Zevende college Algoritmiek

april 2020

Verdeel en Heers

```
4
Feyenoord Rotterdam
AZ Alkmaar
PSV Eindhoven
Ajax Amsterdam
```

```
construeerschema(...):

Ronde 1:
 Feyenoord AZ
 PSV Ajax
Ronde 2:
 AZ PSV
 Ajax Feyenoord
Ronde 3:
 Feyenoord PSV
 AZ Ajax
```

```
construeerschema(...):
Ronde 1:
 Feyenoord - Feyenoord
 Feyenoord - Feyenoord
Ronde 2:
 Feyenoord - Feyenoord
 Feyenoord - Feyenoord
Ronde 3:
 Feyenoord - Feyenoord
 Feyenoord - Feyenoord
```

Eisen:

- elke club ($\leq$?) één keer per ronde
- iedere club één keer tegen iedere andere club
- $|\text{uit} - \text{thuis}| \leq 1$, na iedere ronde
- $\leq$ één thuiswedstrijd per stad per ronde
- ...

Backtracken:

```
Ronde 1:
 Feyenoord AZ
 PSV Ajax
Ronde 2:
 AZ PSV
 Ajax Feyenoord
Ronde 3:
 Feyenoord PSV
 AZ Ajax
```

Mogelijke implementatie: array met indices van de clubs, voor alle rondes:

0 1 2 3 1 2 3 0 0 2 1 3

Verdeel en Heers

Divide and Conquer

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

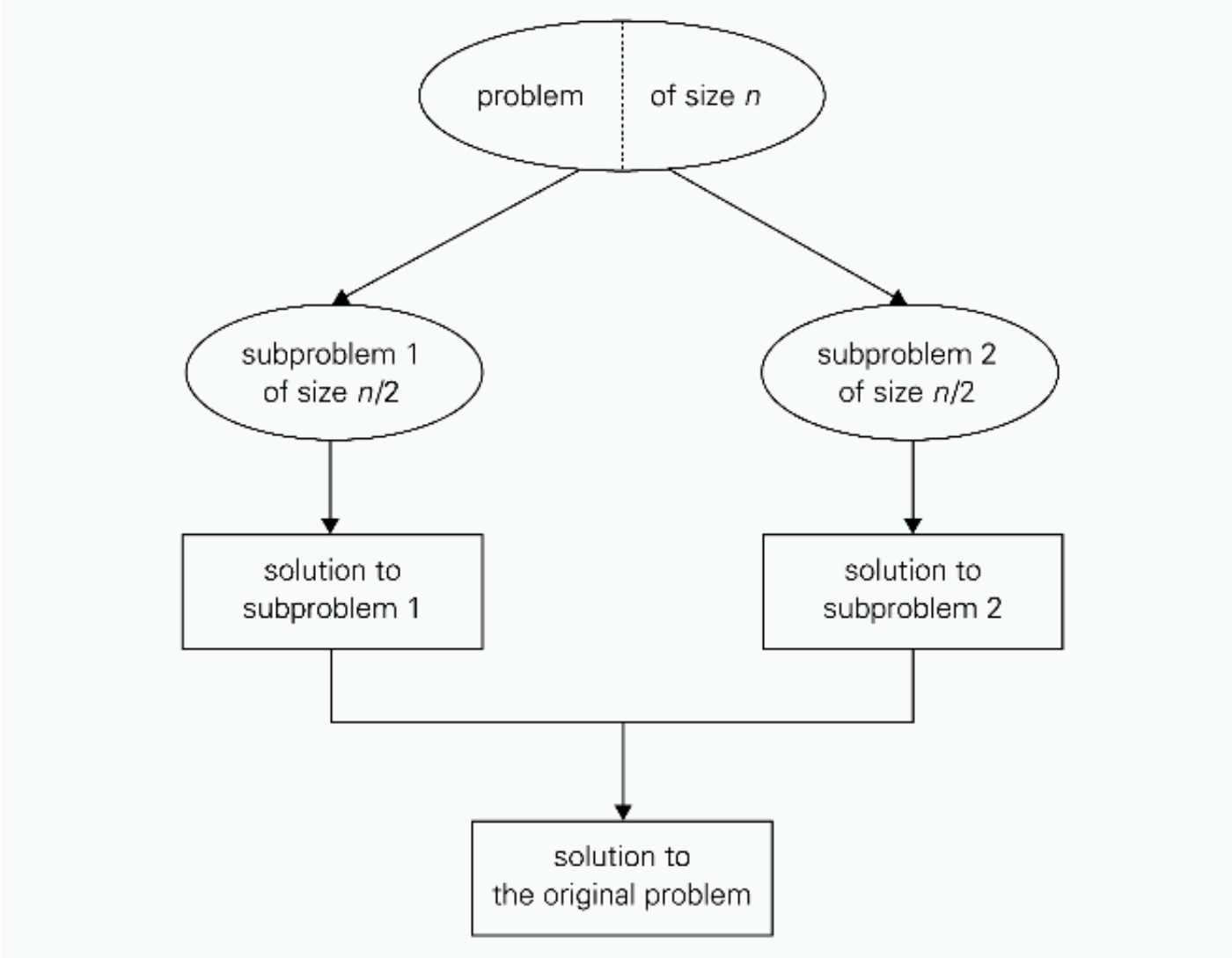
2. Los de kleinere instanties op: meestal recursief

