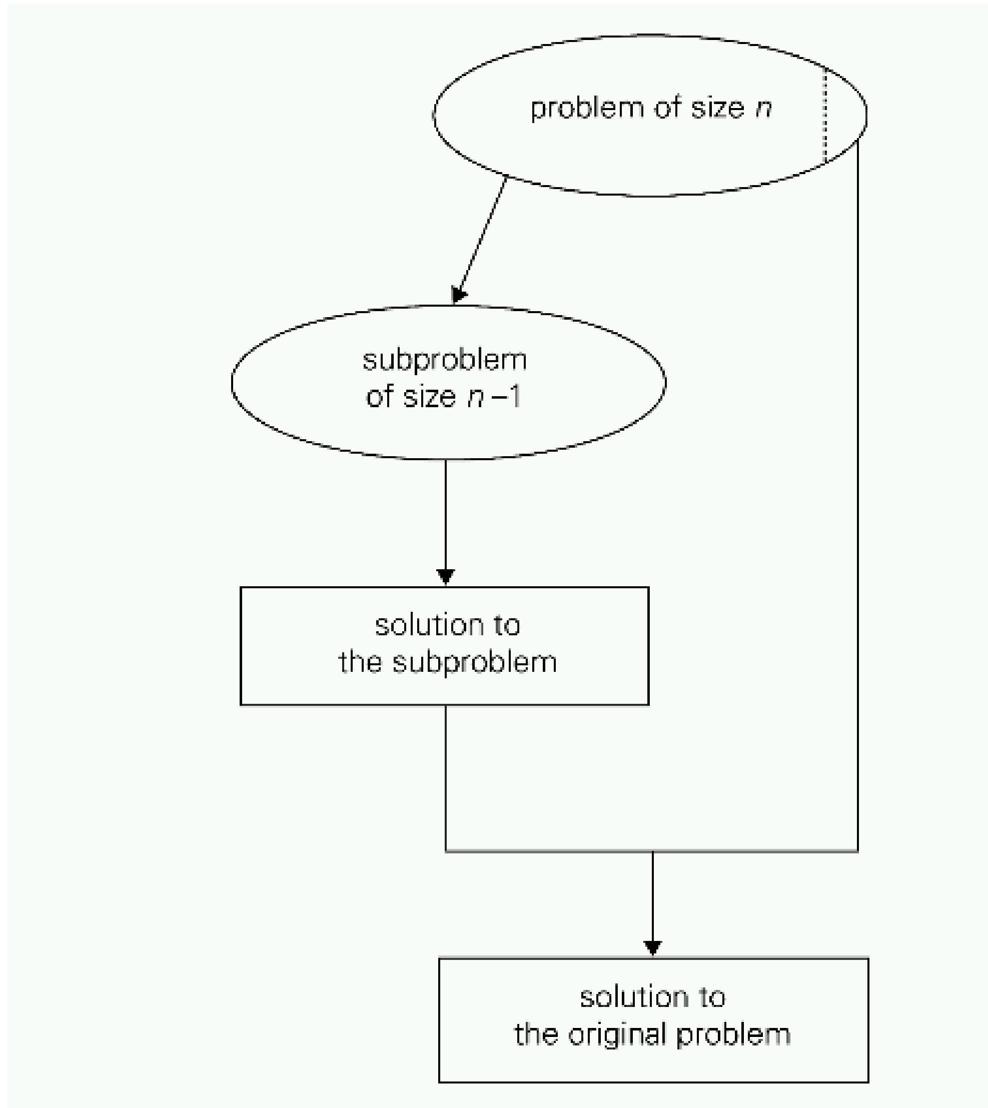
3. Breid de oplossing van de kleinere probleeminstantie uit tot een oplossing van de oorspronkelijke instantie

In het boek wordt onderscheid gemaakt tussen:

Decrease by one
Decrease by a constant factor
Variable-size decrease

Schematisch (2)

