Achtste college algoritmiek

30 maart 2026

Verdeel en Heers

- dinsdag 31 maart: geen werkcollege

- Programmeeropdracht 1
  - deadline: 20 april 2026, 23.59 uur
  - woensdag 1 april: practicumbijeenkomst

Divide and Conquer

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

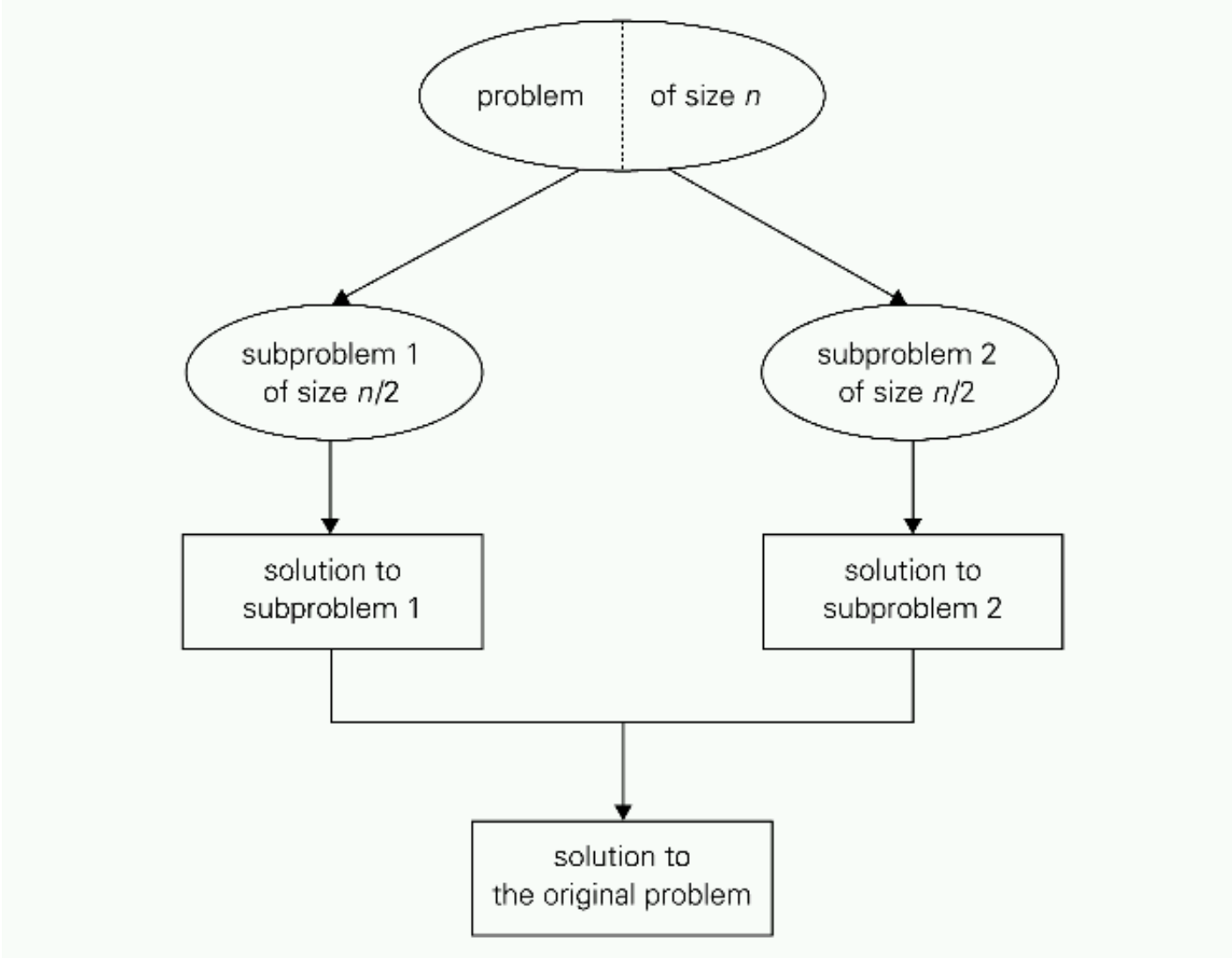
2. Los de kleinere instanties op: meestal recursief

