Zevende college algoritmiek

28 maart 2014

Verdeel & Heers

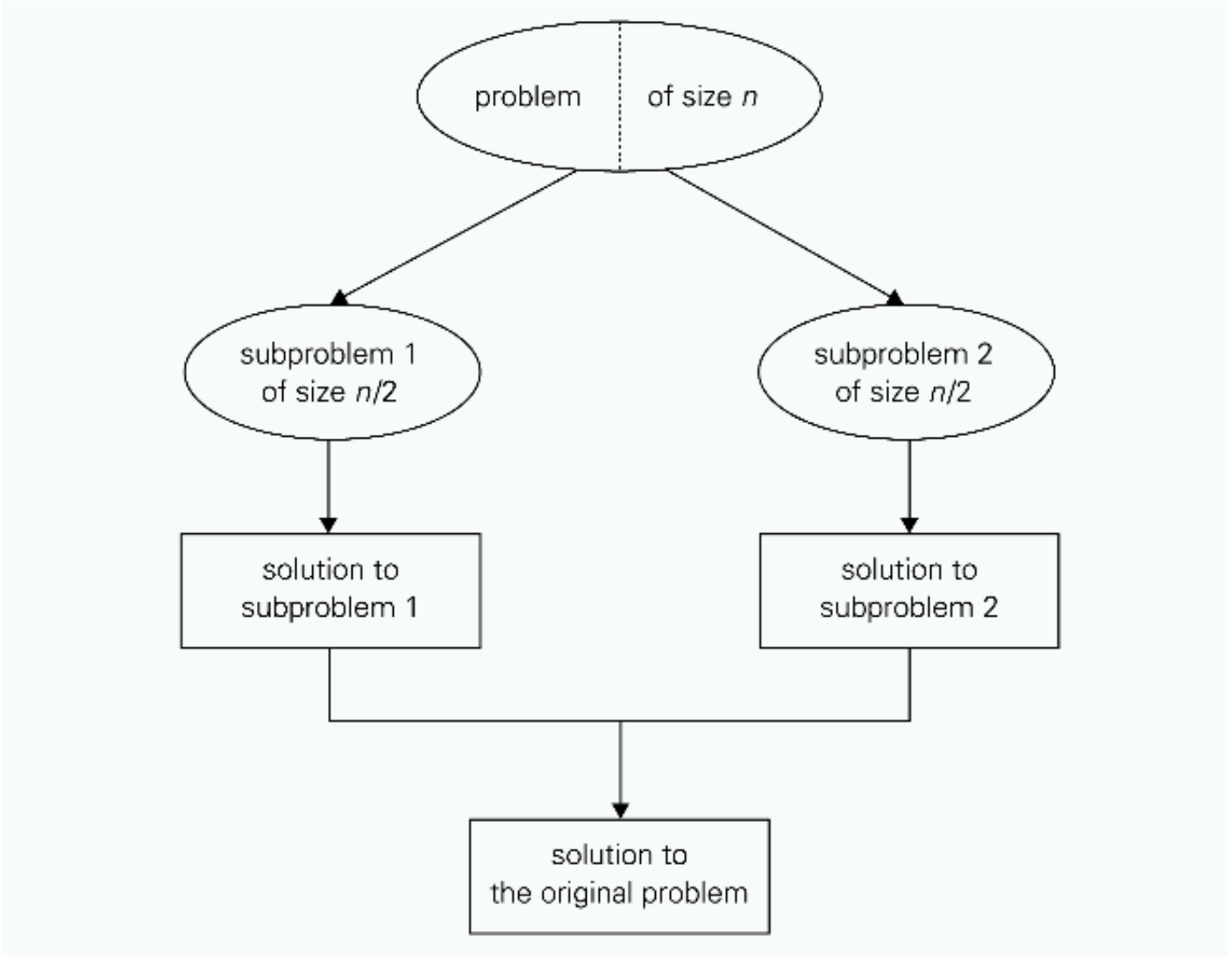
Divide and Conquer

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

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

Verdeel en heers

(vaak: verdelen in twee gelijke delen)