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.

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

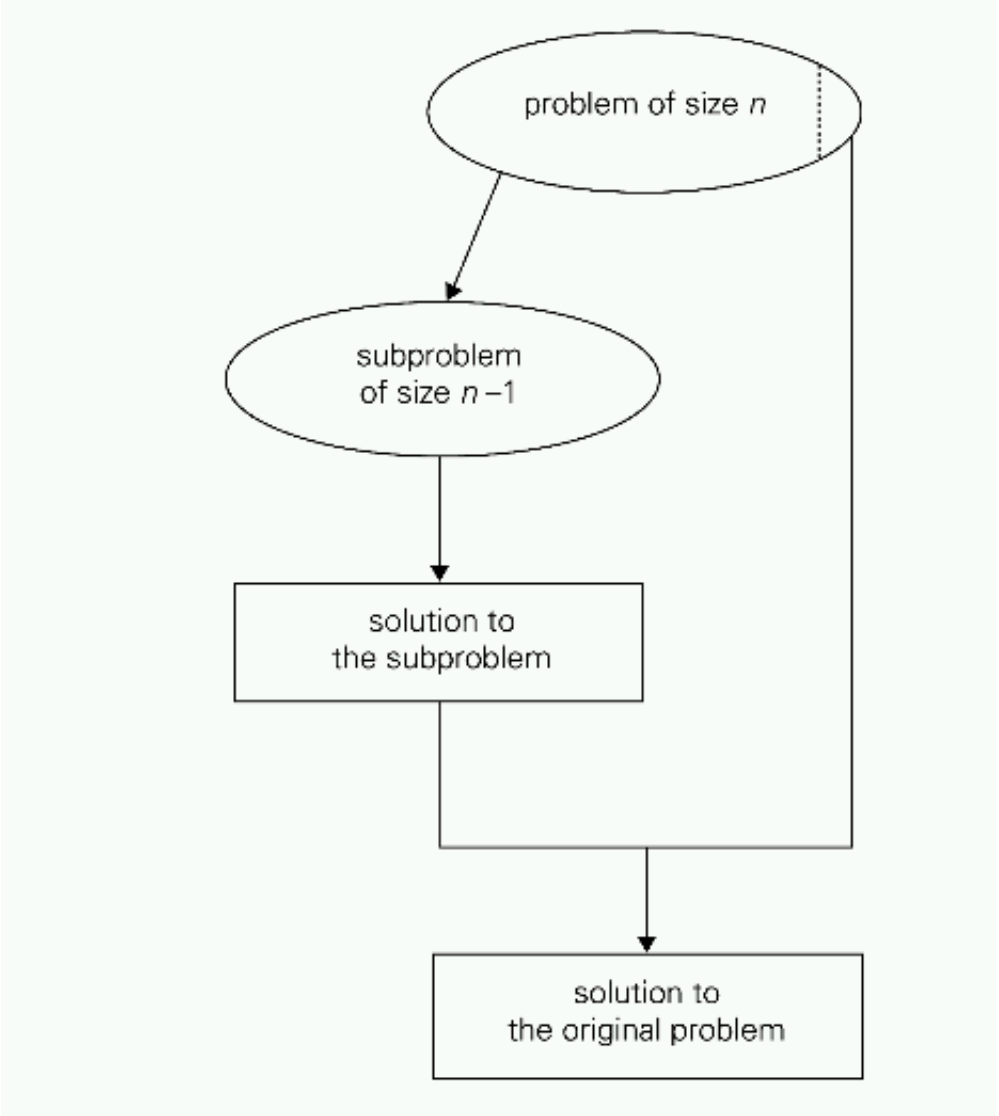
2. Los de kleinere instantie op: vaak recursief