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

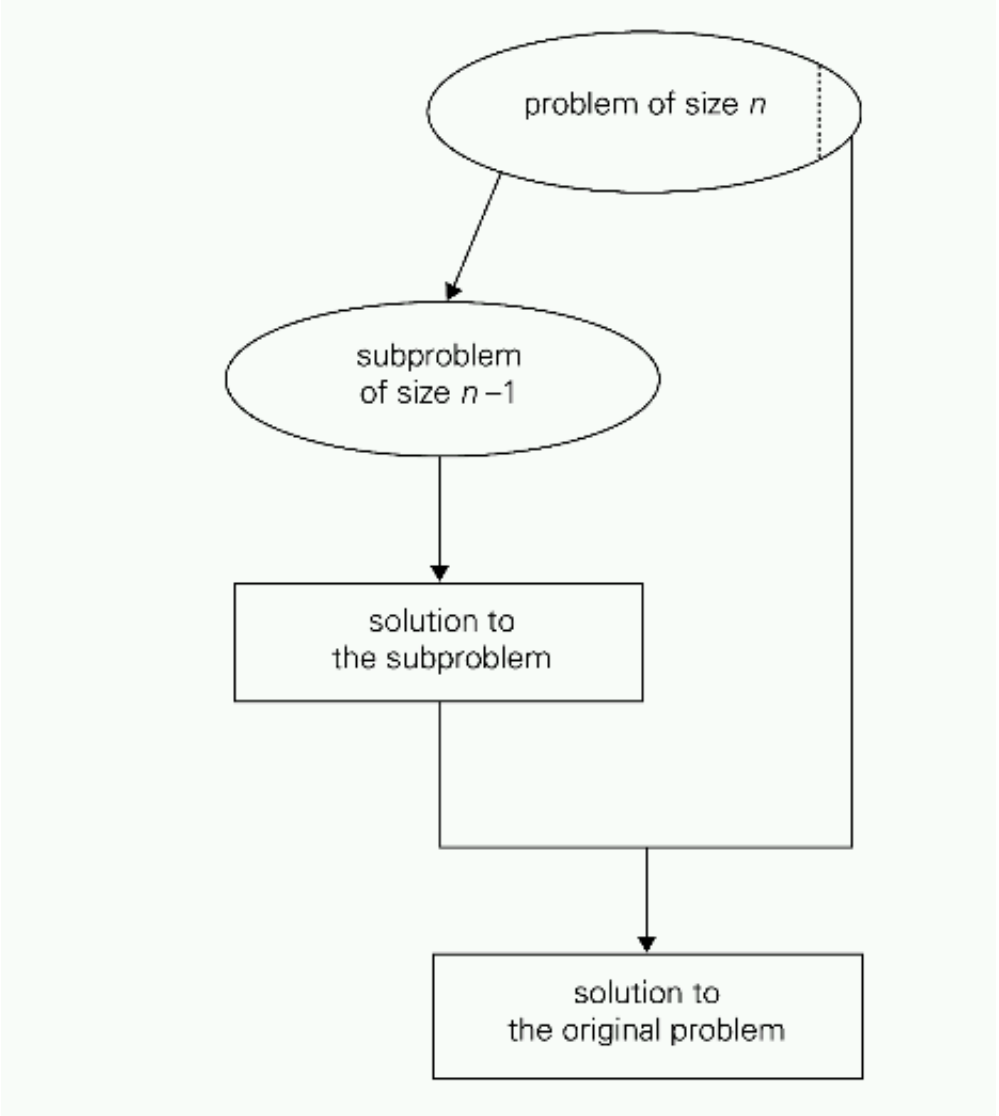
3. Breid de oplossing van de kleinere probleeminstantie uit tot een oplossing van de oorspronkelijke instantie

In het boek wordt onderscheid gemaakt tussen:

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

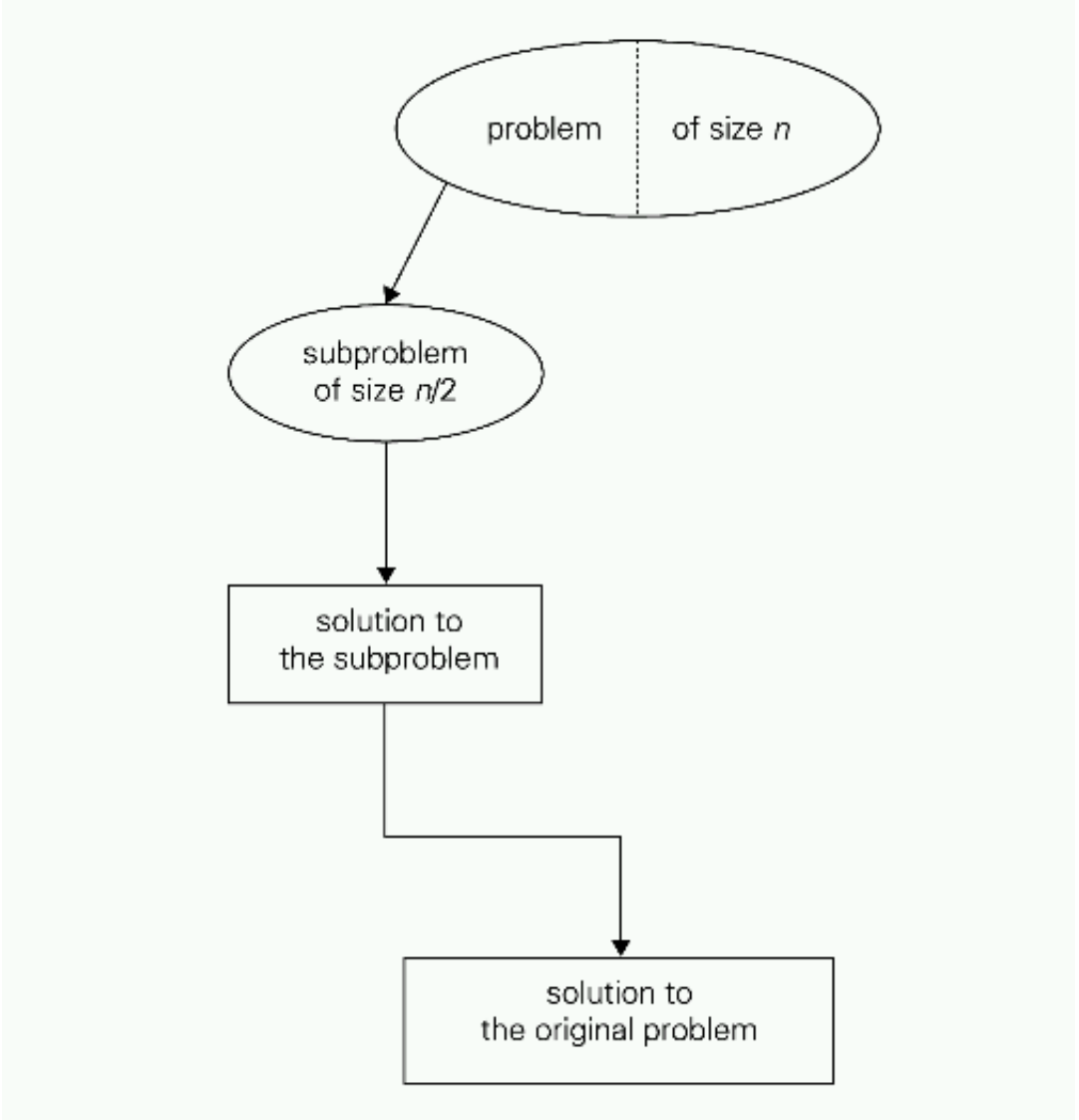
Schematisch (2)

Decrease by one



Decrease by a constant factor

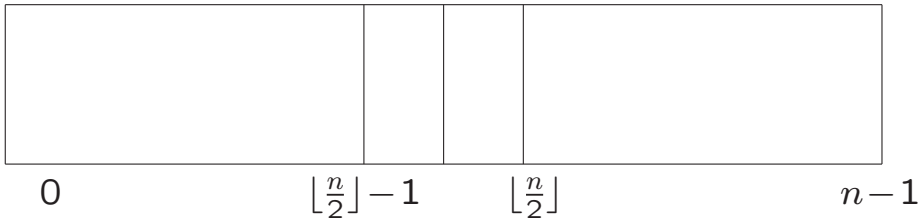
(decrease by half)



