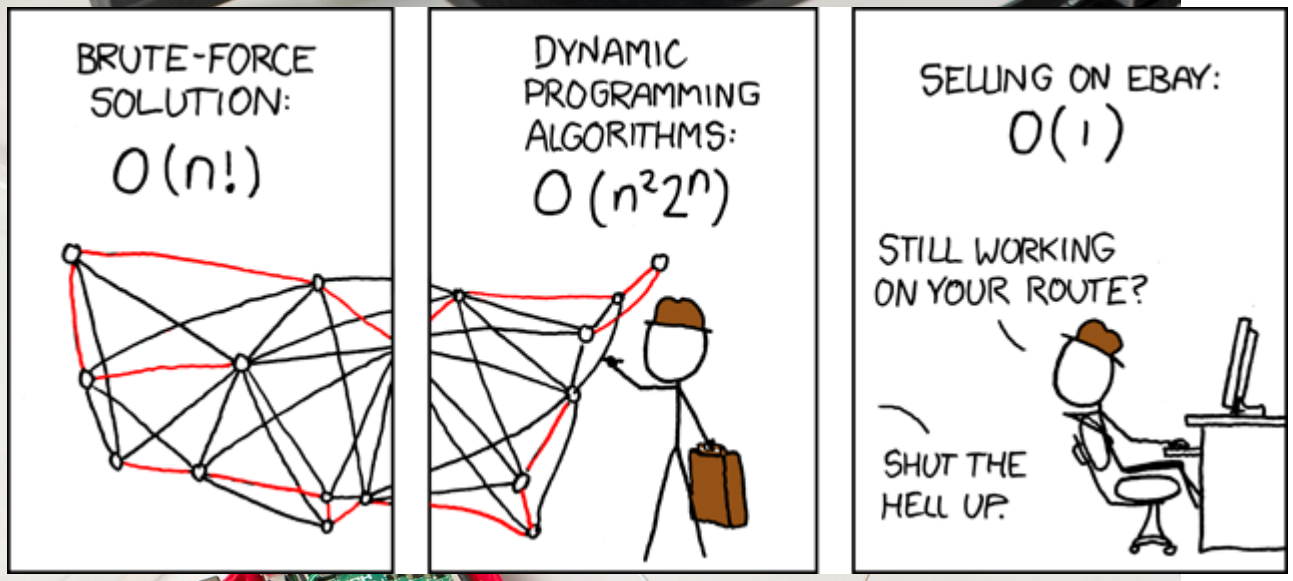
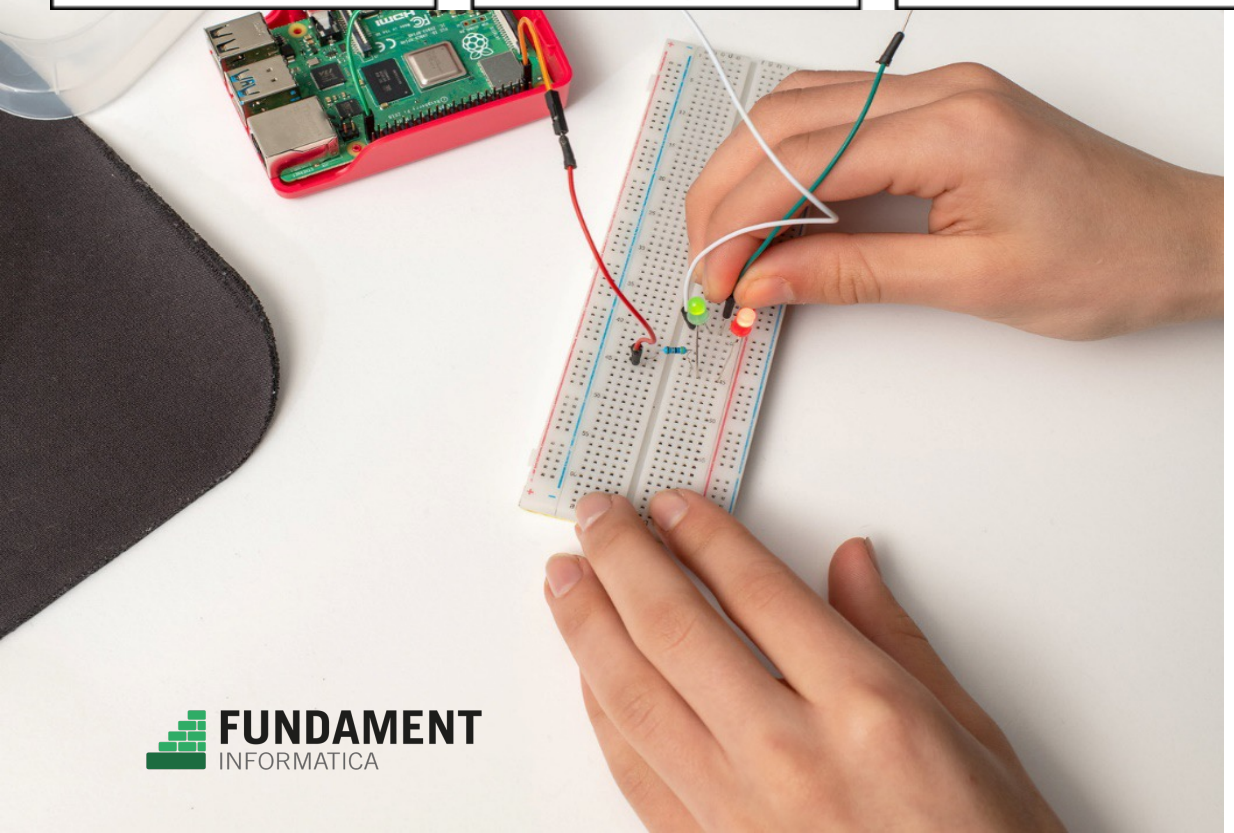




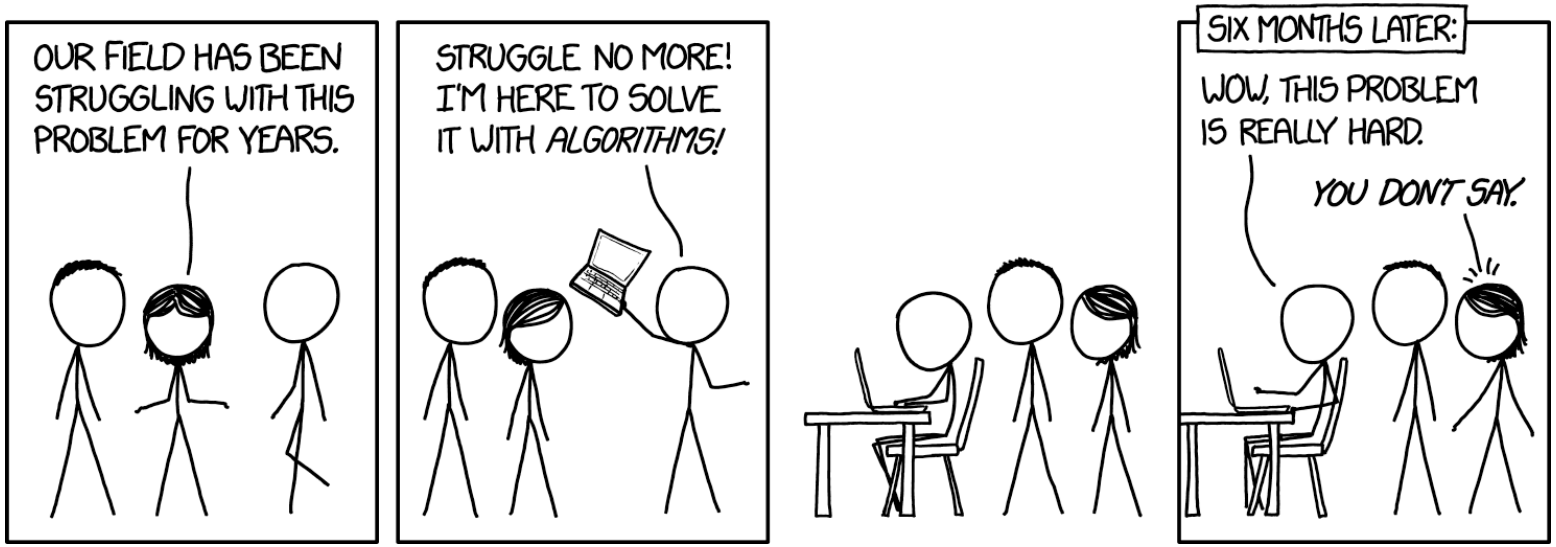
Module G. Algoritmiek

Hoe bepaal je de orde van een algoritme?

WORKSHOP VOORJAARSCONFERENTIE INFORMATICA 2026

Module G. Algoritmiek – op vwo-niveau

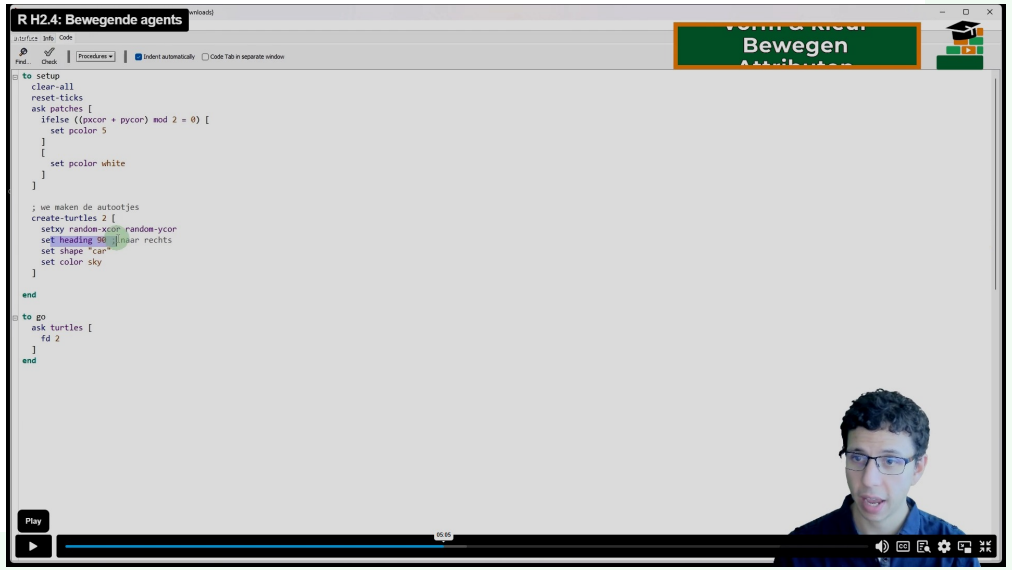
- Introductie
- Groeigedrag (tijdscomplexiteit)
- Orde
- Analyseren



FUNDAMENT INFORMATICA

Dit ben ik

- Dr. Ir. Kevin van As
- Studie: natuurkunde TUDelft (2010 – 2024)
- Docent informatica & natuurkunde (Ashram College Alphen a/d Rijn) (2019 – heden)
- Organisator CoderDojo club (2017 – heden)
- Ontwikkelaar bij Instruct (2023 – heden) (lesmethode Fundament informatica)



FUNDAMENT INFORMATICA

Eindterm 1

Complexiteit van algoritmen

- De kandidaat kan (havo:) van gegeven algoritmen de **complexiteit vergelijken**, en kan klassieke 'moeilijke' problemen herkennen en benoemen.