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

In het boek wordt onderscheid gemaakt tussen:

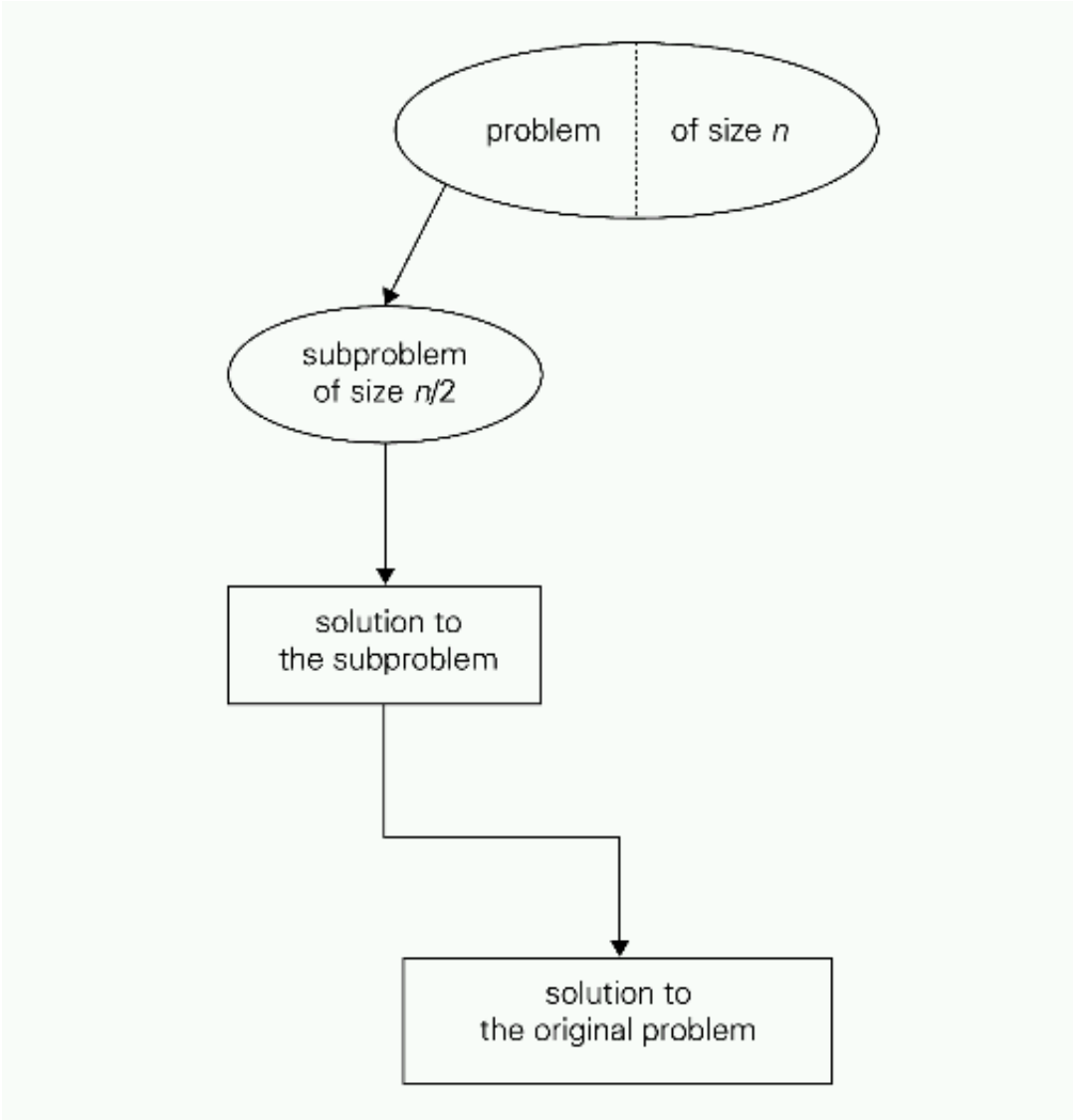
Decrease by one
Decrease by a constant factor
Variable-size decrease

Decrease by one



Decrease by a constant factor

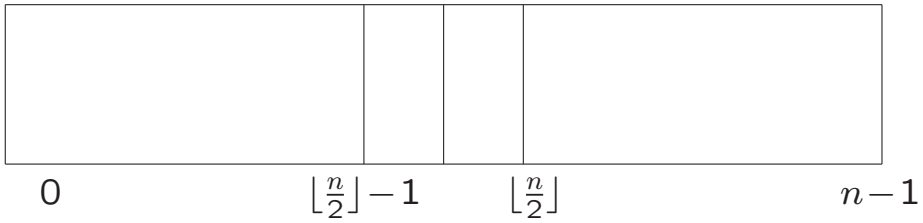
(decrease by half)