Decrease and Conquer

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

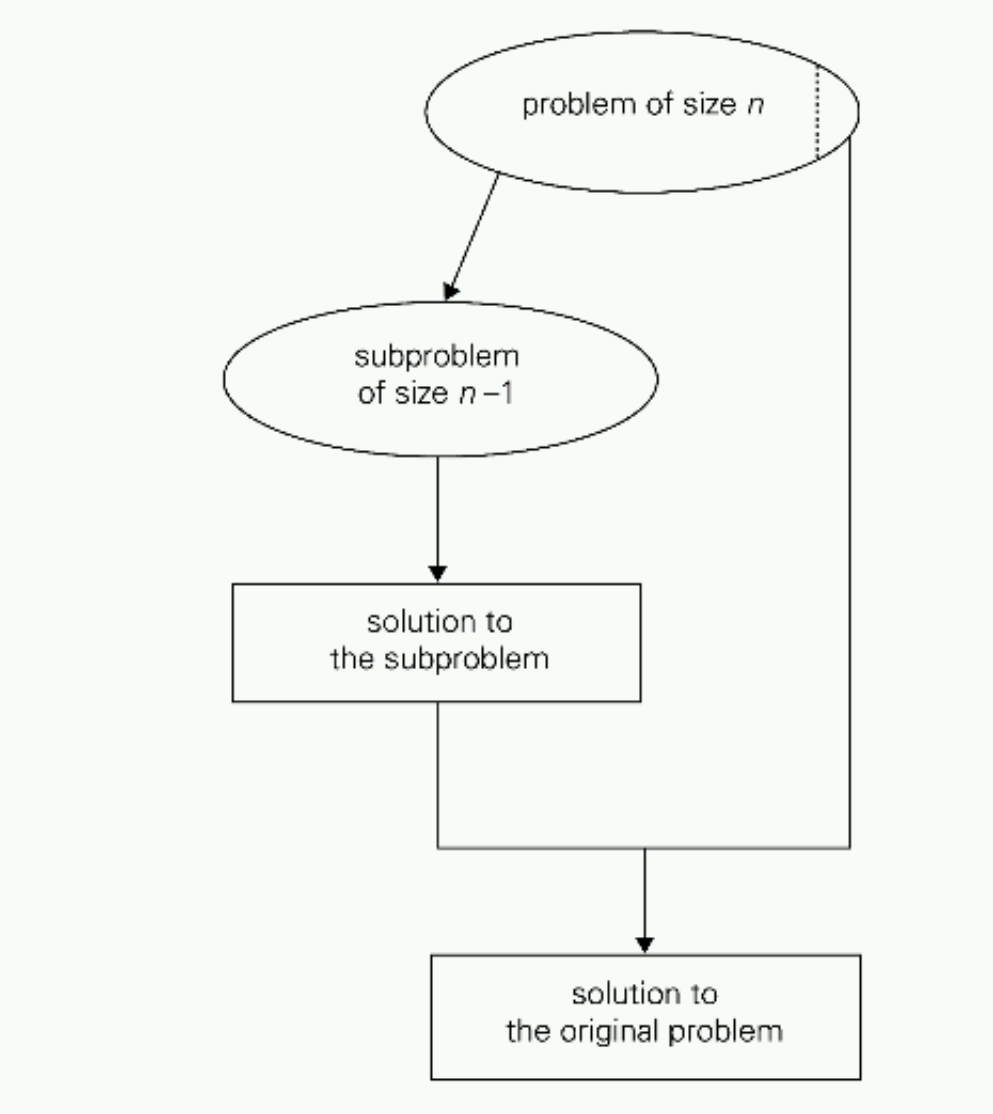
2. Los de kleinere instantie op: vaak recursief