- De kandidaat kan (vwo:) het verschil tussen exponentiële en polynomiale **complexiteit uitleggen**, kan algoritmen op basis hiervan onderscheiden, en kan klassieke 'moeilijke' problemen herkennen en benoemen.

~~Eindtermen 2 en 3~~

Berekenbaarheid

- De kandidaat kan berekeningen op verschillende abstractieniveaus karakteriseren en relateren, en kan klassieke **onberekenbare problemen** herkennen en benoemen.

Logica

- De kandidaat kan eigenschappen van digitale artefacten uitdrukken in **logische formules**.

Groeigedrag

Module G. Algoritmiek

FUNDAMENT INFORMATICA

Complexiteit van algoritmen

- **Tijdscomplexiteit**
  - Hoeveel langer duurt het algoritme om uit te voeren bij een grotere invoer?
- **Ruimtecomplexiteit**
  - Hoeveel meer werkgeheugen heeft het algoritme nodig bij een grotere invoer?

- Vaak een afweging: tussenantwoorden bewaren of opnieuw berekenen?

- Voor leerlingen: alleen tijdscomplexiteit → **groeigedrag**

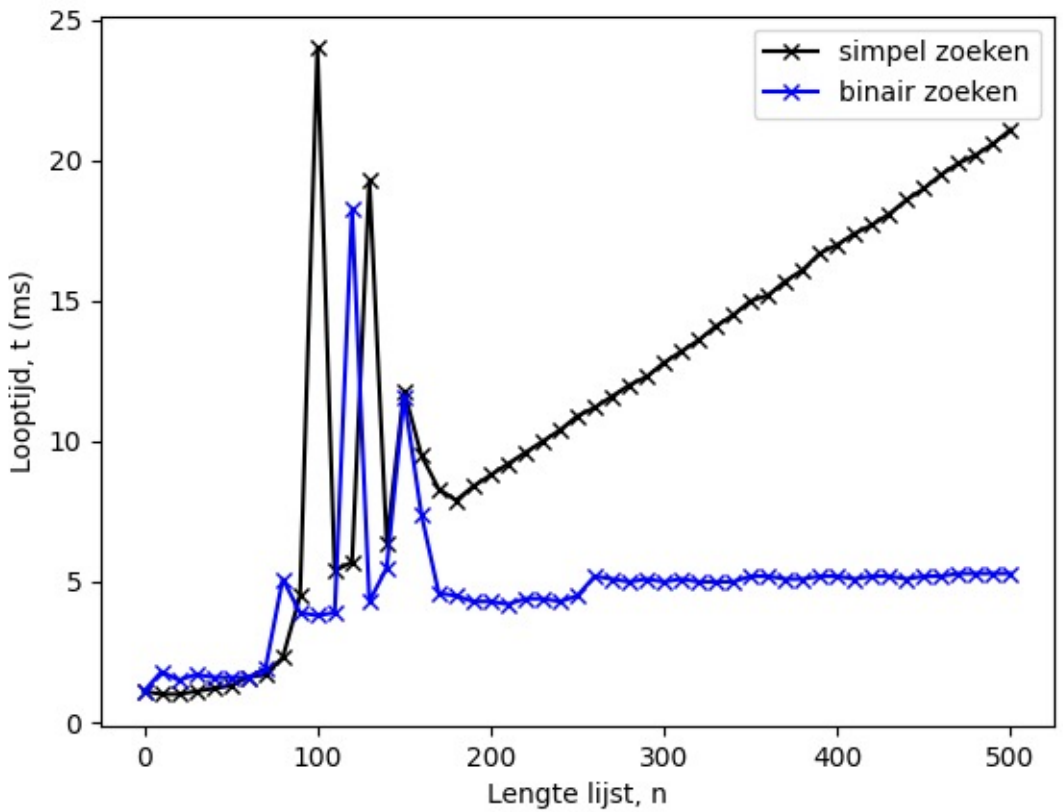
Looptijden meten

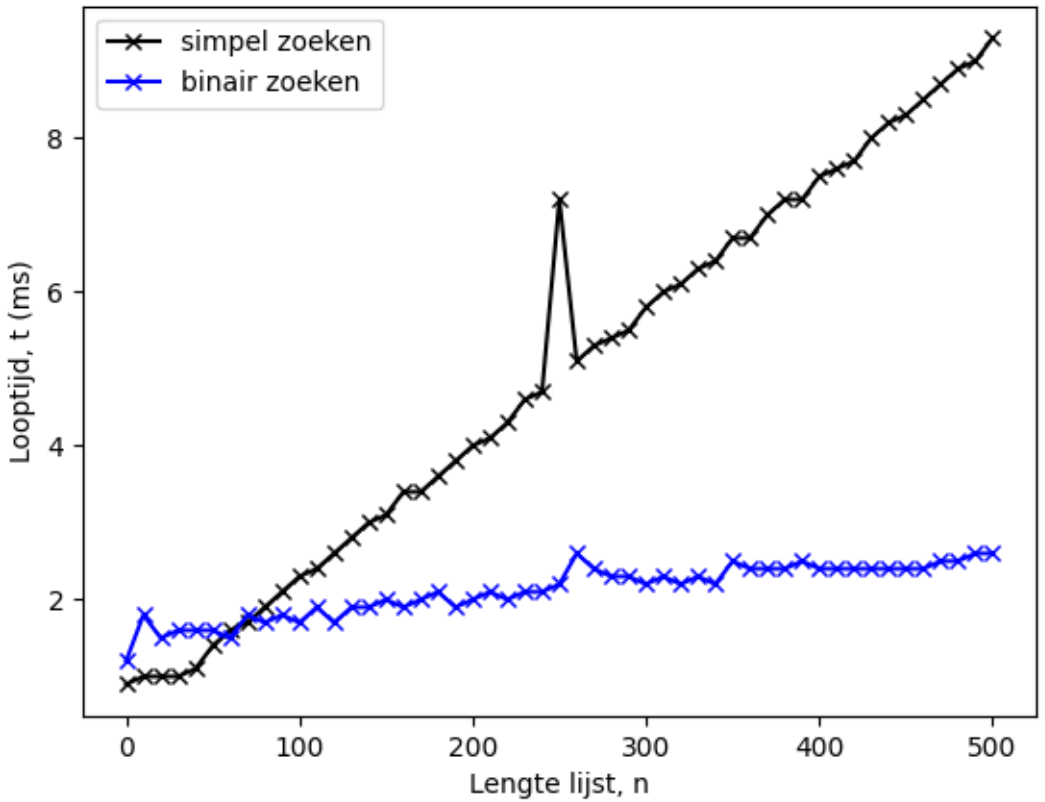
- Looptijd: $\Delta t = t_{\text{na}} - t_{\text{voor}} = 15{:}00 - 12{:}30 = 2{:}30$

Looptijden plotten

Vraag: waarom dit verschil?

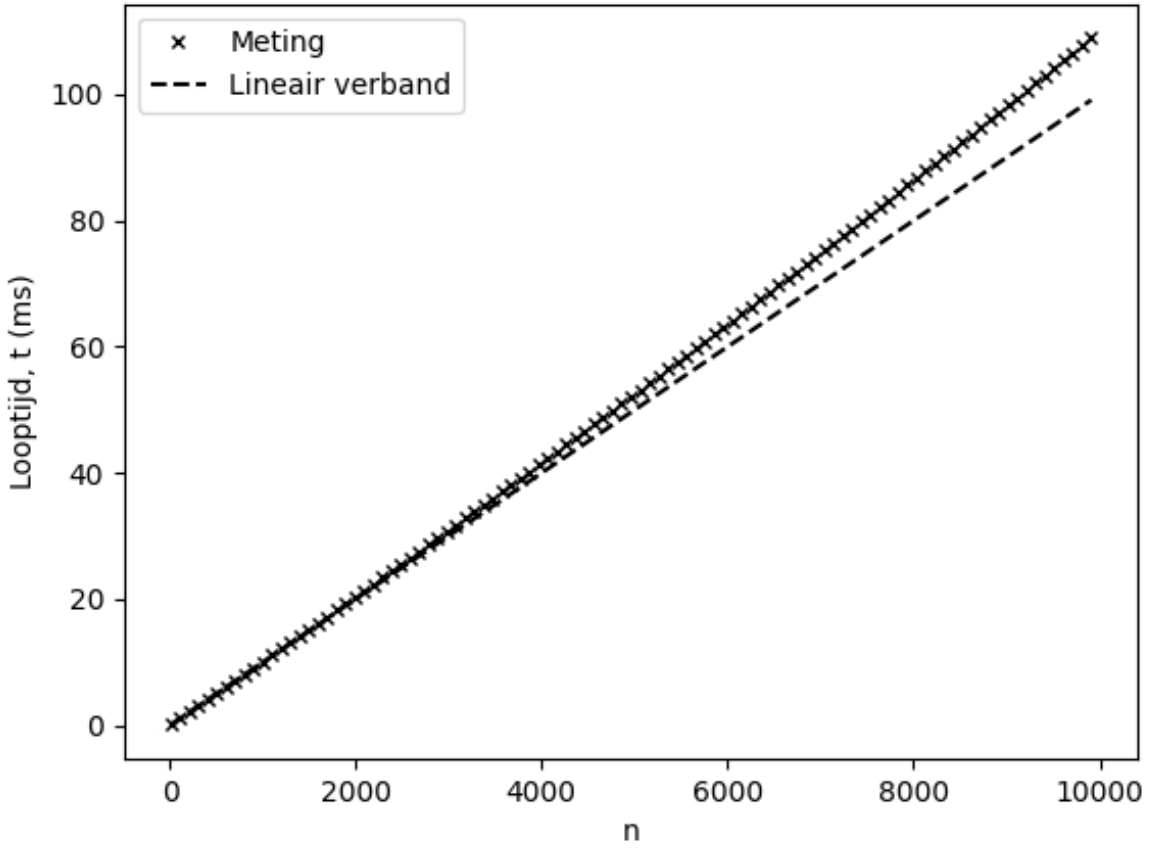
- Twee keer uitvoeren, twee verschillende resultaten