Verdeel en heers en sorteren:

```
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 .
```

Divide and conquer



Mergesort

Quicksort

Decrease and conquer (decrease by one)

Insertion sort

Mergesort($A[0 \cdots n-1]$)::
    // sorteert het array $A[0..n-1]$ recursief
    // uitvoer: $A[0..n-1]$ oplopend gesorteerd
    if $n > 1$
        copieer($A[0 \cdots \lfloor \frac{n}{2} \rfloor - 1], B[0 \cdots \lfloor \frac{n}{2} \rfloor - 1]$);
        copieer($A[\lfloor \frac{n}{2} \rfloor \cdots n-1], C[0 \cdots \lceil \frac{n}{2} \rceil - 1]$);
        Mergesort($B[0 \cdots \lfloor \frac{n}{2} \rfloor - 1]$);
        Mergesort($C[0 \cdots \lceil \frac{n}{2} \rceil - 1]$);
        Merge(B, C, A);
    fi .

Voorbeeld: 8 3 2 9 7 1 5 4

Merge($B[0 \cdots p-1], C[0 \cdots q-1], A[0 \cdots p+q-1]$) ::
// voegt 2 gesorteerde arrays B en C samen tot 1 gesorteerd array A
$i, j, k := 0$;
// voeg samen totdat een van de twee op is: ritsen
while $i < p$ **and** $j < q$ **do**
if $B[i] \leq C[j]$ **then**
$A[k] := B[i]; k := k+1; i := i+1;$
else
$A[k] := C[j]; k := k+1; j := j+1;$
od
// en de rest
if $i = p$ **then**
copieer $C[j \cdots q-1]$ naar $A[k \cdots p+q-1]$;
else
copieer $B[i \cdots p-1]$ naar $A[k \cdots p+q-1]$;
fi .

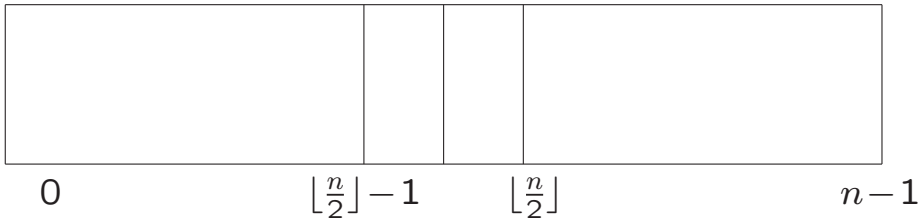
Mergesort:

- worst case complexiteit: . . .
- extra geheugen: . . .

Mergesort:

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

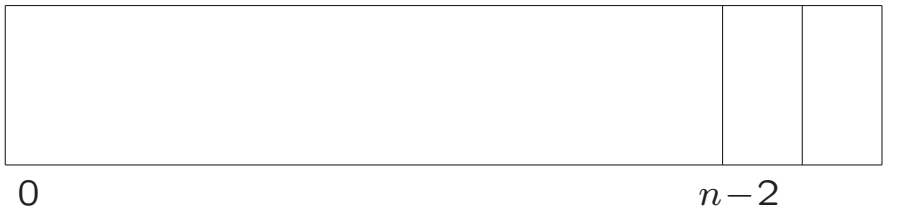
Divide and conquer



Mergesort

Quicksort

Decrease and conquer (decrease by one)



Insertion sort

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**

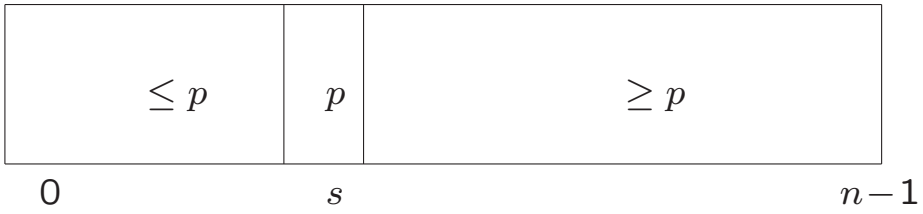
Wissel($A[i], A[j]$);

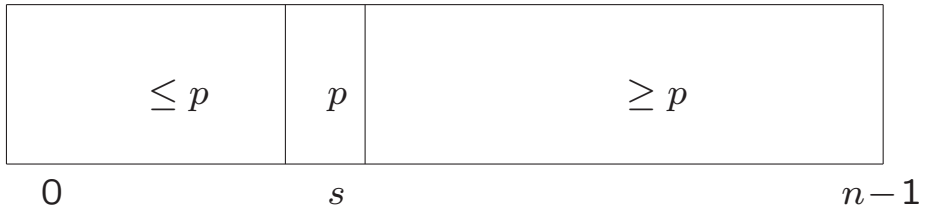
if

until $i \geq j$;

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

return j; .