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 .
```

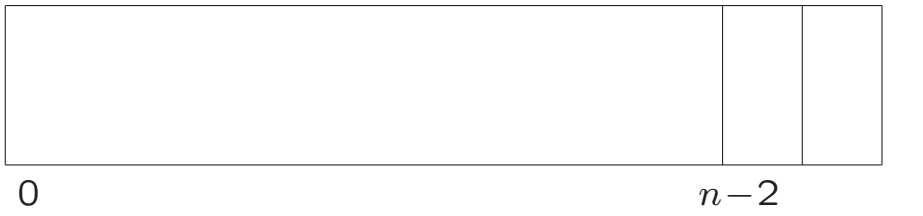
Divide and conquer



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 $n > 1$
kopieer($A[0 \cdots \lfloor \frac{n}{2} \rfloor - 1], B[0 \cdots \lfloor \frac{n}{2} \rfloor - 1]$);
kopieer($A[\lfloor \frac{n}{2} \rfloor \cdots n-1], C[0 \cdots \lceil \frac{n}{2} \rceil - 1]$);
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

Mergesort:

- worst case tijdcomplexiteit: . . .
- extra geheugen: . . .

Mergesort:

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

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**
        kopieer $C[j \cdots q-1]$ naar $A[k \cdots p+q-1]$;
    else
        kopieer $B[i \cdots p-1]$ naar $A[k \cdots p+q-1]$;