Groeigedrag vergelijken

- **Orde**
  - Formule voor groeigedrag
  - Formule alleen geldig bij “hele grote” invoergroottes
- Grote-O **notatie**:
  - “Dit algoritme heeft orde n”: $O(n)$
  - “Dit algoritme heeft orde n²”: $O(n^2)$



Looptijden inschatten m.b.v. orde

- **Orde**
  - Verhouding tussen invoergrootte *n* en looptijd t_n

Verhoudingstabel:

t_{n_2}	O_{n_2}
t_{n_1}	O_{n_1}

- Voorbeelden:
  - $O(n)$ $\xrightarrow{\text{betekent}}$ $\frac{t_{n_2}}{t_{n_1}} = \frac{n_2}{n_1}$ $\xrightarrow{\text{omschrijven}}$ $t_{n_2} = t_{n_1} \cdot \frac{n_2}{n_1}$
  - $O(\log(n))$ $\xrightarrow{\text{betekent}}$ $\frac{t_{n_2}}{t_{n_1}} = \frac{\log(n_2)}{\log(n_1)}$ $\xrightarrow{\text{omschrijven}}$ $t_{n_2} = t_{n_1} \cdot \frac{\log(n_2)}{\log(n_1)}$

Misconceptie quiz

workshop

- Visit **gosocrative.com** and enter room name **KEVIN2664**

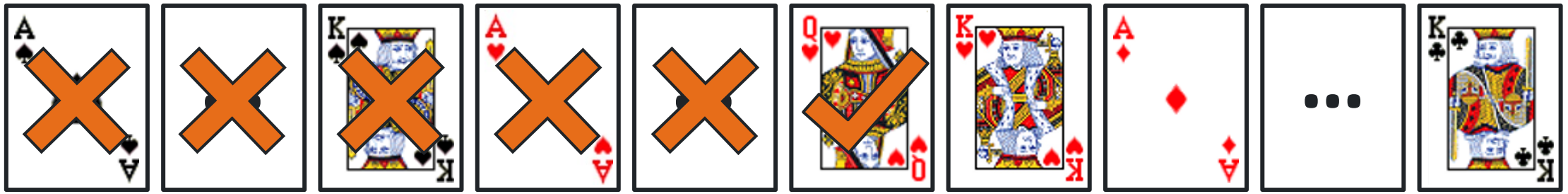
Algoritmen analyseren

Module G. Algoritmiek

FUNDAMENT INFORMATICA

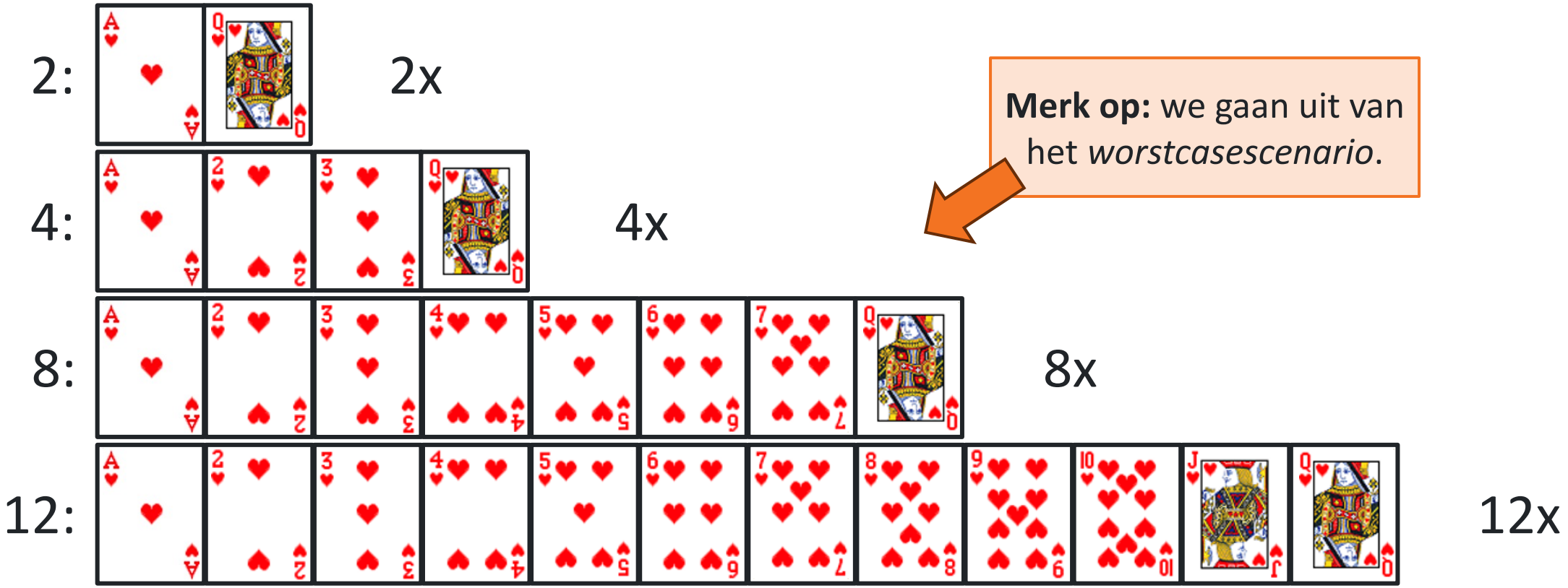
Lineair zoeken

Vraag: hoeveel pogingen nodig?



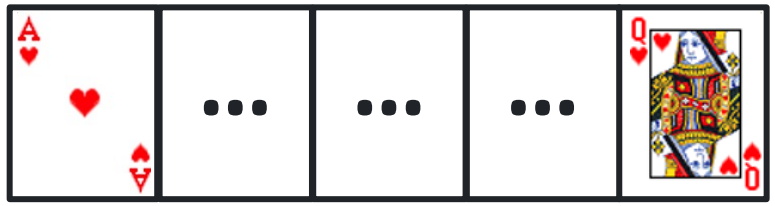
Orde van lineair zoeken

- Hoe vaak moeten we raden om ❤️V te vinden?



Orde van lineair zoeken

- Hoe vaak moeten we raden om ❤️V te vinden?
- In *het algemene geval* geldt (*n* is de grootte van de lijst):

n:  *n*x

- ***O*(*n*)** stappen
- Lineair verband!
  - Grootte van invoer x2 geeft looptijd x2

Van aantal stappen naar groeigedrag

- Groeigedrag != aantal stappen
- Groeigedrag = hoe schaalt de looptijd met de invoergrootte:

$$\text{totaal\_werk} = \text{werk\_per\_stap} \times \text{aantal\_stappen}$$

- Lineair zoeken
  - Aantal stappen: O(n)
  - Werk per stap: één if-conditie: *"Is dit de* ❤️*V?"* Onafhankelijk van *n*, dus O(1)
  - Totale orde: O(n) x O(1) = **O(n)**

Het gebeurt heel vaak dat het O(1) is, maar zeker niet altijd.

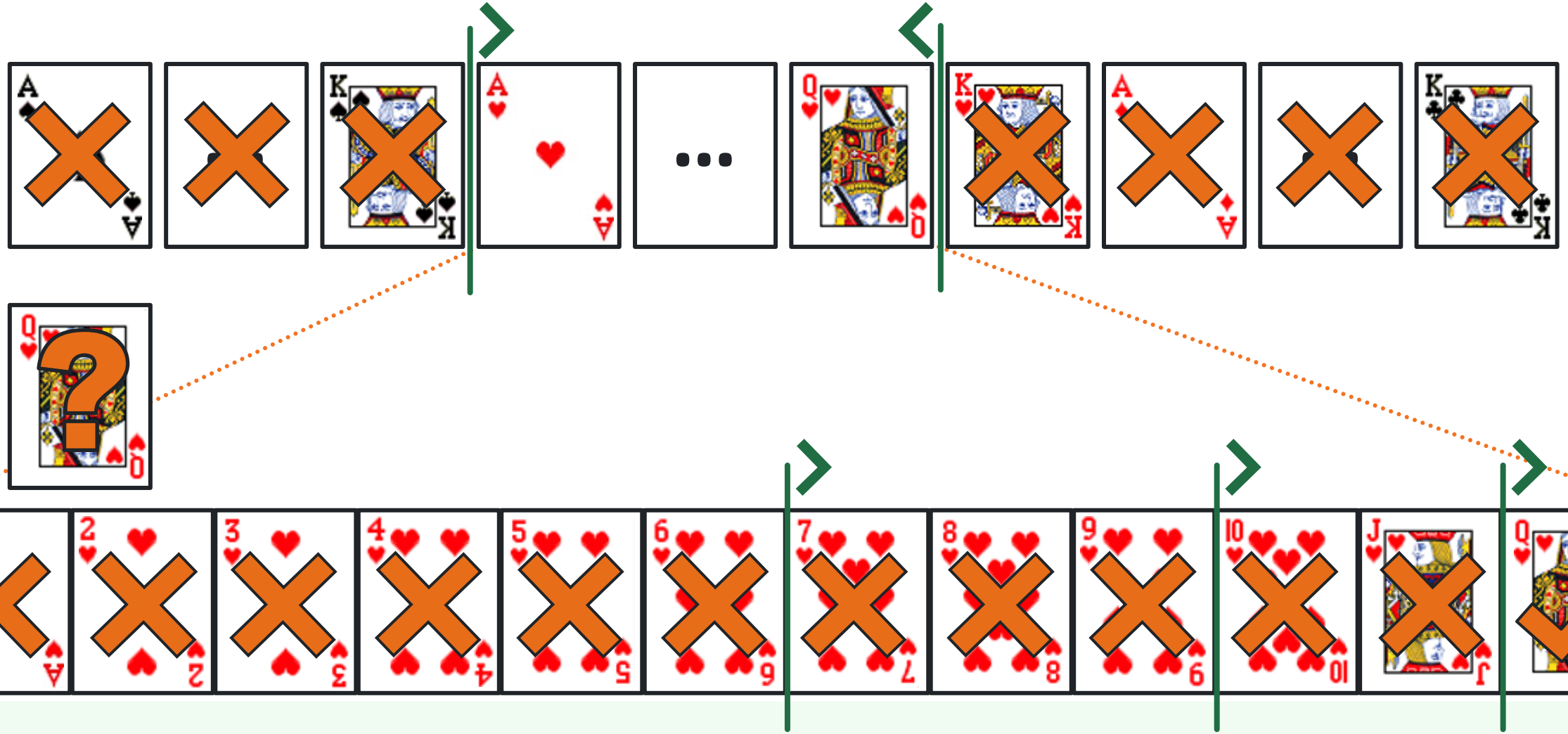
Aan de slag!

workshop

- Maak opdracht 1 en 2
- Experts: maak opdracht 1 en 3

Binair zoeken

Vraag: hoeveel pogingen nodig?