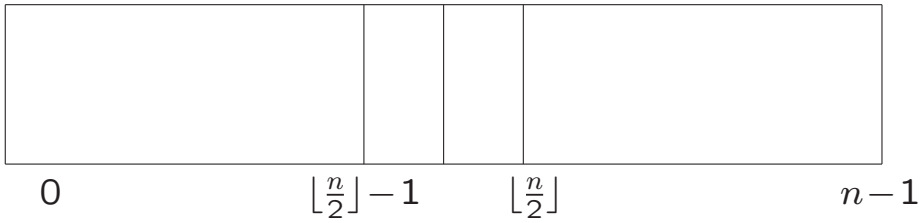
Quicksort

- ruimte complexiteit: . . .
- worst case tijd: . . .
- average case tijd: . . .

Quicksort

- ruimte complexiteit: $O(\log n)$
- worst case tijd: $\Theta(n^2)$
- average case tijd: $\Theta(n \log n)$

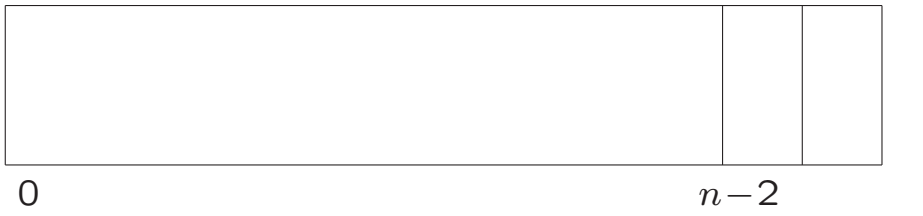
Divide and conquer



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 .

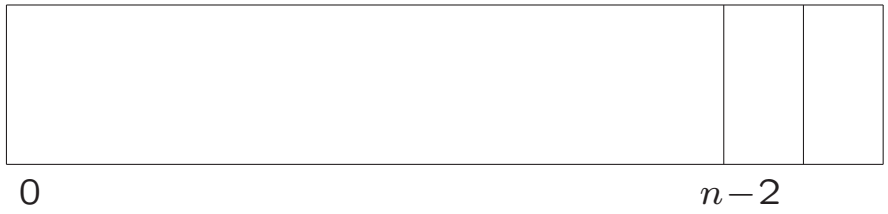
Invoegen van $A[m-1]$ in het reeds gesorteerde voorstuk $A[0] \cdots A[m-2]$ door van rechts naar links $A[m-1]$ te vergelijken met $A[i]$. Deze recursieve versie komt overeen met de iteratieve versie zoals bij Programmeermethoden behandeld (zie ook Levitin):

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

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

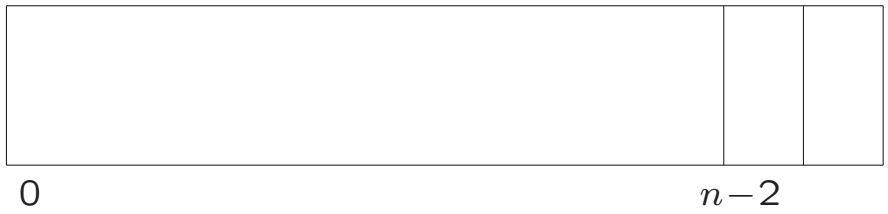
hier invoegen

Voorbeeld: 5 3 1 9 8 2 4 7



Insertion sort

- ruimte complexiteit: . . .
- worst case tijd: . . .
- average case tijd: . . .
- best case tijd: . . .