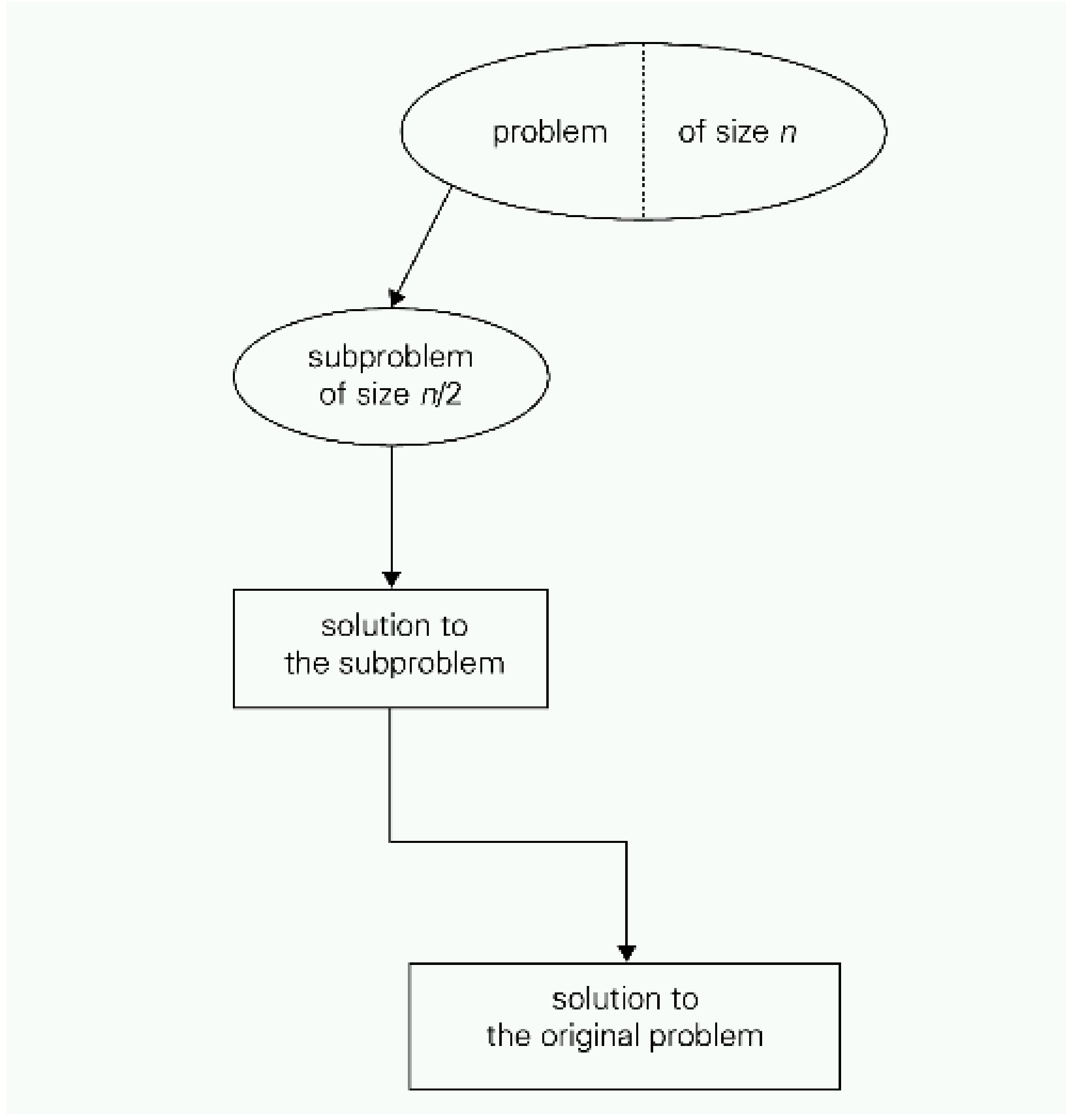
Decrease
by one



Schematisch (3)

Decrease by a constant factor

(decrease by half)



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 $n = 8$:

$$87593264 * 49367251 =$$

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

$$c_2 = 8759 * 4936$$

$$c_0 = 3264 * 7251$$

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

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

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:

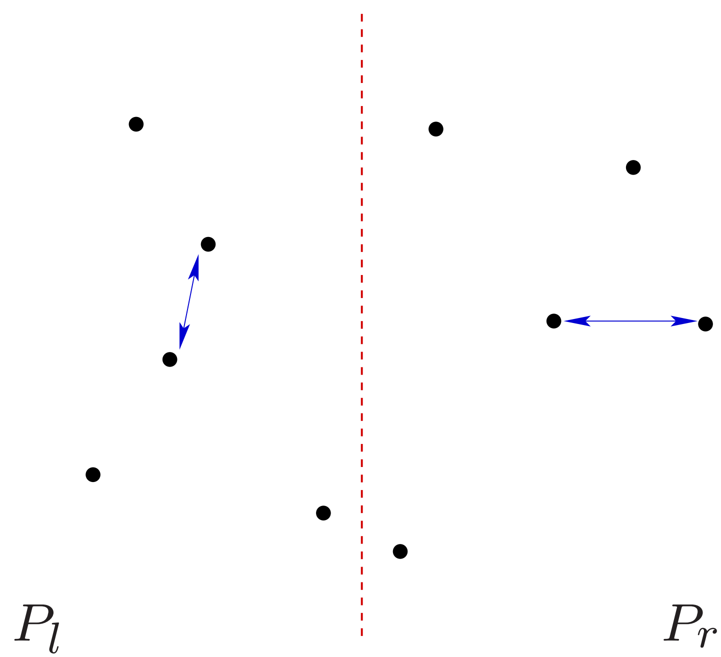
$$M(n) = 3 * M(n/2) \text{ als } n > 1; M(1) = 1,$$

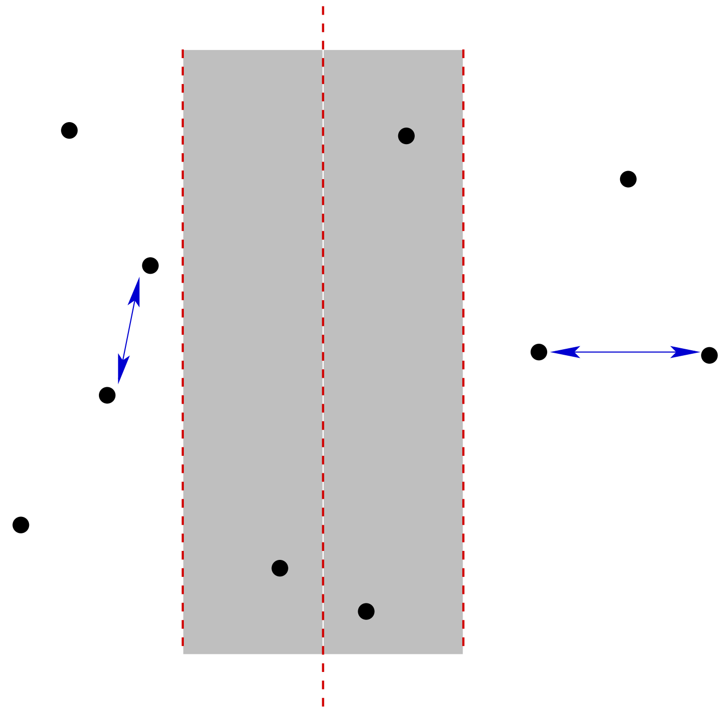
en vinden we: $M(n) = n^{\lg 3} < n^{\lg 4} = n^2$.

Divide and Conquer:

Verdeel de verzameling van n punten in twee verzamelingen P_l en P_r van elk $\frac{n}{2}$ punten door een geschikte lijn te trekken.

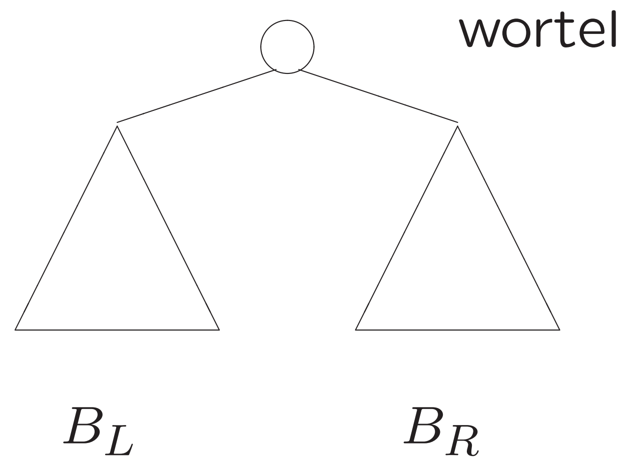
Los beide deelproblemen (recursief) op en laat d de kleinst voorkomende afstand zijn tussen punten van P_l resp. P_r.





Controleer of er binnen de strip ter breedte $2d$ rondom de scheidslijn tussen P_l en P_r puntenparen (p, p') zijn met $p \in P_l$ en $p' \in P_r$ en onderlinge afstand kleiner dan d.
Hiervoor blijken slechts $\mathrm{O}(n)$ puntenparen bekeken te moeten worden.
Dit levert een $\mathrm{O}(n \lg n)$ algoritme op.