FUNDAMENT
INFORMATICA

Orde van binair zoeken

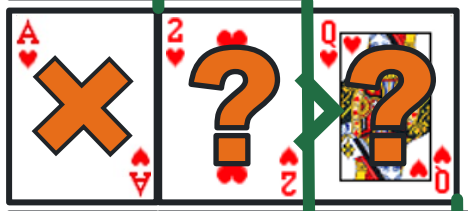
Merk op: we gaan uit van het *worstcasescenario*.

- Hoe vaak moeten we raden om ♥V te vinden?

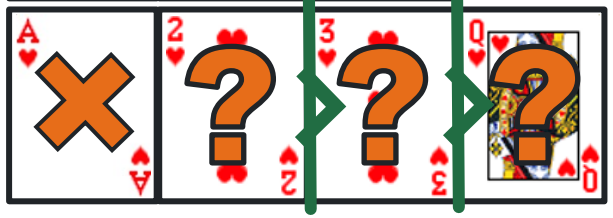
1: 1x

2: 2x

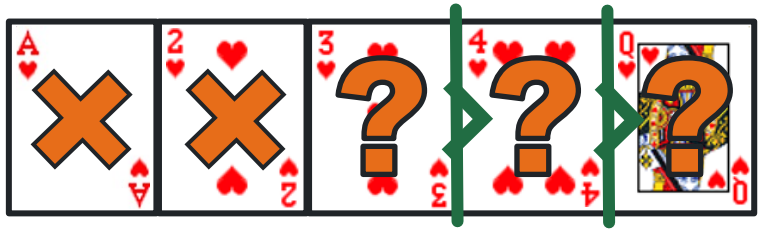
3: 2x



4: 3x



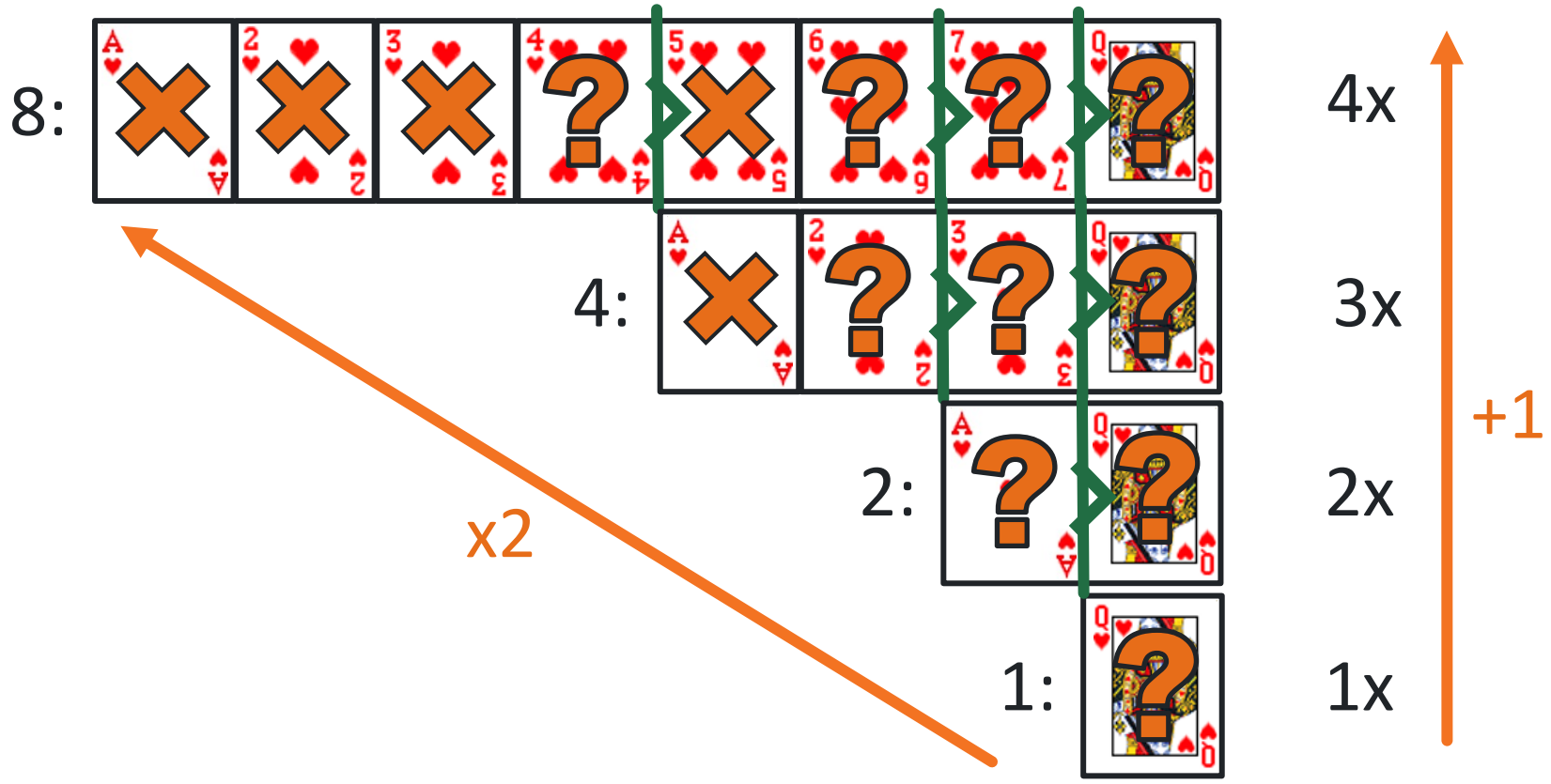
5: 3x



- Wat is het verband…?

Orde van binair zoeken

- Hoe vaak moeten we raden om ❤️V te vinden?



Orde van binair zoeken

- Hoe vaak moeten we raden om ❤️V te vinden?

$100/2/2/2/2/2/2/2 \approx 1$

7 keer halveren

200 kaarten

$200/2/2/2/2/2/2/2/2 \approx 1$

8 keer halveren

260.000 kaarten

$260.000/2/\cdots/2 \approx 1$

18 keer halveren

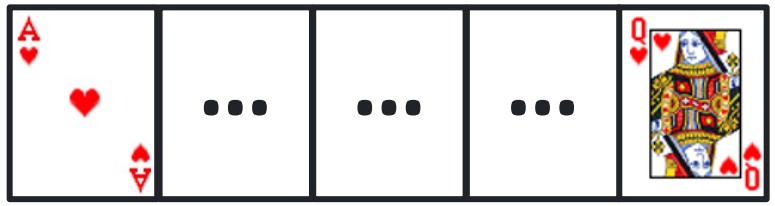
In het algemene geval: *n* kaarten

$n/2/\cdots/2 \approx 1$

$\log_2(n)$ keer halveren

Orde van binair zoeken

- Hoe vaak moeten we raden om ❤️V te vinden?
- In *het algemene geval* geldt:

n: 

$$\log_2(n) = \lg(n)$$

- ***O*(lg(*n*))** (er is weer O(1) werk per stap)
- Logaritmisch verband!
  - Grootte van invoer x2 geeft looptijd +constante

n	Looptijd binair zoeken
100	7 milliseconden
100.000	17 milliseconden
100.000.000	27 milliseconden
100.000.000.000	37 milliseconden

Aan de slag!