Insertion sort

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

Mergesort:

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

Quicksort:

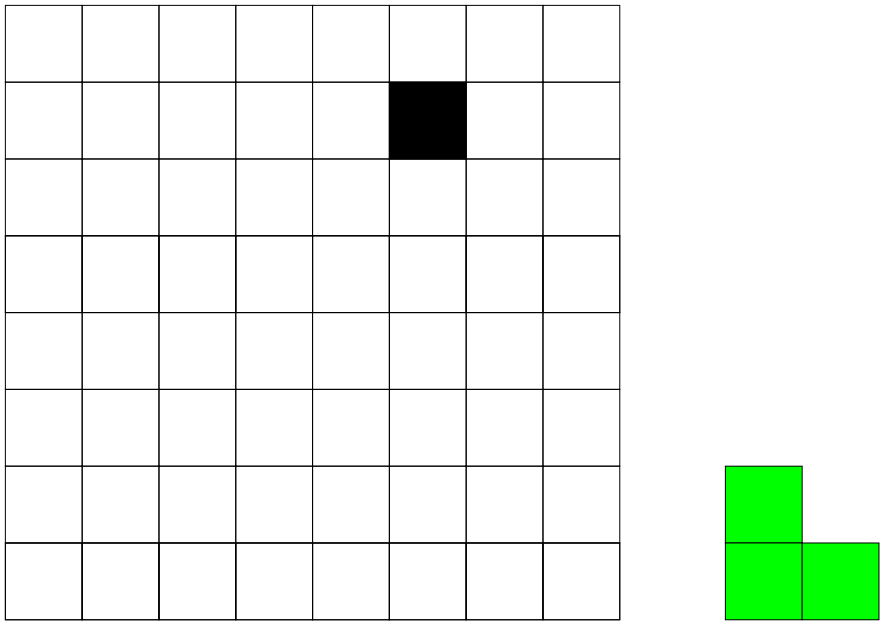
- worst case complexiteit: $\Theta(n^2)$ voor (o.a.) het reeds gesorteerde rijtje
- average case complexiteit: $\Theta(n \log n)$
- extra geheugen: $O(\log n)$

Insertion sort:

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

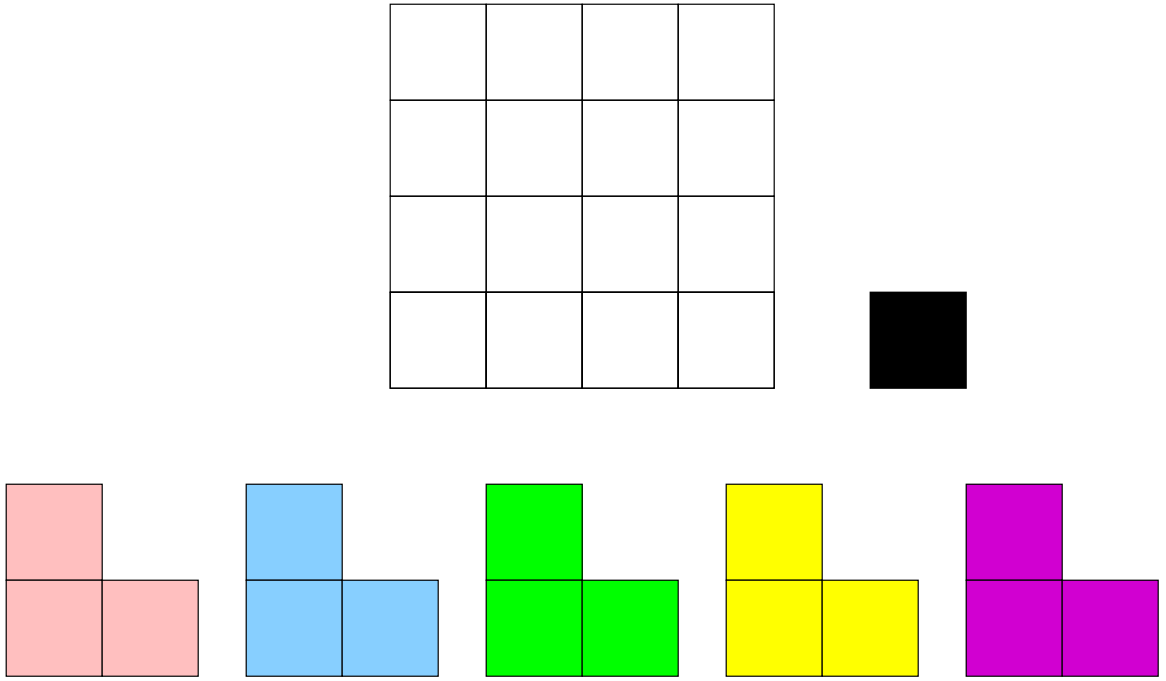
Nog een leuk voorbeeld van de methode Verdeel en heers is de Tromino puzzel, Levitin 5.1.11.