    fi .

Basisoperatie: . . .

Worst case

Recurrente betrekking voor $n = 2^k$: . . .

Basisoperatie: `if (B[i] <= C[j])`

Worst case

Recurrente betrekking voor $n = 2^k$:

$$C(n) = \begin{cases} 0 & \text{als } n = 1 \\ n - 1 + 2 \cdot C(n/2) & \text{als } n \geq 2 \end{cases}$$

Met inductie: $\forall k \geq 0 : C(2^k) = (k-1) \cdot 2^k + 1$,
zodat $C(n) = C(2^k) \in \Theta(n \lg n)$

Master Theorem

Stel rekentijd $C(n)$ voldoet aan volgende recurrente betrekking:

$$C(n) = aC(n/b) + f(n)$$

Als $f(n) \in \Theta(n^d)$ met $d \geq 0$, dan

$$C(n) \in \begin{cases} \Theta(n^d) & \text{als } a < b^d \\ \Theta(n^d \log n) & \text{als } a = b^d \\ \Theta(n^{\log_b a}) & \text{als } a > b^d \end{cases}$$

Bij mergesort. . .

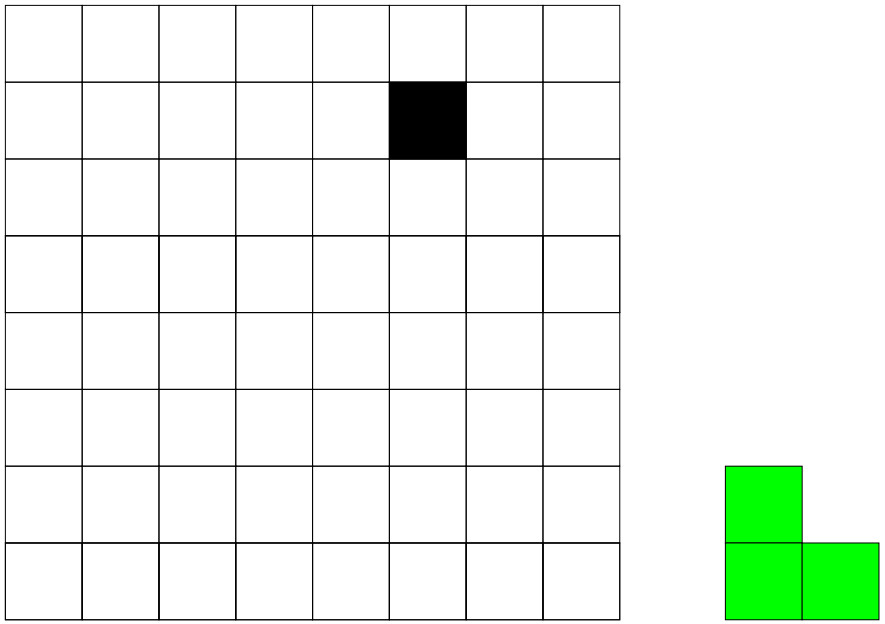