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 nivo dat voorkomt, waarbij de wortel per definitie op nivo 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 == NULL )       // 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):

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

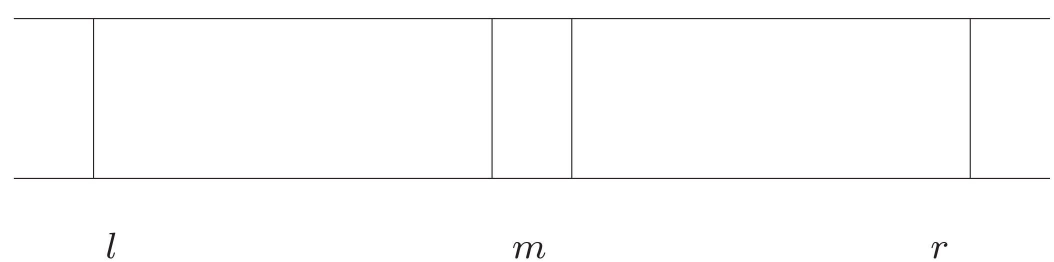
kleiner of gelijk $A[m-1]$ $\uparrow$ groter dan $A[m-1]$

hier invoegen

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)

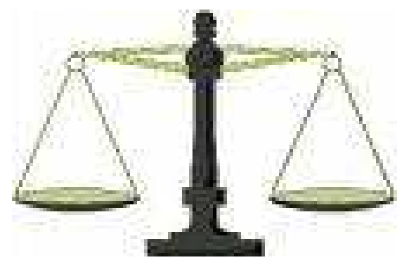
$l := 0; r := n-1;$
while $l \leq r$ **do**
  $m := \lfloor \frac{l+r}{2} \rfloor;$
  if $K = A[m]$ **then** // gevonden
    return m;
  else if $K < A[m]$ **then** // links verder zoeken
    $r = m - 1;$
  else // rechts verder zoeken
    $l = m + 1;$ **fi**
  fi
od
return -1;



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.

Decrease-by-half

Vermenigvuldig twee positieve gehele getallen n en m op basis van:

$$n * m = \frac{n}{2} * 2m \text{ als } n \text{ even}$$

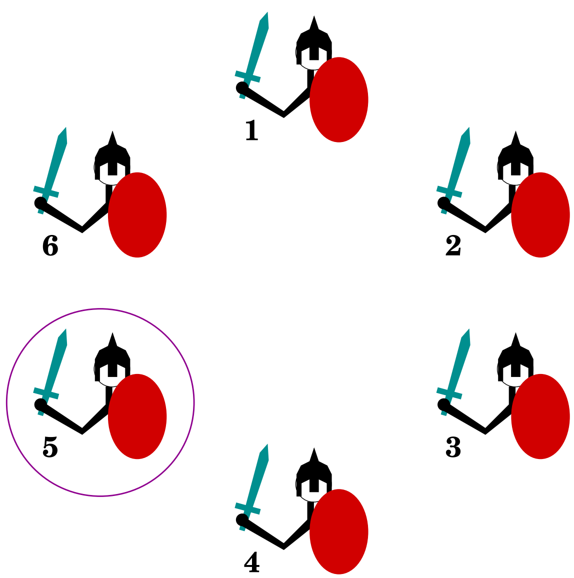
$$n * m = \frac{n-1}{2} * 2m + m \text{ als } n \text{ oneven}$$