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

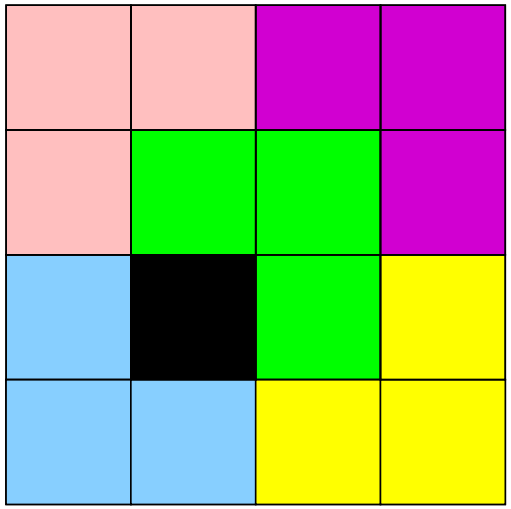


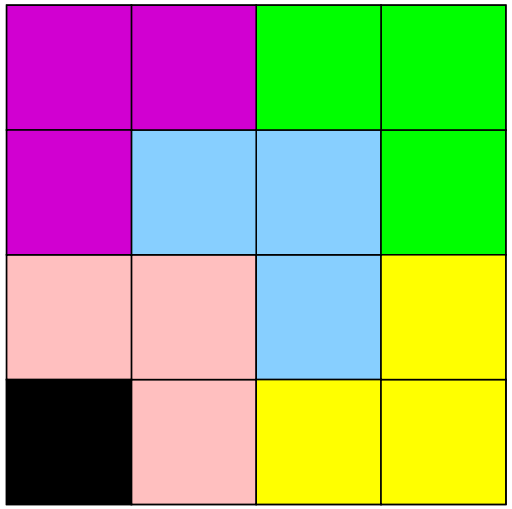
Het 8x8 geval

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



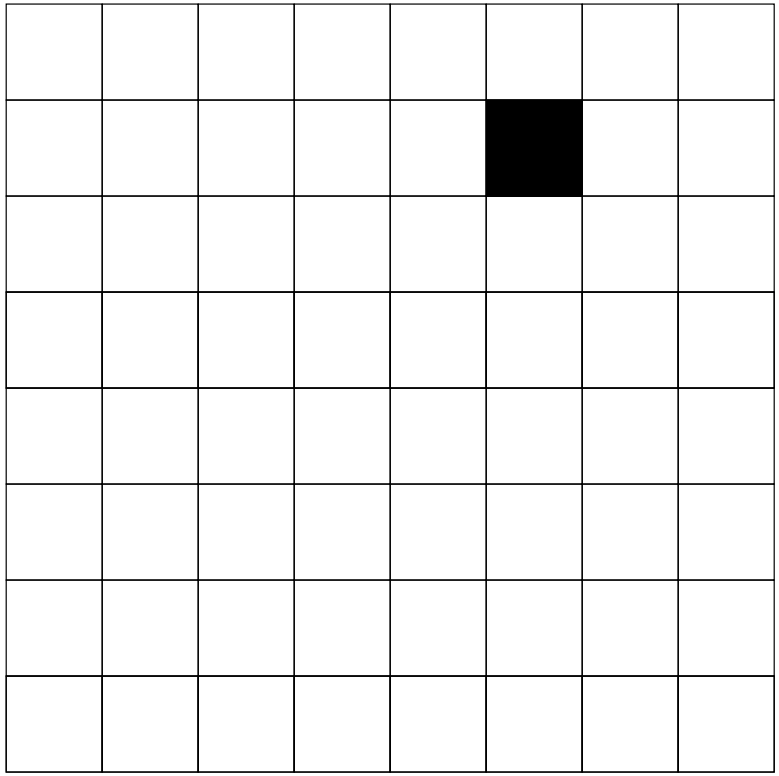
Het 4x4 geval: twee probleeminstanties met oplossing (= bedekking)



