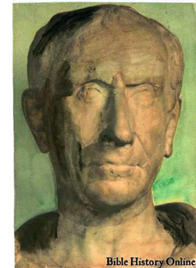
Zevende college algoritmiek

4 april 2013

Verdeel & Heers



Divide and Conquer

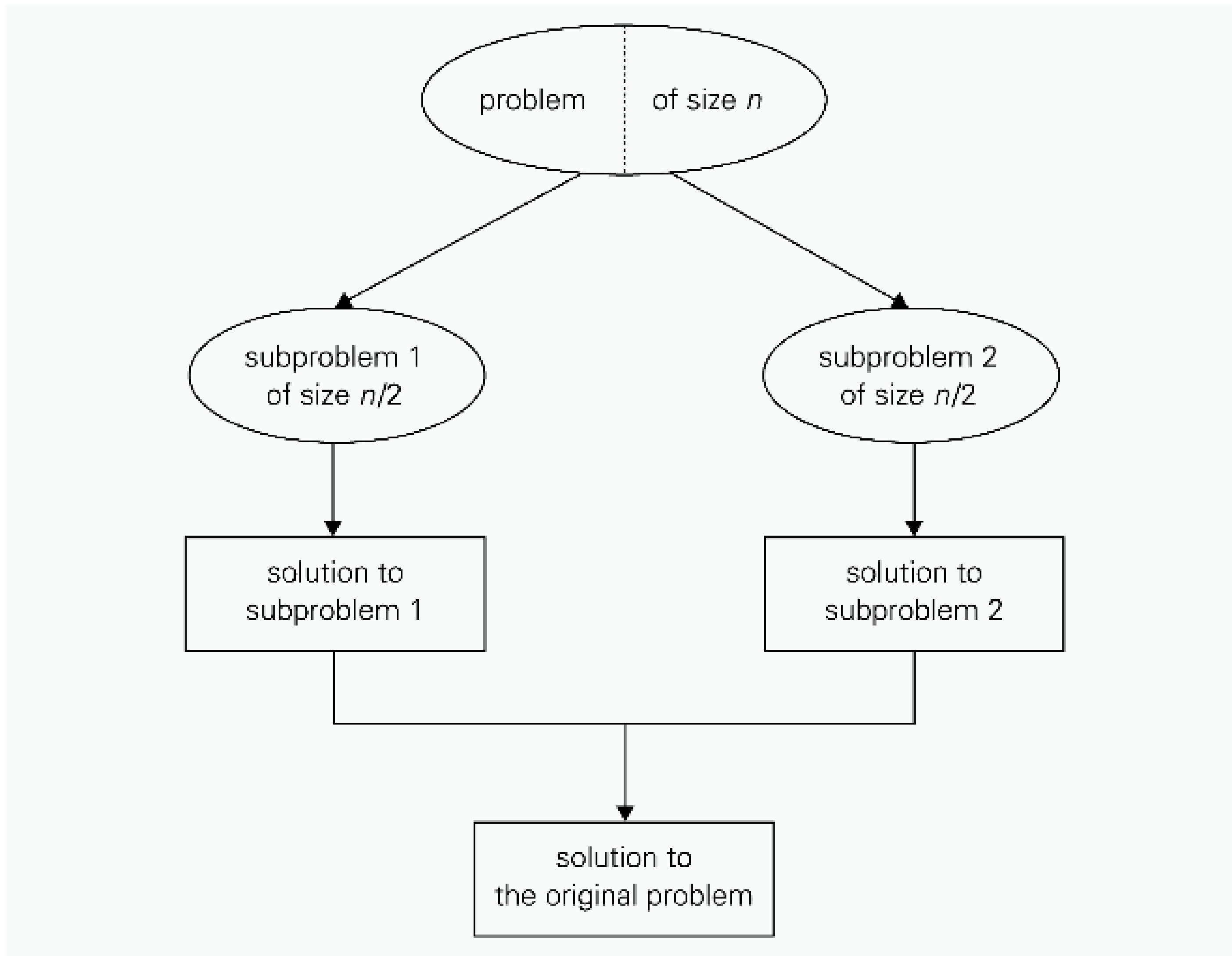
1. Verdeel een instantie van het probleem in twee of meer kleinere instanties van hetzelfde probleem

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: verdeel in twee gelijke delen)