```
int som (int A[], int l, int r)
{
  if (l == r)
    return A[l];
  else
  { int mid = (l+r)/2;
    return som (A, l, mid) + som (A, mid+1, r);
  }
}
```

recurrente betrekking...

master theorem...

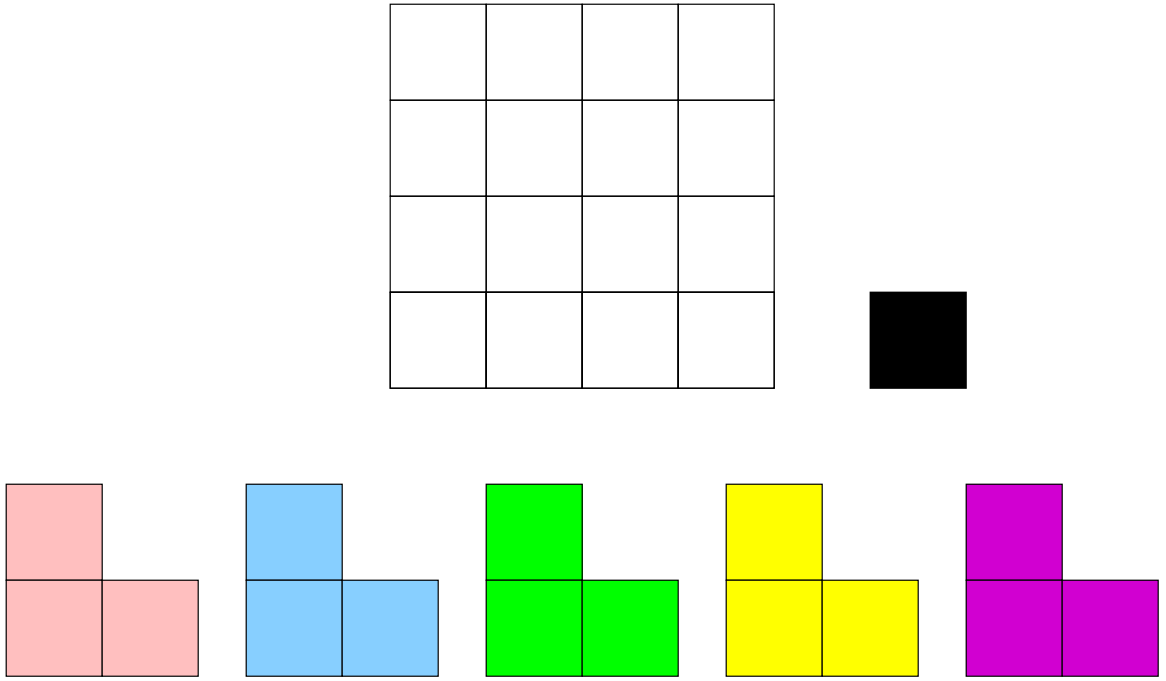
Nog een leuk voorbeeld van de methode Verdeel en heers is de Tromino puzzel, Levitin 5.1.11.

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

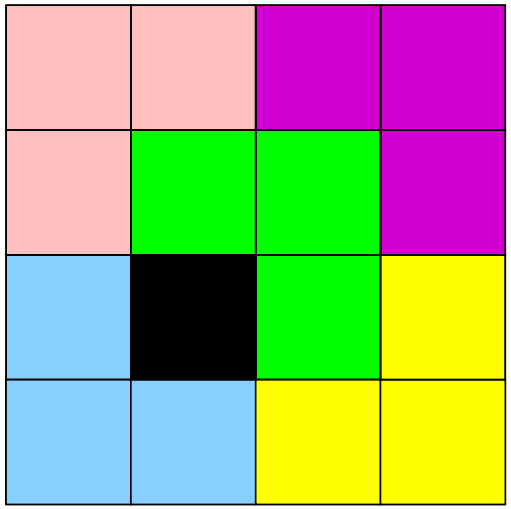


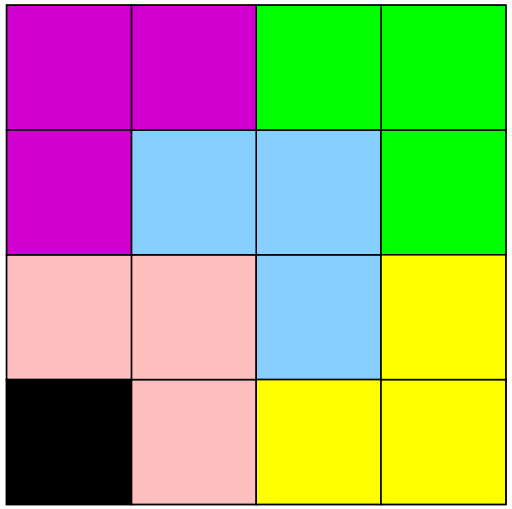
Het 8x8 geval

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



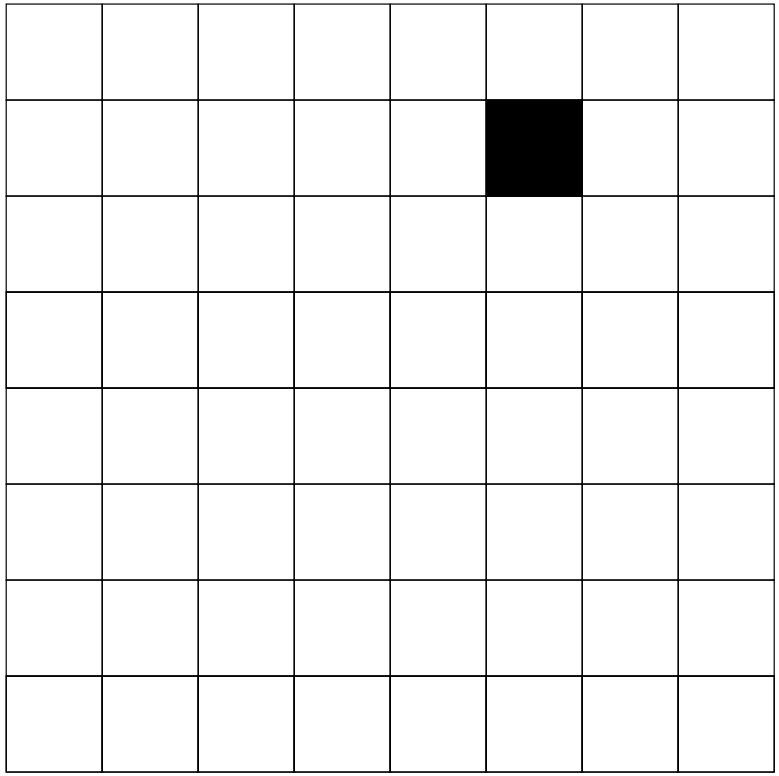
Het 4x4 geval: twee probleeminstanties met oplossing (= bedekking)



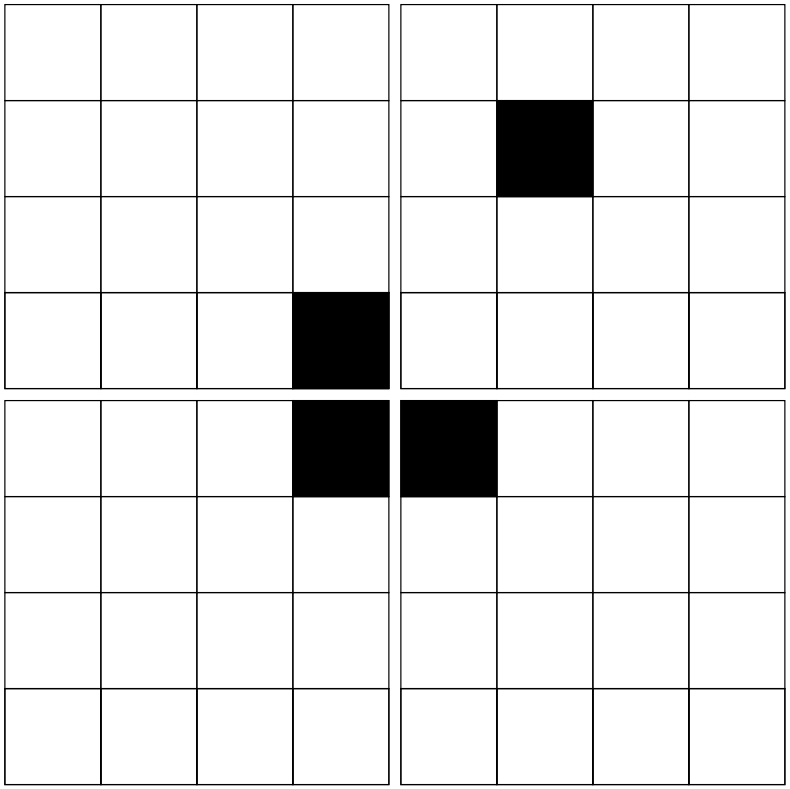


Het 8x8 geval: hoe vinden we een (de?) bedekking met tromino's?