workshop

- Maak opdracht 4
- Experts: maak opdracht 5

Dijkstra’s algoritme

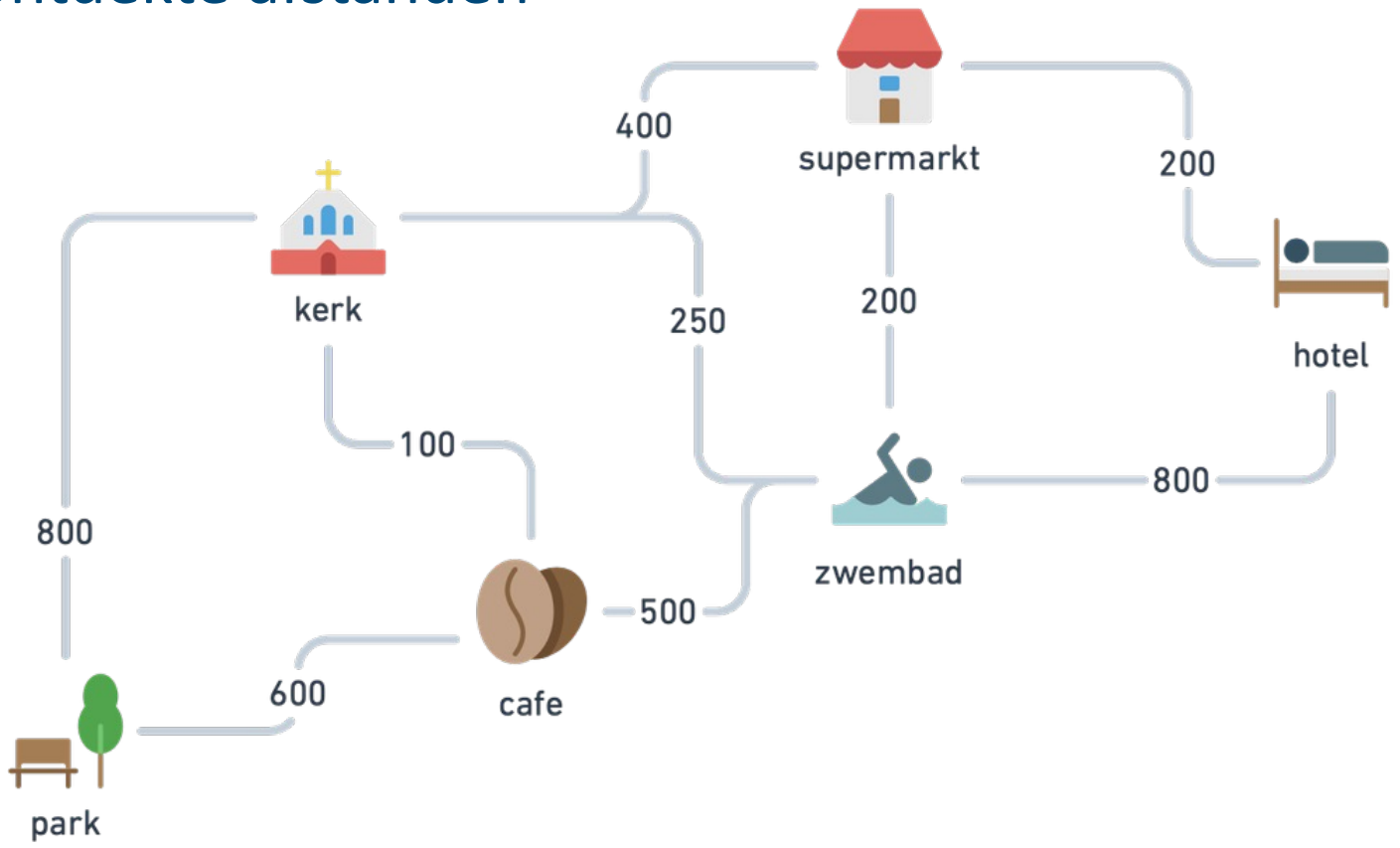
Module G. Algoritmiek

FUNDAMENT INFORMATICA

Dijkstra’s algoritme

1. Maak een lijst met de ontdekte afstanden

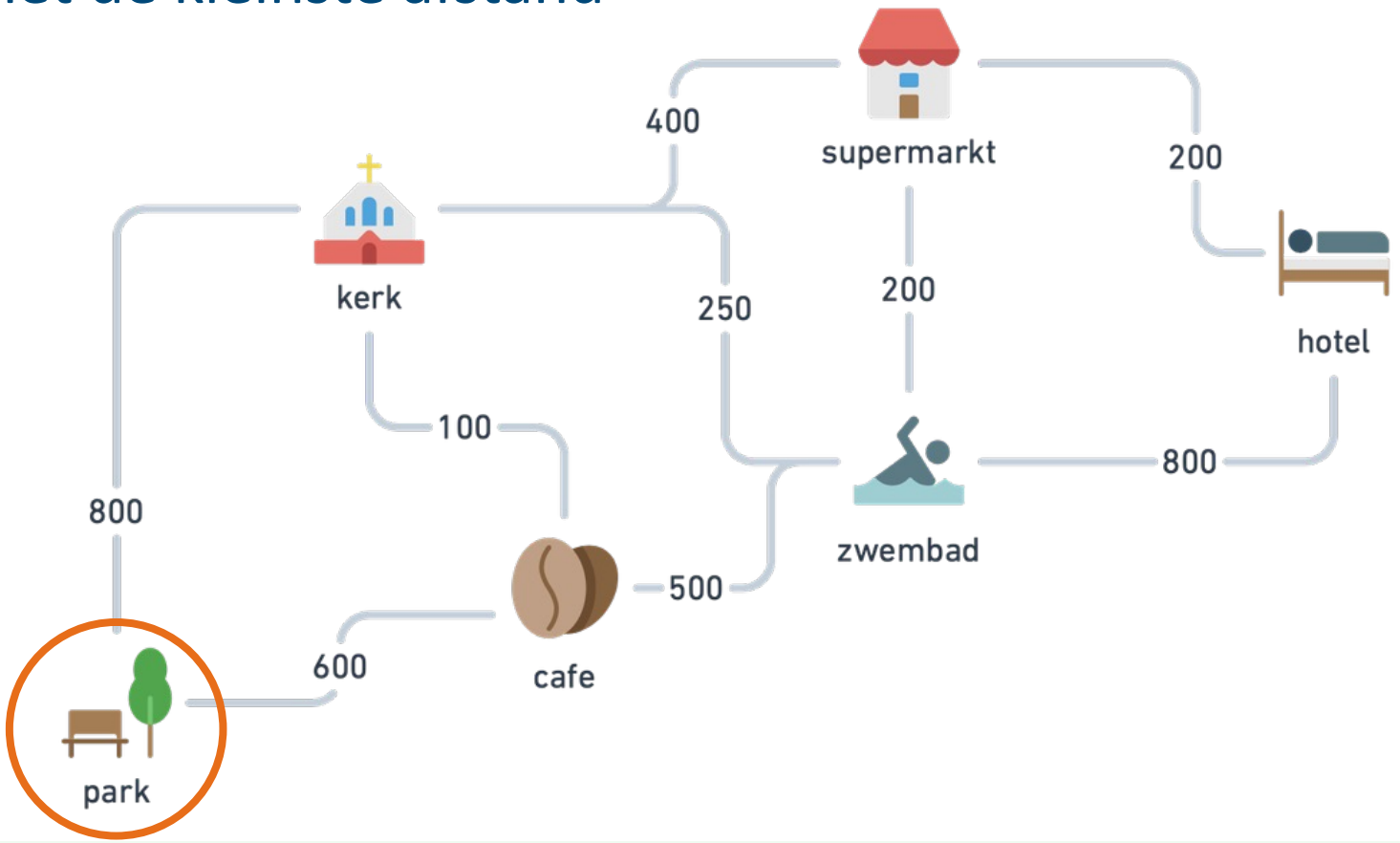
Knopen	Afstand
Park	0
Kerk	∞
Café	∞
Supermarkt	∞
Zwembad	∞
Hotel	∞



Dijkstra’s algoritme

2. Selecteer de knoop met de kleinste afstand

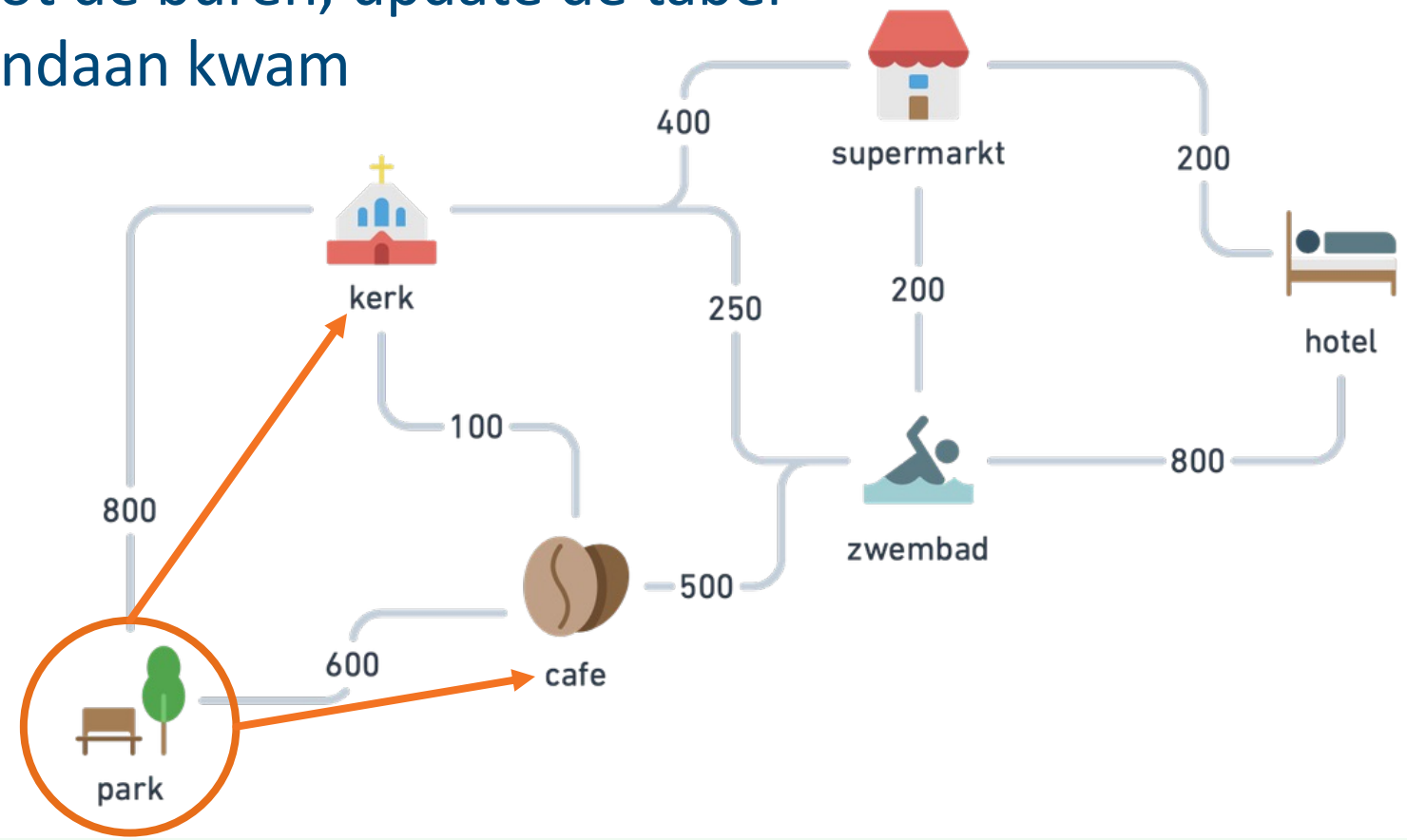
Knopen	Afstand
Park	0
Kerk	∞
Café	∞
Supermarkt	∞
Zwembad	∞
Hotel	∞