Decrease and Conquer

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

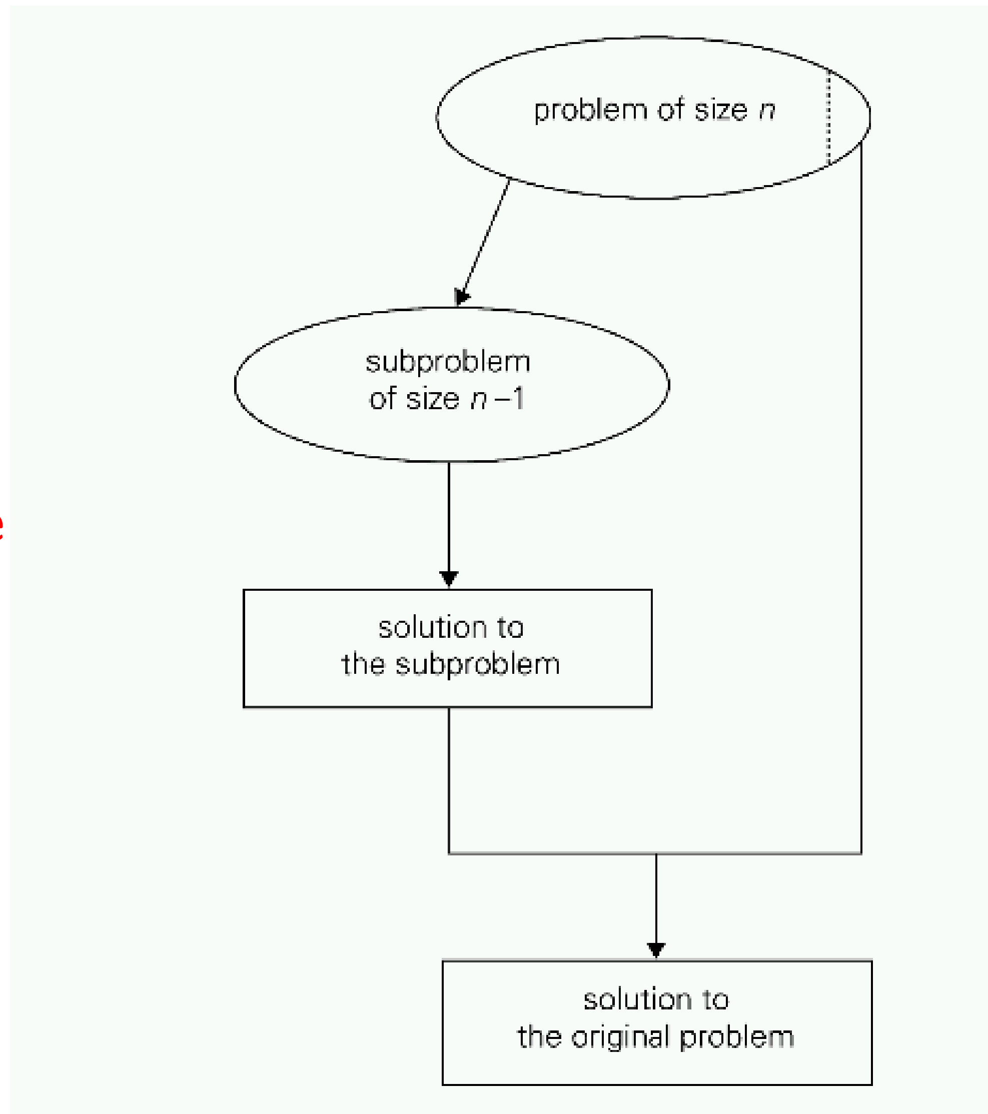
2. Los de kleinere instantie op: vaak recursief