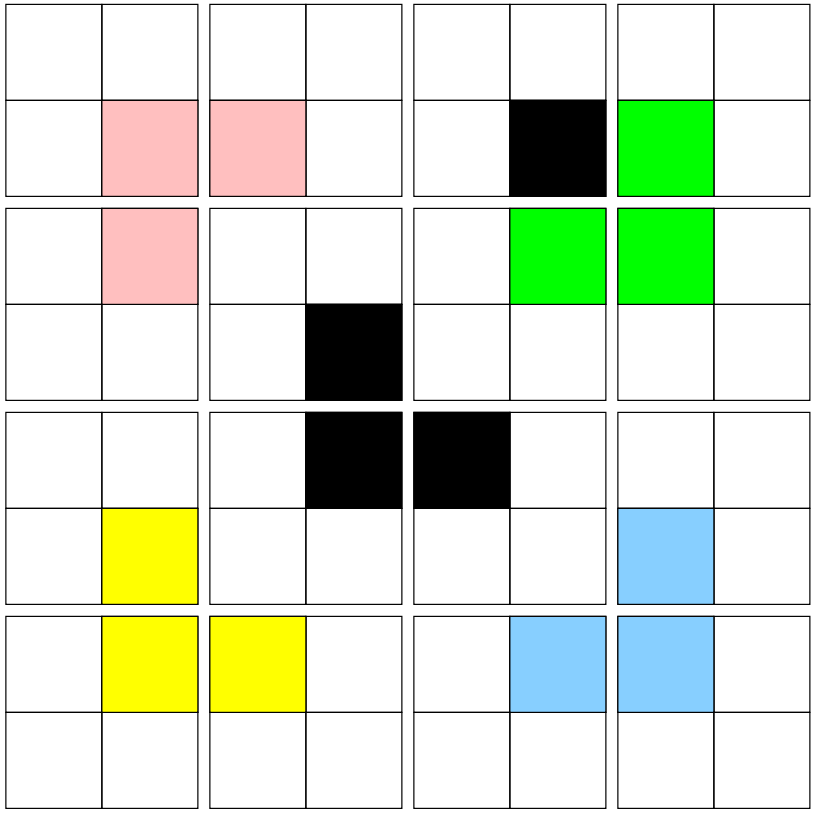
Bijvoorbeeld voor dit bord:



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.

- Voor welke waarden van m zijn $m \times m$ schaakborden zeker niet te bedekken met tromino's?

- Geef een bedekking van het 5×5 schaakbord met het lege vakje bijvoorbeeld in het midden van de meest linker kolom.

- Geef een bedekking van het 8×8 bord van vorige slide, anders dan die is gevonden met behulp van het divide and conquer algoritme.

Gegeven n punten $p_1 = (x_1, y_1), \ldots, p_n = (x_n, y_n)$.
Gevraagd het/een tweetal punten dat het dichtst bij elkaar ligt. Afstandsmaat: $\mathrm{d}(p_i, p_j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$.

Brute force algoritme: alle paren (p_i, p_j) (met $i < j$) aflopen en hun onderlinge afstanden $\mathrm{d}(p_i, p_j)$ vergelijken.