$$1 * m = m$$

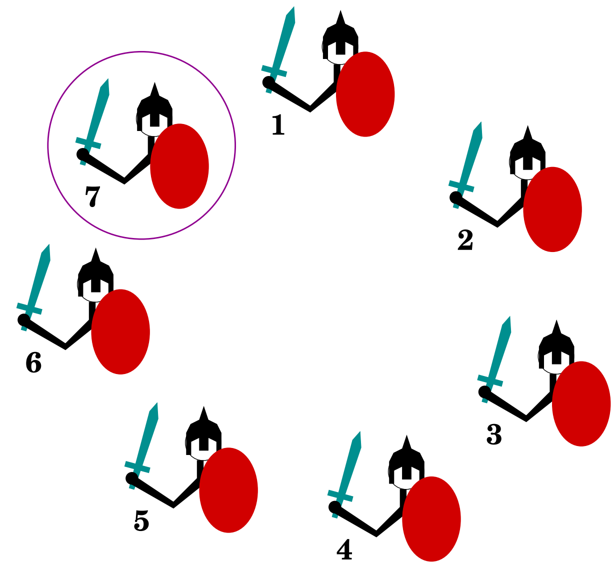
Voorbeeld: $50 * 65\ldots$

Zie Levitin 4.4: Russian Peasant Multiplication

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

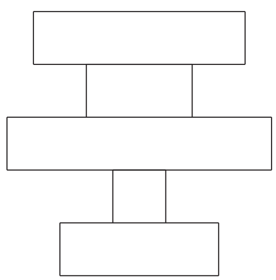
$$\text{ggd}(m,n) = \text{ggd}(n, m \bmod n),$$

totdat de tweede parameter nul wordt.

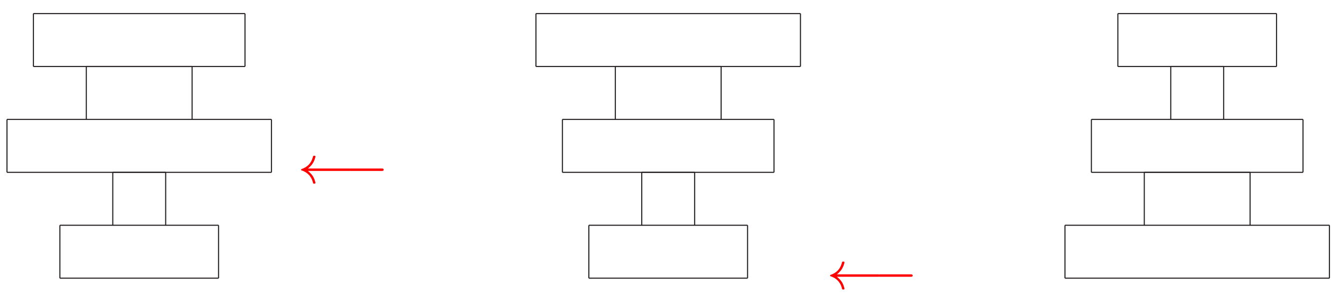
Voorbeeld: ggd(60,24) = ggd(24, 12) = ggd(12, 0) = 12

Algoritme van Euclides: worst case . . .

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 σ of the integers from 1 to n, let $f(\sigma)$ be the smallest number of prefix reversals that will transform σ to the identity permutation, and let $f(n)$ be the largest such $f(\sigma)$ for all σ 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 $3n/2 - 1 \leqslant g(n) \leqslant 2n+3$.

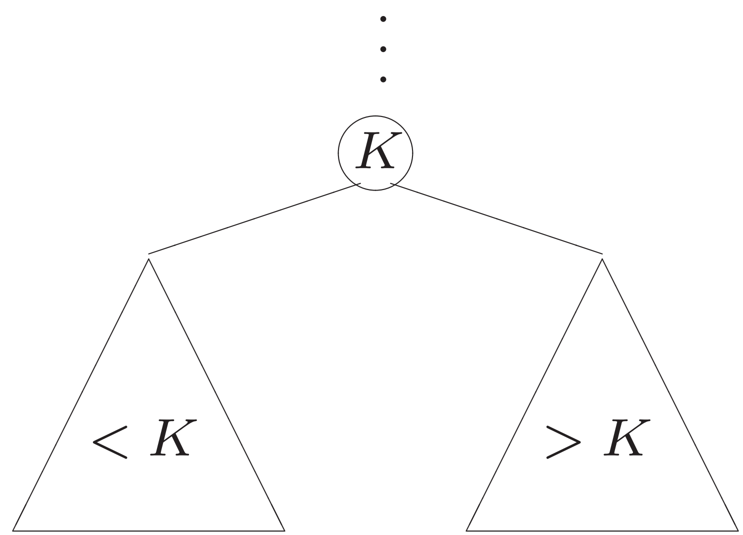
1. Introduction

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 == null )      // lege boom
    return null;
  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
```

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:**

  Paragrafen 4 inl., 4.1, 4.4, 5.2-5.5 (geen Strassen, convex hull)

- **Werkcollege:**

  donderdag 7 april 2016, 13:45–15:30, in B01/B02:

- **Opgaven:**

  zie `http://www.liacs.leidenuniv.nl/~graafjmde/ALGO/`

- **Volgend college:**

  vrijdag 8 april 2016