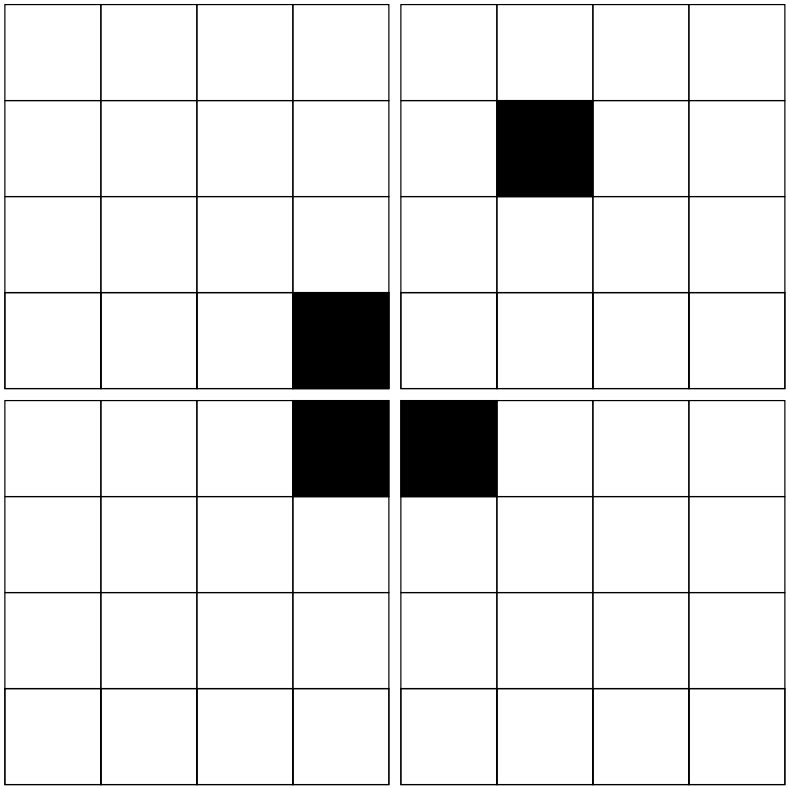


Het 8x8 geval: hoe vinden we een (de?) bedekking met tromino's?

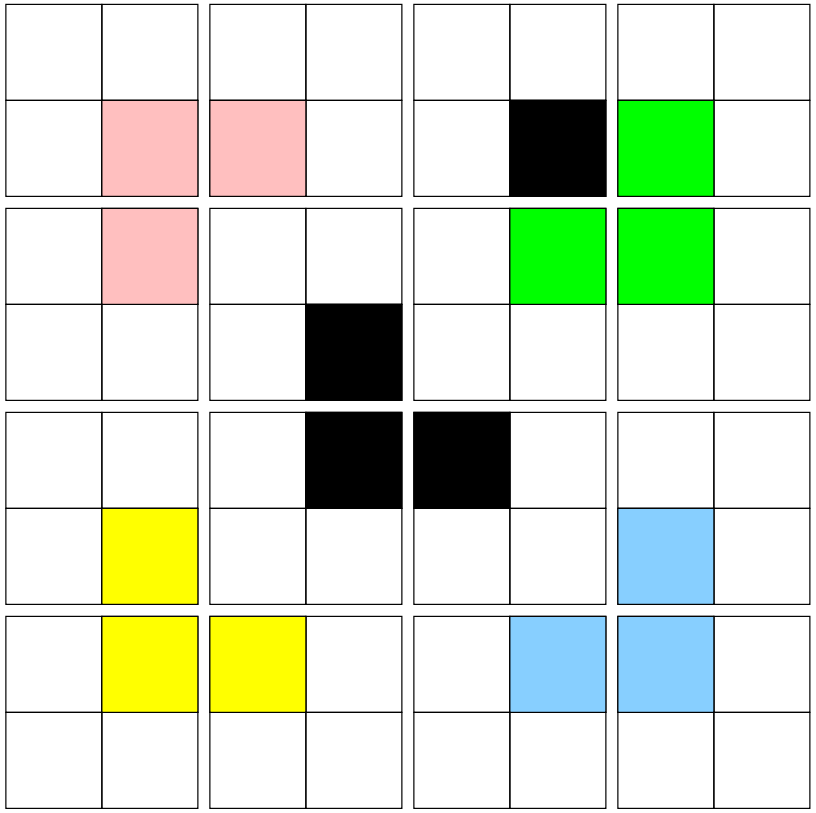
Bijvoorbeeld voor dit bord:



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 $n = 8$:

$$87593264 * 49367251 =$$

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

$$c_2 = 8759 * 4936$$

$$c_0 = 3264 * 7251$$

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

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

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

- **Lezen/leren bij dit college:**
  Paragraaf 5 inl., 5.1, 5.2, 5.4, 5.5 (geen Strassen, convex hull),
  4 inl., 4.1

- **Werkcollege:** verdeel en heers
  ook online

- **Volgend college:** Verdeel en Heers / Dynamisch Programmeren

- **Deadline opdracht 1:** . . .

- **Opdracht 2:** . . .