Dijkstra's algoritme

3. Bereken de afstand tot de buren, update de tabel en noteer waar je vandaan kwam

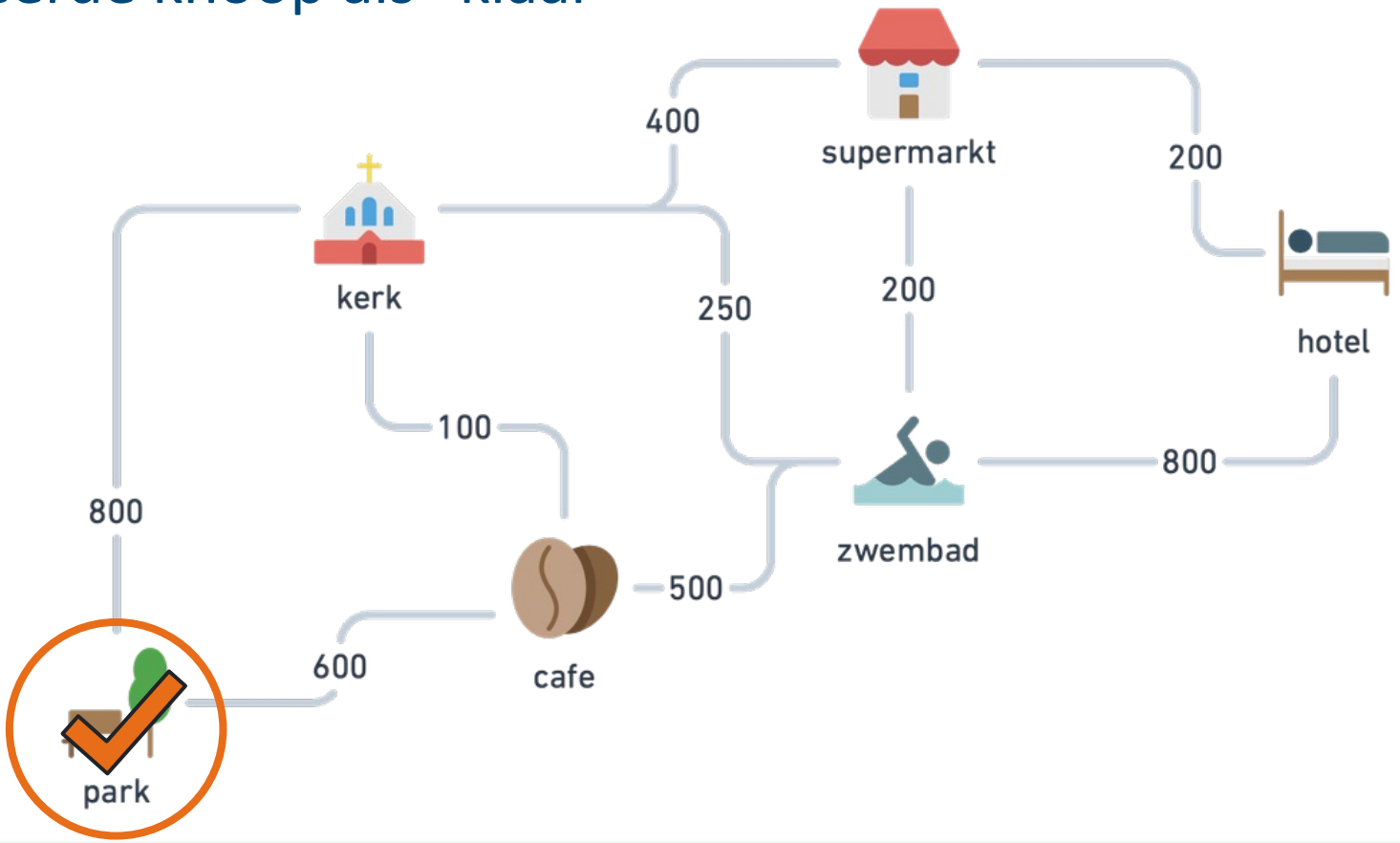
Knopen	Afstand	Vorige
Park	0	-
Kerk	800	Park
Café	600	Park
Supermarkt	∞	?
Zwembad	∞	?
Hotel	∞	?



Dijkstra's algoritme

4. Markeer de geselecteerde knoop als "klaar"
5. Herhaal vanaf stap 2

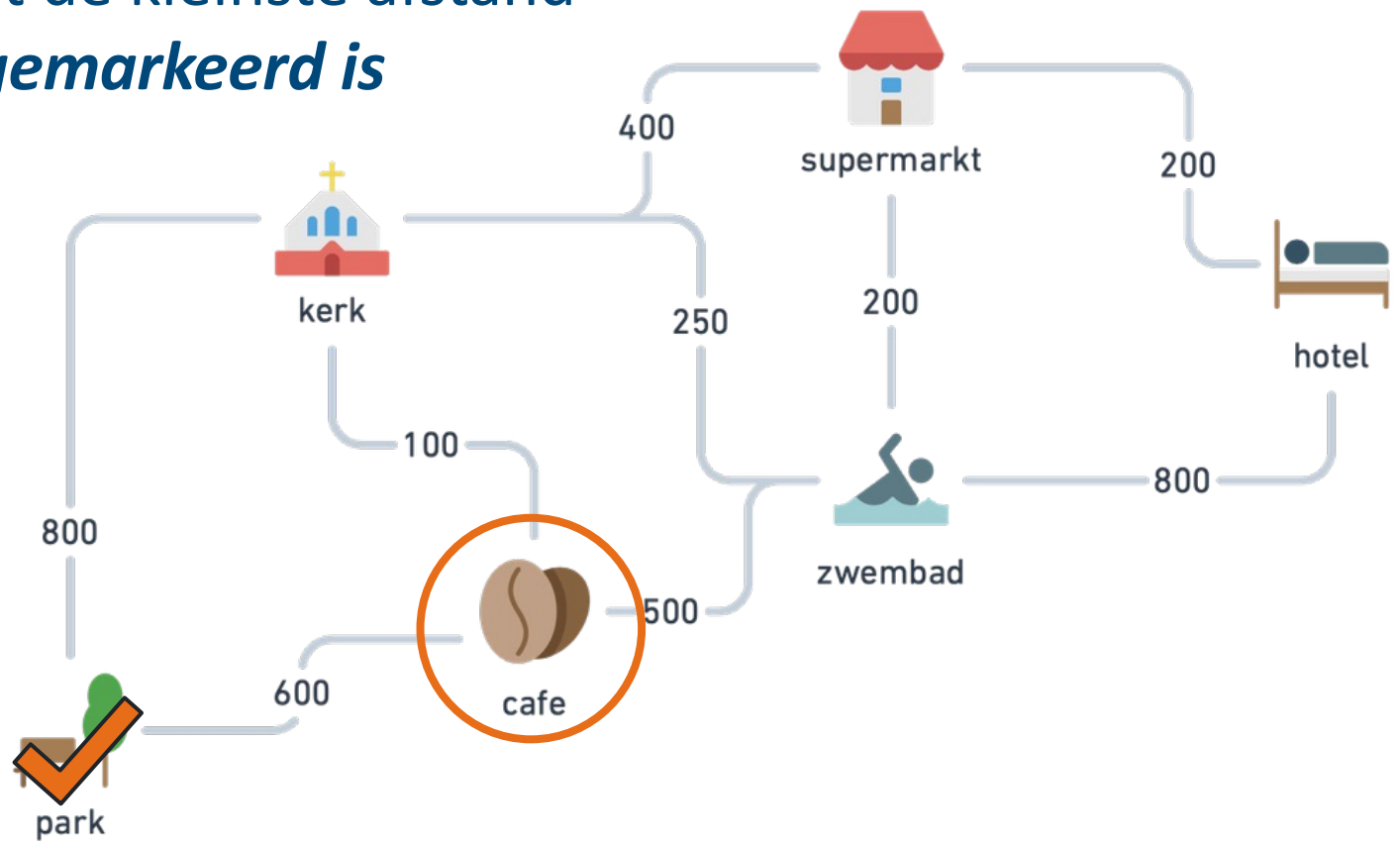
✓?	Knopen	Afstand	Vorige
✓	Park	0	-
	Kerk	800	Park
	Café	600	Park
	Supermarkt	∞	?
	Zwembad	∞	?
	Hotel	∞	?



Dijkstra's algoritme

2. Selecteer de knoop met de kleinste afstand
   die nog niet als klaar gemarkeerd is

✓?	Knopen	Afstand	Vorige
✓	Park	0	-
	Kerk	800	Park
	Café	600	Park
	Supermarkt	∞	?
	Zwembad	∞	?
	Hotel	∞	?

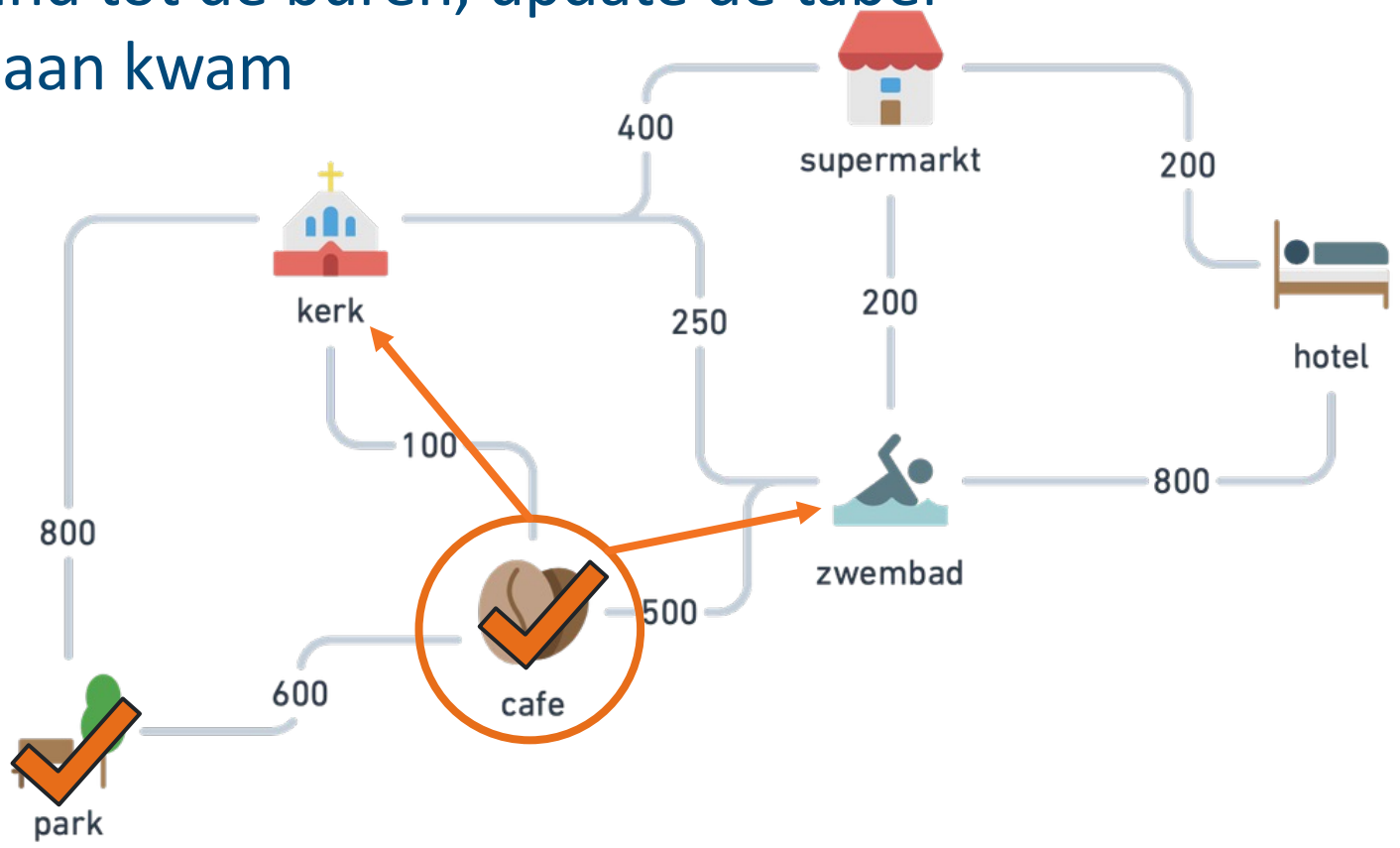


Dijkstra's algoritme

3. Bereken de ***totale*** afstand tot de buren, update de tabel en noteer waar je vandaan kwam

✓?	Knopen	Afstand	Vorige
✓	Park	0	-
	Kerk	700	Café
✓	Café	600	Park
	Supermarkt	∞	?
	Zwembad	1100	Café
	Hotel	∞	?