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

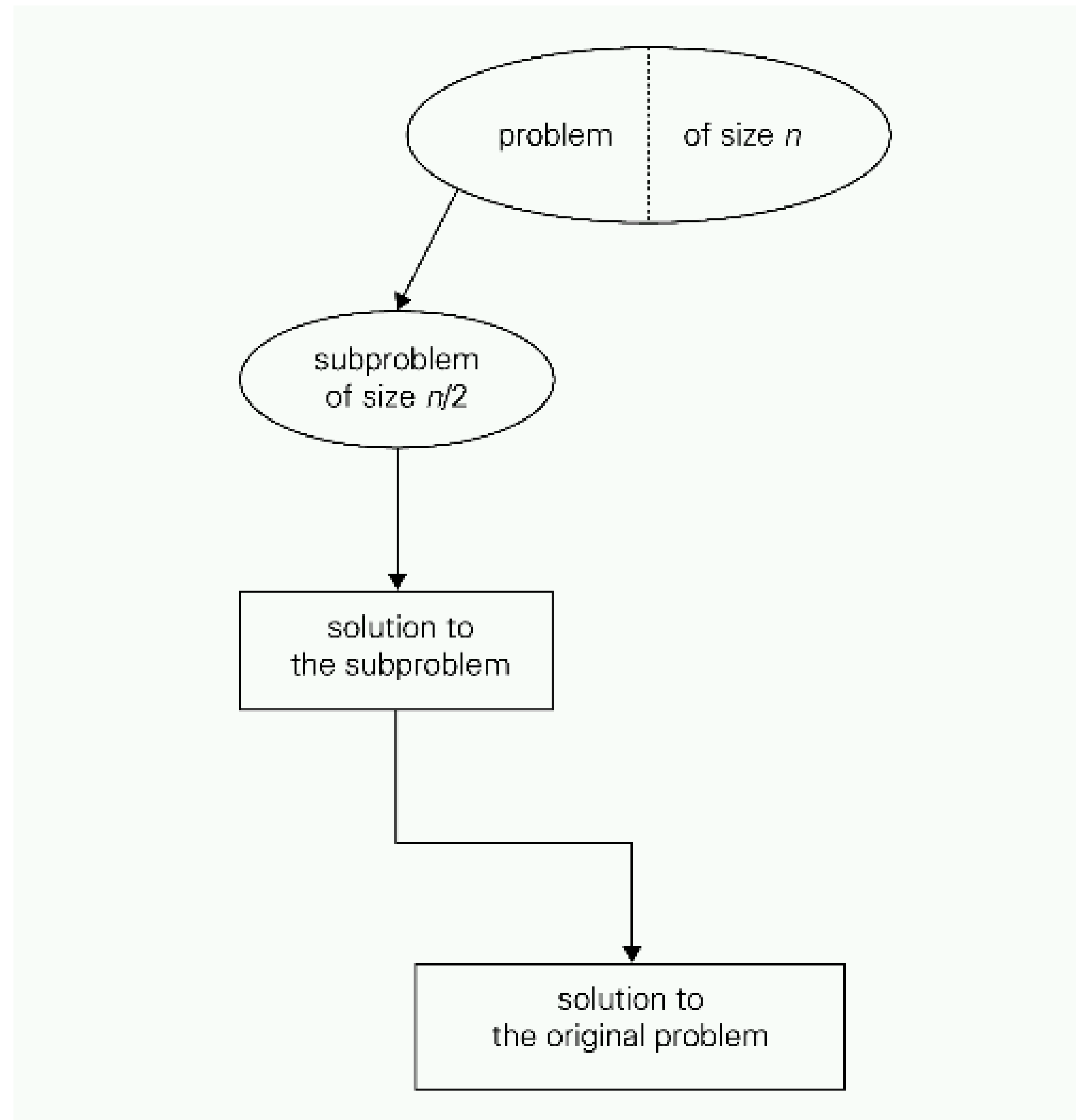
In het boek wordt onderscheid gemaakt tussen:

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

Decrease by one

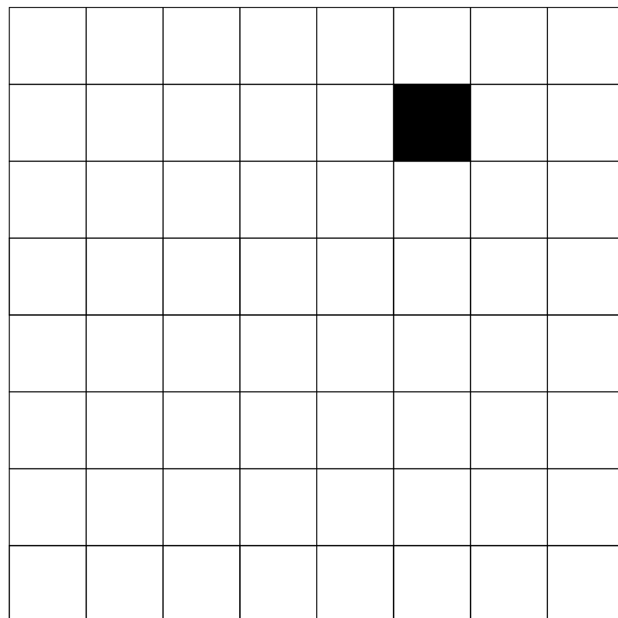


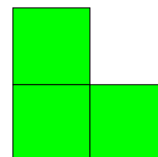
Decrease by a constant factor
(decrease by half)



Nog een leuk voorbeeld van de methode Verdeel en heers is de Tromino puzzel, Levitin 5.1.11.