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

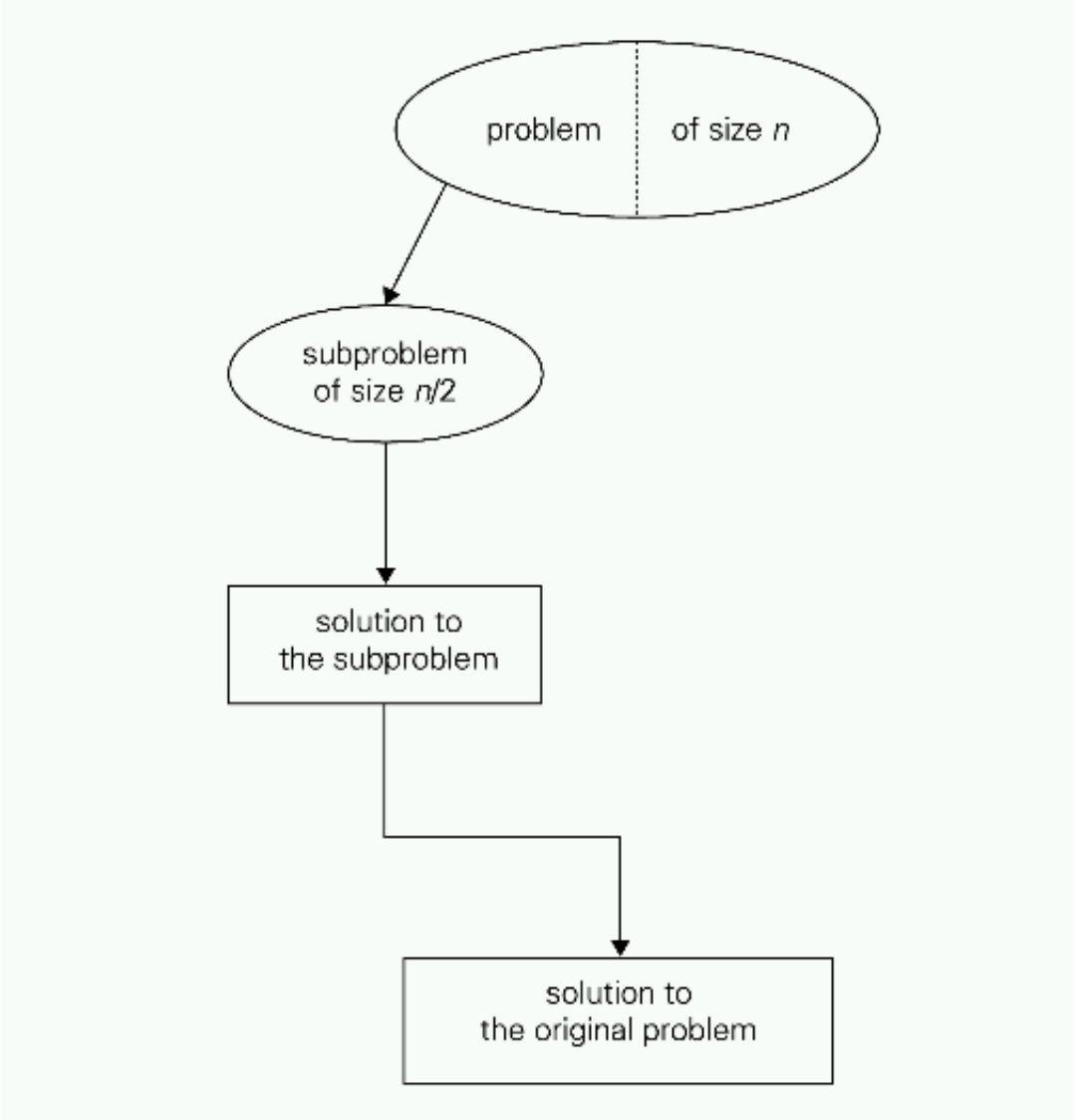
In het boek wordt onderscheid gemaakt tussen:

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

Decrease by one



Decrease by a constant factor
(decrease by half)



Vorige week bespraken we Mergesort, een sorteermethode die gebaseerd is op het principe van verdeel en heers. Vandaag Quicksort, eveneens gebaseerd op dit 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 .

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

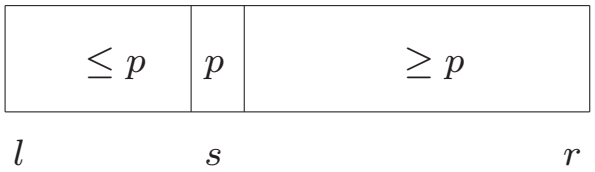
if

until $i \geq j$;

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

return j; .

Partitie



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.

```
i = 0; j = n-1;
while (i <= j) {
    if (A[i] < 0)
        i = i+1;
    else {
        wissel(A[i],A[j]);
        j = j-1;
    }
}
```

Variant (Dutch National Flag): gegeven een array met 'R', 'W' en 'B'. Reorganiseer het array zodat v.l.n.r. eerst alle 'R', dan de 'W' en dan de 'B' staan. Zie Levitin, opgave 5.2.9.a. Thuis over nadenken!

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: in situ

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 ($n = 3$):

$$876 * 543 =$$

$$(8 \cdot 10^2 + 7 \cdot 10^1 + 6 \cdot 10^0) * (5 \cdot 10^2 + 4 \cdot 10^1 + 3 \cdot 10^0) =$$

$$8 * 5 \cdot 10^4 + (8 * 4 + 7 * 5) \cdot 10^3 + (8 * 3 + 7 * 4 + 6 * 5) \cdot 10^2$$

$$+(7 * 3 + 6 * 4) \cdot 10^1 + 6 * 3 \cdot 10^0$$

Vermenigvuldiging van grote integers:

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 $n = 8$:

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

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

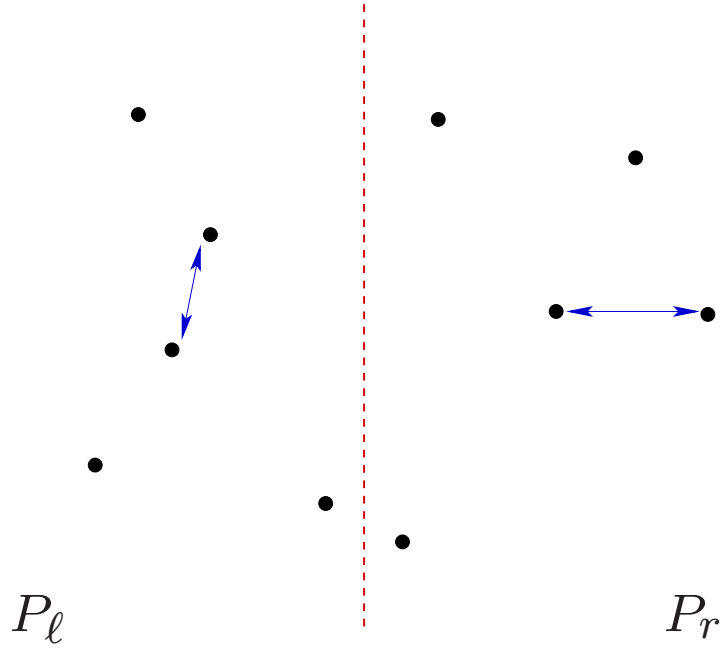
Het vermenigvuldigen van twee getallen bestaande uit $n = 2^k$ bits is zo teruggebracht tot 3 keer hetzelfde probleem voor $n/2 = 2^{k-1}$. Als $M(n)$ het aantal digitvermenigvuldigingen is voor $n = 2^k$, dan voldoet $M(n)$ aan:

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

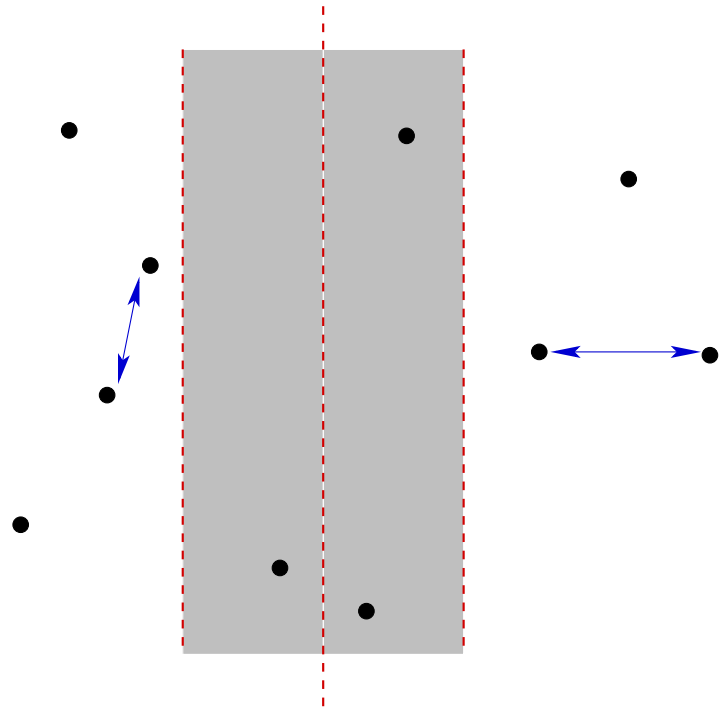
en vinden we: $M(n) = n^{\lg 3} < n^{\lg 4} = n^2$.