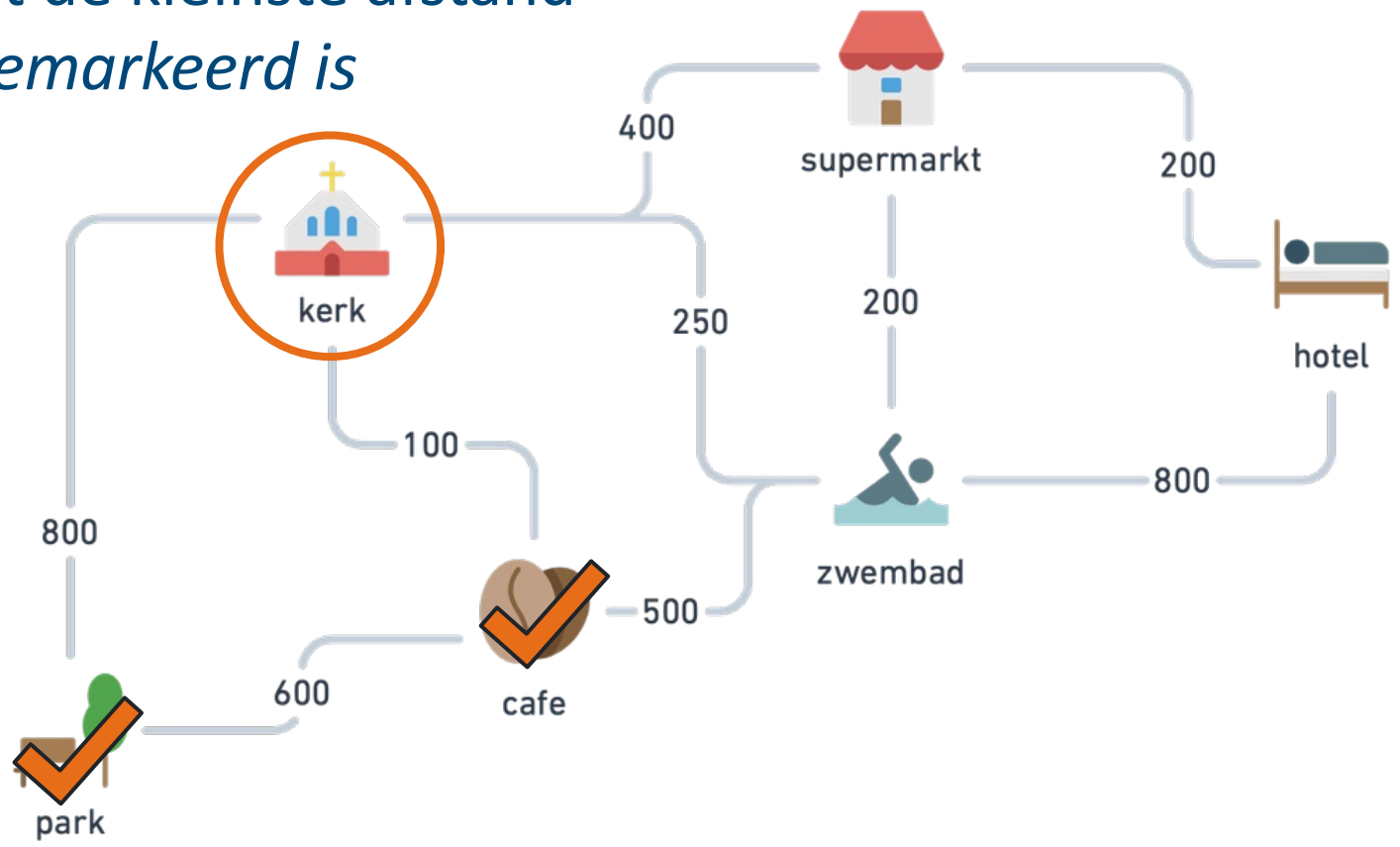
4. Markeer als "klaar"



Dijkstra's algoritme

2. Selecteer de knoop met de kleinste afstand
   die nog niet als klaar gemarkeerd is

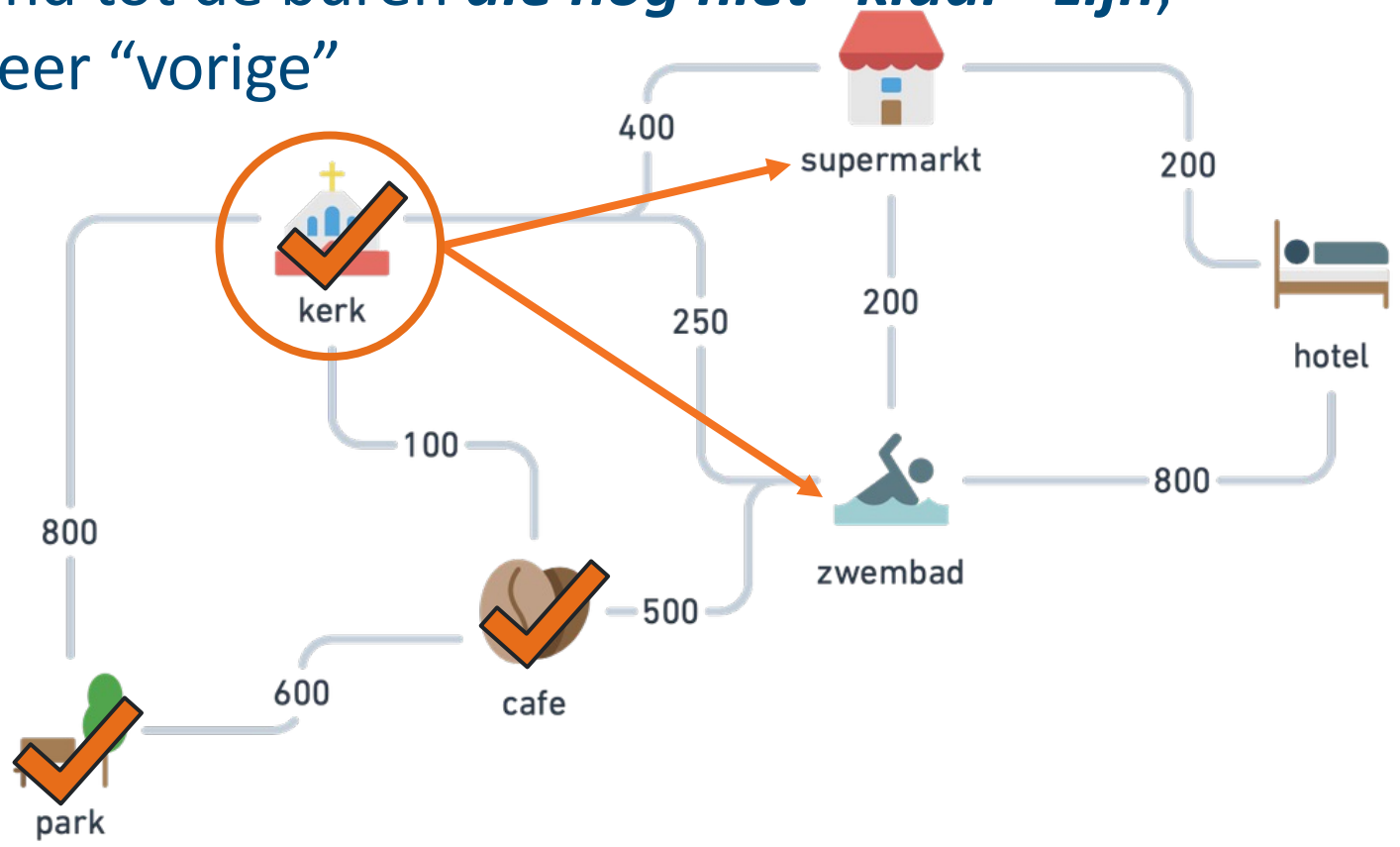
✓?	Knopen	Afstand	Vorige
✓	Park	0	-
	Kerk	700	Café
✓	Café	600	Park
	Supermarkt	∞	?
	Zwembad	1100	Café
	Hotel	∞	?



Dijkstra's algoritme

3. Bereken de *totale* afstand tot de buren ***die nog niet "klaar" zijn***, update de tabel en noteer "vorige"

✓?	Knopen	Afstand	Vorige
✓	Park	0	-
✓	Kerk	700	Café
✓	Café	600	Park
	Supermarkt	1100	Kerk
	Zwembad	950	Kerk
	Hotel	∞	?