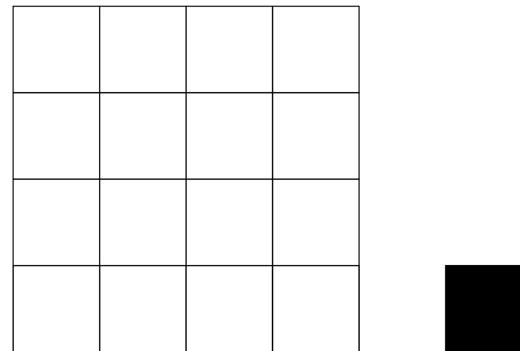
Bedek een 2^nx2^n schaakbord –waaruit één vakje mist– met tromino's.

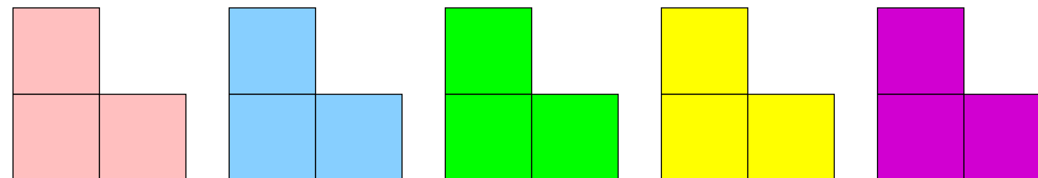




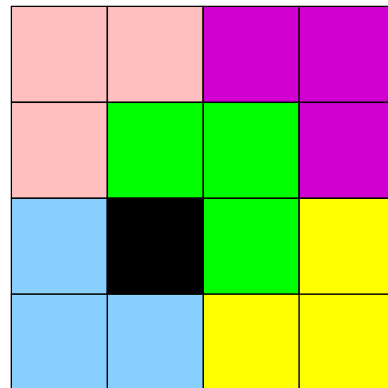
Het 8x8 geval

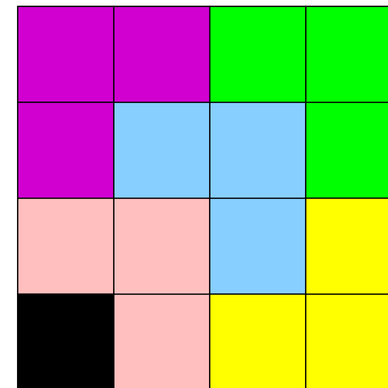
Het 4x4 geval: gegeven een 4x4 bord waaruit één vakje mist. Bedek het bord volledig (behalve het missende vakje), dus met 5 tromino's.





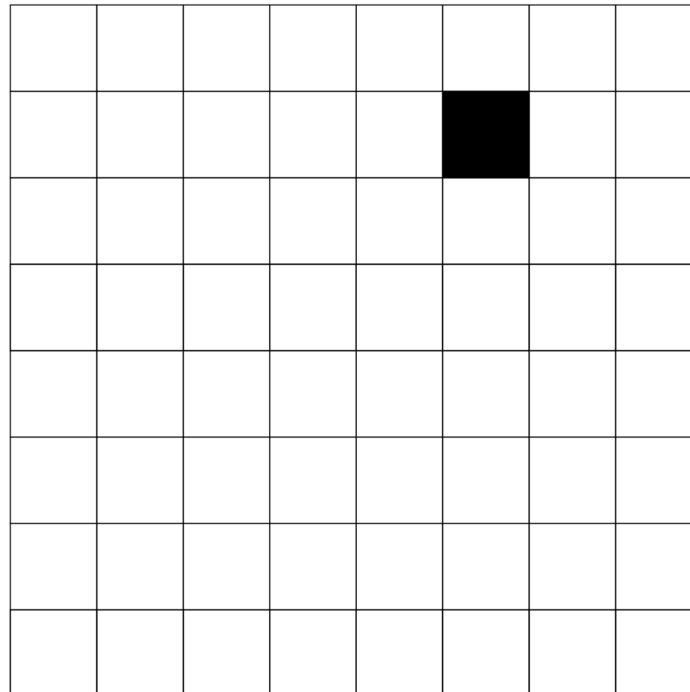
Het 4x4 geval: twee probleeminstanties met oplossing (= bedekking)