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



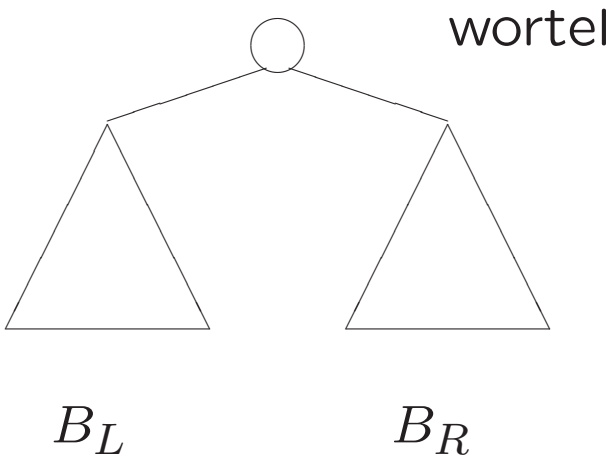
Closest Pair: conquer



Controleer of er binnen de strip ter breedte $2d$ rondom de scheidslijn tussen P_ℓ en P_r puntenparen (p, p') zijn met $p \in P_\ell$ en $p' \in P_r$ en onderlinge afstand kleiner dan d. Hiervoor blijken slechts $O(n)$ puntenparen bekeken te moeten worden. Dit levert een $O(n \lg n)$ 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 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 == NULL )       // lege boom
    return -1;
  else
    return (1+max(hoogte(root->links),hoogte(root->rechts)));
}  // hoogte
```

Zie verder college 2 en bijbehorende werkcollege.

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

if $m > 1$

Insertionsort($A[0 \cdots m-2]$); DECREASE by one

Voeg $A[m-1]$ op de juiste plek in; & CONQUER

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):

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

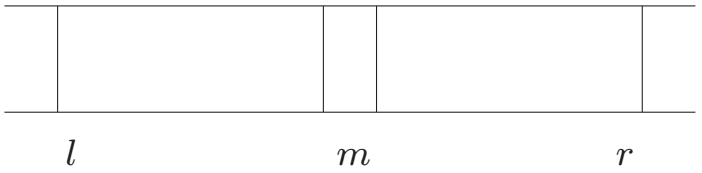
kleiner of gelijk $A[m-1]$ ↑ groter dan $A[m-1]$

hier invoegen

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)

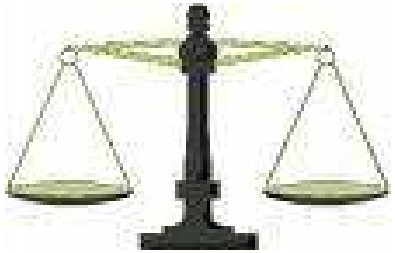
$l := 0; r := n-1;$
while $l \leq r$ **do**
  $m := \lfloor \frac{l+r}{2} \rfloor;$
  if $K = A[m]$ **then** // gevonden
    return m;
  else if $K < A[m]$ **then** // links verder zoeken
    $r = m - 1;$
  else // rechts verder zoeken
    $l = m + 1;$ **fi**
  fi
od
return -1;



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.



Valse munt: oplossing

Decrease by a constant factor (hier decrease-by-half): 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: $W(n) = \lfloor \lg n \rfloor$. (notatie: $\lg n = \log_2 n$)

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:

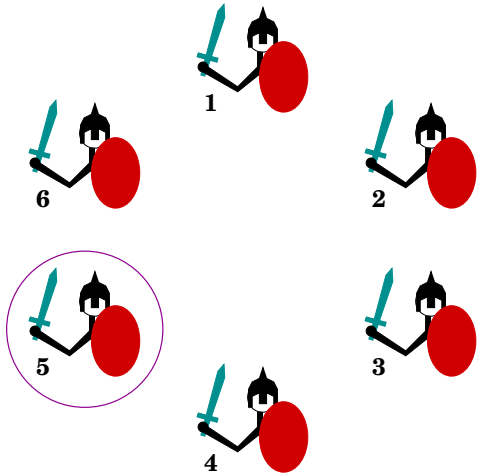
$$n * m = \frac{n}{2} * 2m \quad \texttt{als } n \texttt{ even}$$

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

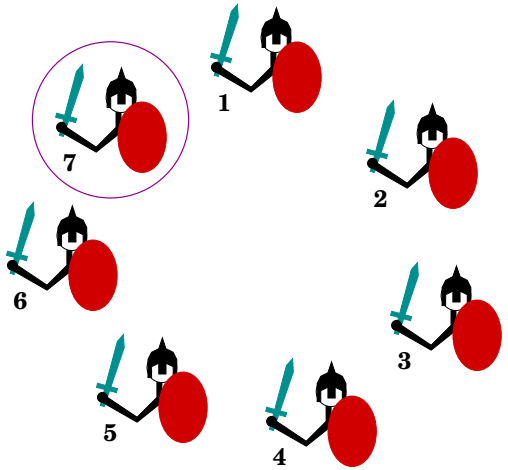
$$1 * m = m$$

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

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 4.1, 4.4, 5.2-5

- **Werkcollege:**

  donderdag 3 april 2014 in zaal B2/B3:
  backtracking en verdeel & heers

- **Opgaven:**

  zie http://www.liacs.nl/home/graaf/ALGO/

- **Volgend college:**

  vrijdag 4 april 2014 in zaal B2:
  variable-size decrease en dynamisch programmeren