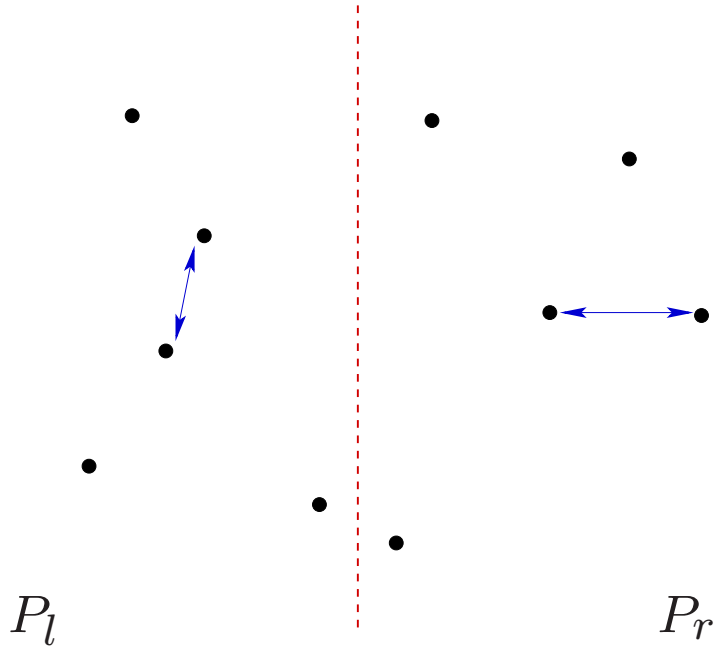
```
dmin := ∞;
for i := 1 to n − 1 do
    for j := i + 1 to n do
        d := (x_i − x_j)^2 + (y_i − y_j)^2;
        if d < dmin
            dmin := d; k := i; l := j;
        fi // (p_k, p_l) voorlopig closest pair
    od
od
```

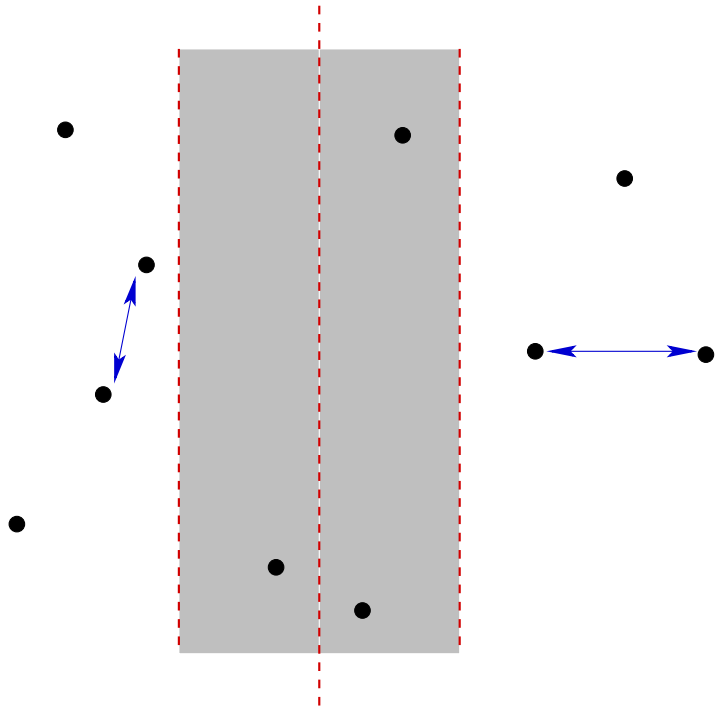
Complexiteit: $\frac{1}{2}n(n-1) = \Theta(n^2)$

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 $O(n)$ puntenparen bekeken te moeten worden.
Dit levert een $O(n \lg n)$ algoritme op.

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

- **Deze week geen werkcollege**

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

- **Volgend college:**
  maandag 13 april 2026, 11.00-12.45