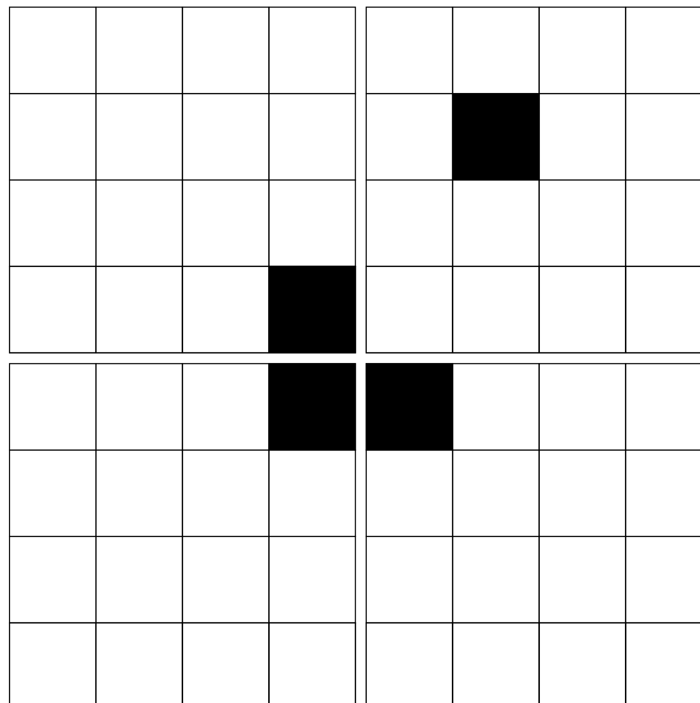


Het 8x8 geval: hoe vinden we een (de?) bedekking met tromino's?

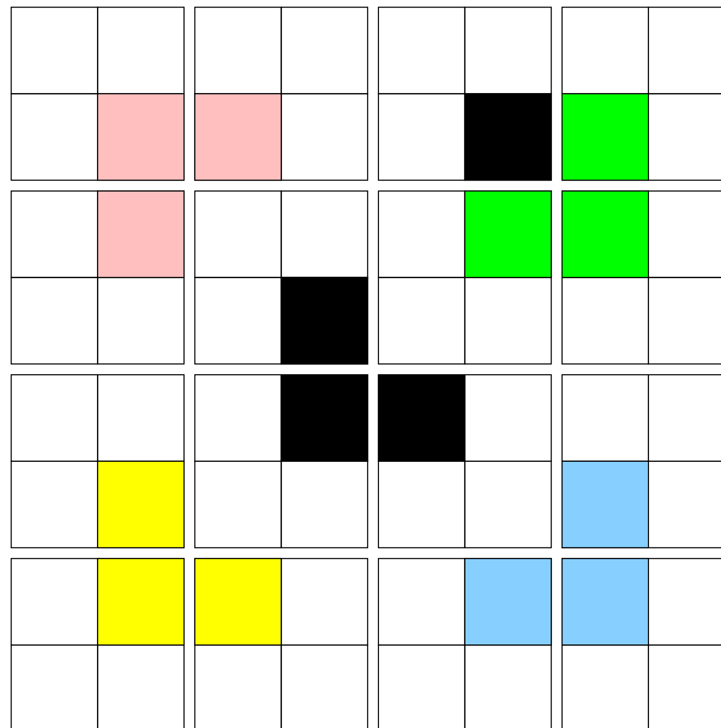
Bijvoorbeeld voor dit bord:



Leg een tromino in het midden, zodat in elk kwart één stuk mist of bedekt is. Het probleem is nu teruggebracht tot vier keer hetzelfde probleem, maar dan voor 4x4 borden.



Doe weer hetzelfde (recursie!) met de 4x4 borden.



Dit divide and conquer algoritme kun je op elk 2^nx2^n bord toepassen.

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.

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 $n = 8$:

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

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

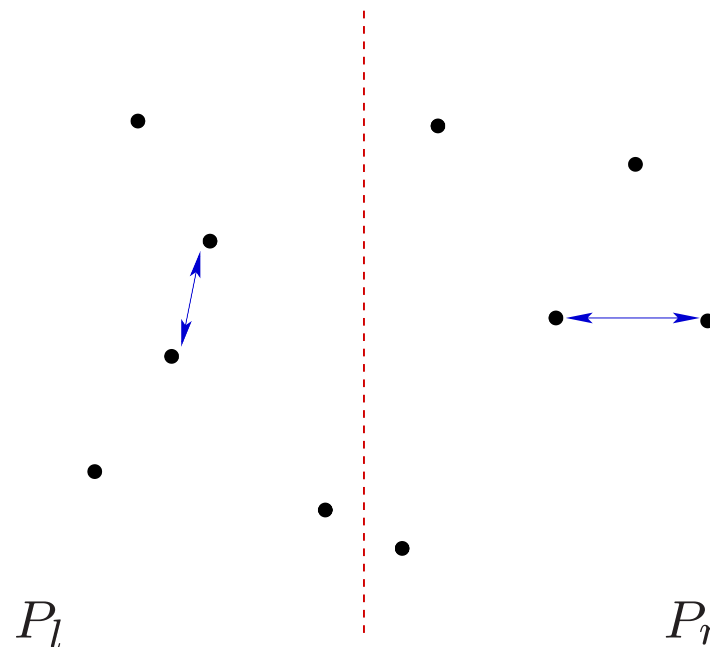
Het vermenigvuldigen van twee getallen bestaande uit $n = 2^k$ bits is zo teruggebracht tot 3 keer hetzelfde probleem voor $n/2 = 2^{k-1}$. Als $M(n)$ het aantal digitvermenigvuldigingen is voor $n = 2^k$, dan voldoet $M(n)$ aan:

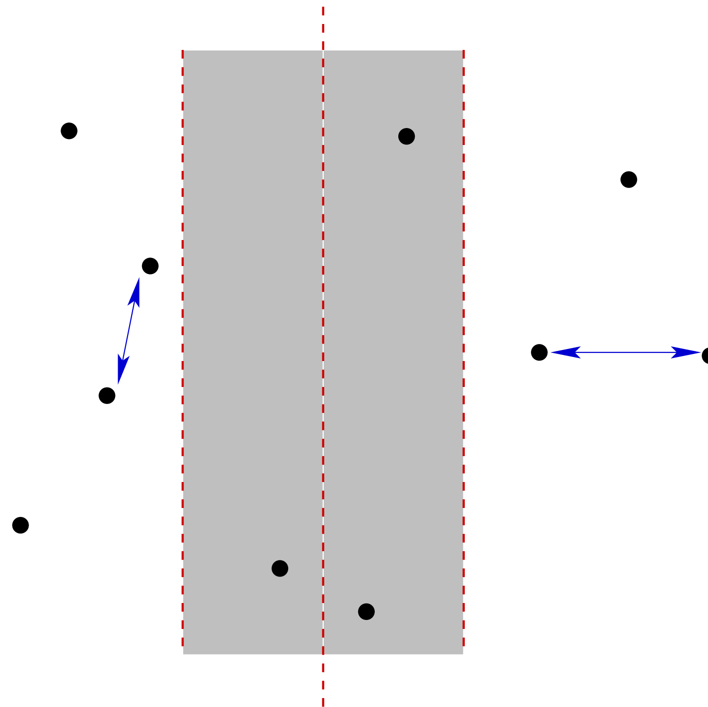
$$M(n) = 3 * M(n/2) \text{ als } n > 1; M(1) = 1,$$

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

Divide and Conquer:

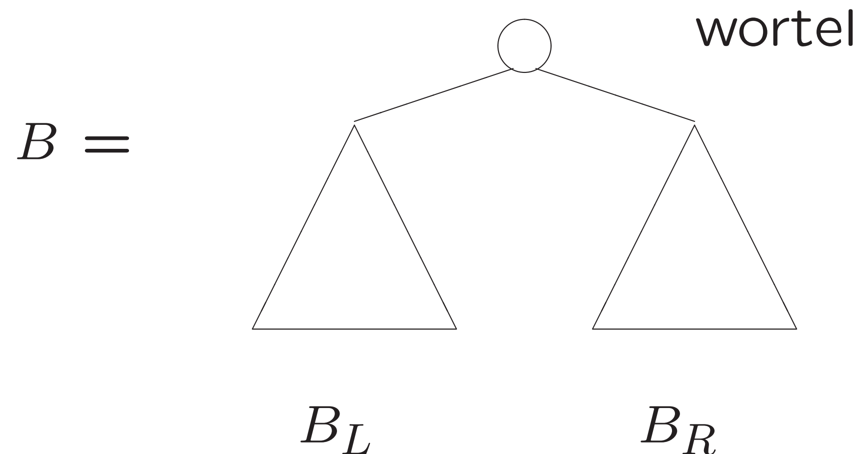
Verdeel de verzameling van n punten in twee verzamelingen P_l en P_r van elk $\frac{n}{2}$ punten door een geschikte lijn te trekken. Los beide deelproblemen (recursief) op en laat d de kleinst voorkomende afstand zijn tussen punten van P_l resp. P_r.





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.



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
```

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):

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

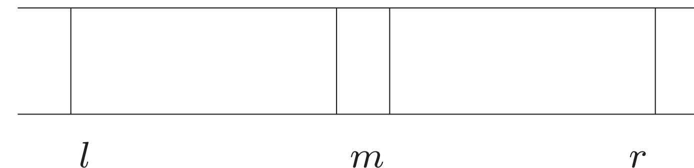
kleiner of gelijk $A[m-1]$ $\uparrow$ groter dan $A[m-1]$

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)

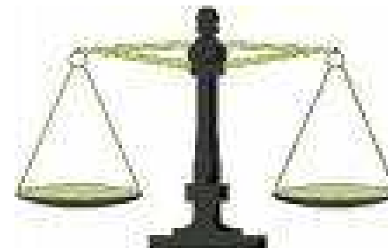
```
l := 0; r := n − 1;
while l ≤ r do
    m := ⌊(l+r)/2⌋;
    if K = A[m] then // gevonden
        return m;
    else if K < A[m] then // links verder zoeken
        r = m − 1;
    else // rechts verder zoeken
        l = m + 1; fi
    fi
od
return −1;
```



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.

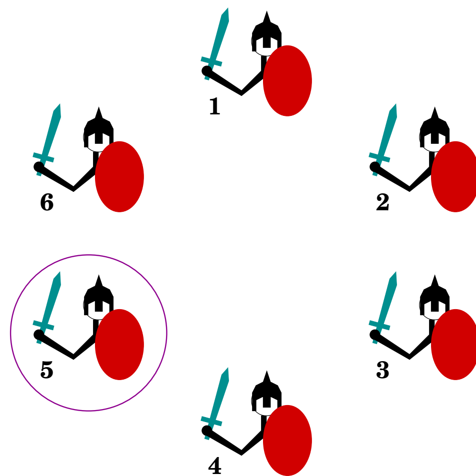
Recurrente betrekking voor het aantal wegingen dat nodig is in de worst case om de valse munt te ontdekken:

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

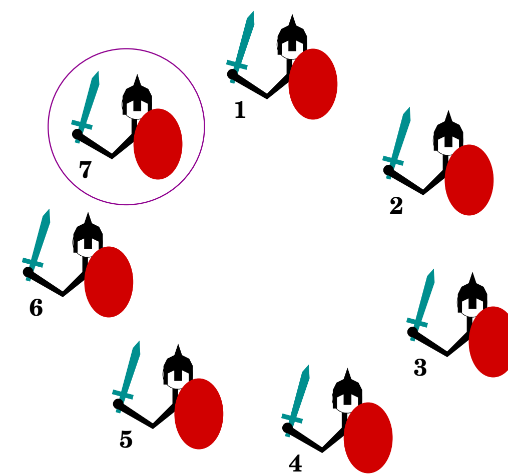
Oplossing: $W(n) = \lfloor \lg n \rfloor$.

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

- **Lezen/leren bij dit college:**

  Paragrafen 4.1, 4.4, 5.3-5.5

- **Werkcollege:**

  donderdag 4 april 2013, 13:45–15:30, in zaal Paleistuin:
  tweede programmeeropdracht (backtracking)

- **Volgend college:**

  donderdag 11 april 2013