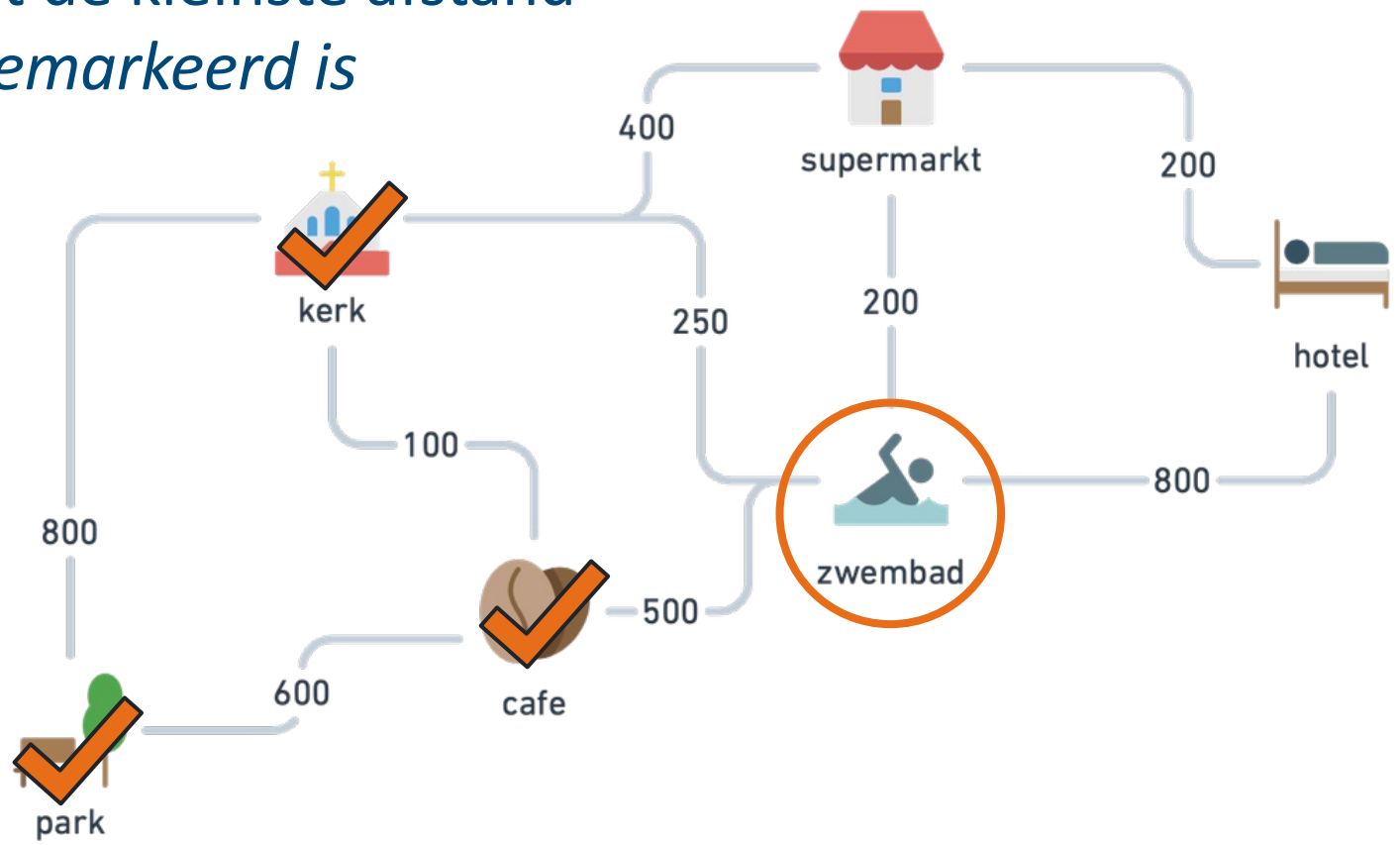


4. Markeer als "klaar"

Dijkstra's algoritme

2. Selecteer de knoop met de kleinste afstand
   die nog niet als klaar gemarkeerd is

✓?	Knopen	Afstand	Vorige
✓	Park	0	-
✓	Kerk	700	Café
✓	Café	600	Park
	Supermarkt	1100	Kerk
	Zwembad	950	Kerk
	Hotel	∞	?

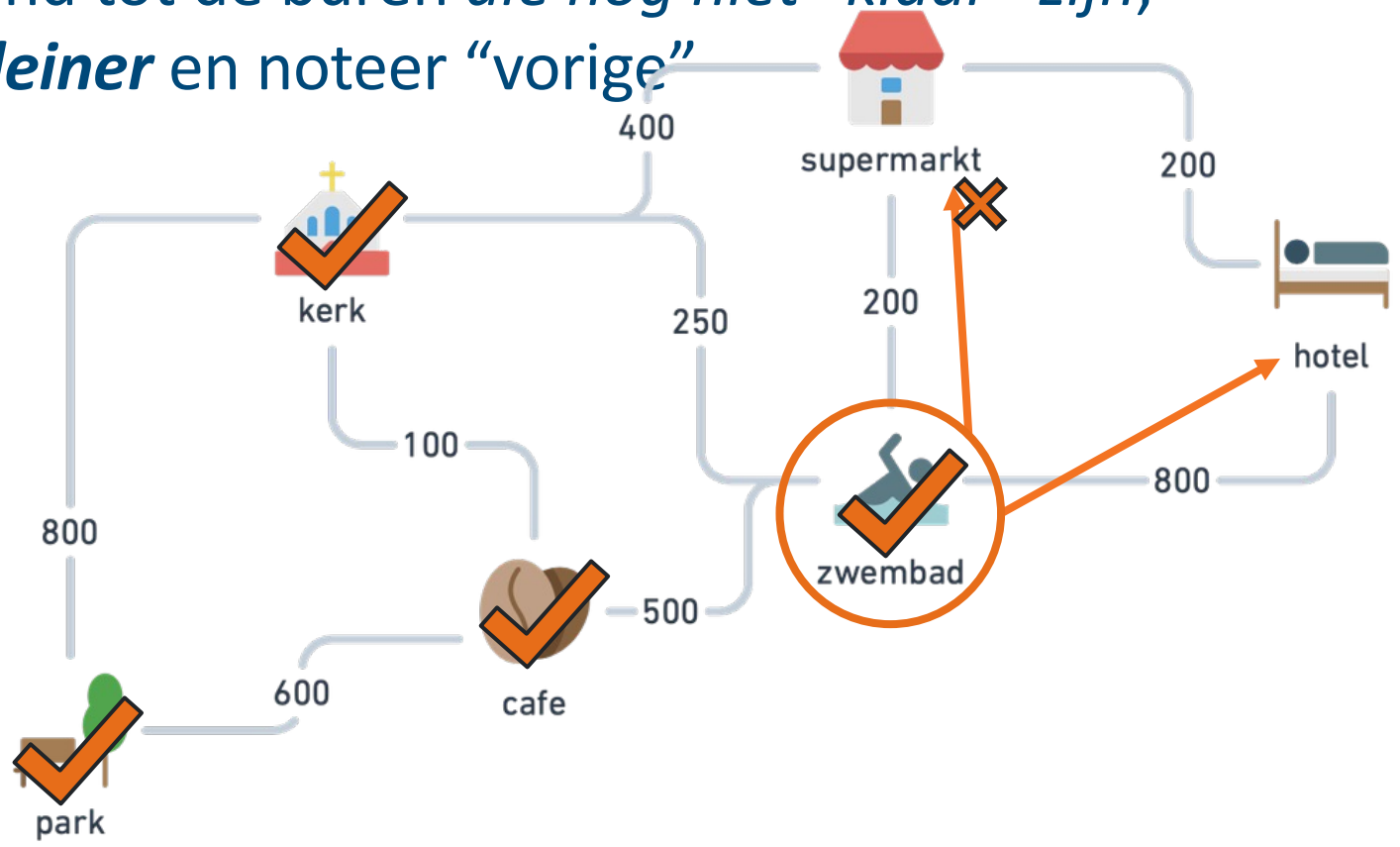


Dijkstra's algoritme

3. Bereken de *totale* afstand tot de buren *die nog niet "klaar" zijn*, update de tabel ***mits kleiner*** en noteer "vorige"

✓?	Knopen	Afstand	Vorige
✓	Park	0	-
✓	Kerk	700	Café
✓	Café	600	Park
	Supermarkt	1100	Kerk
✓	Zwembad	950	Kerk
	Hotel	1750	Zwembad

4. Markeer als "klaar"

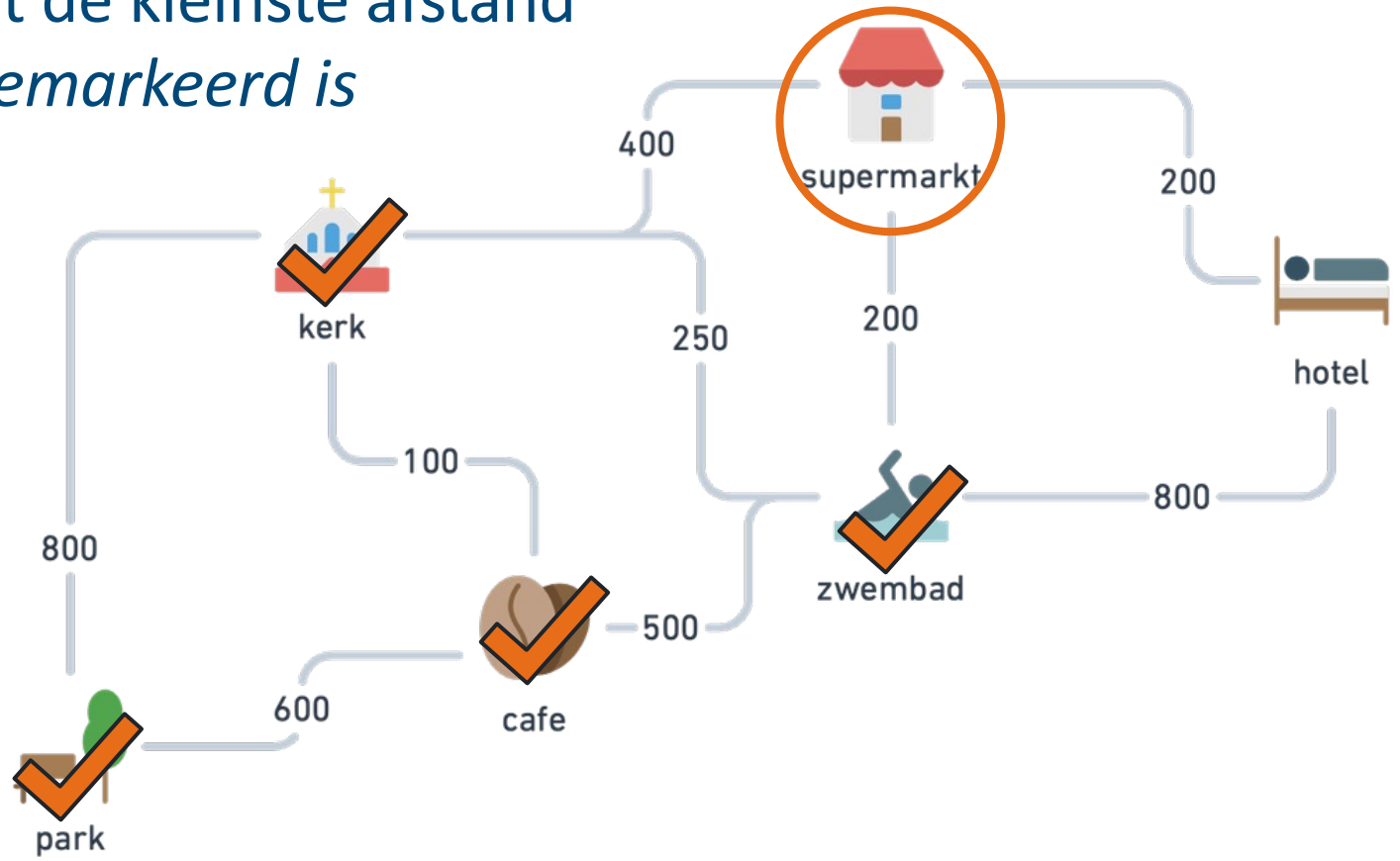


FUNDAMENT INFORMATICA

Dijkstra's algoritme

2. Selecteer de knoop met de kleinste afstand
   die nog niet als klaar gemarkeerd is

✓?	Knopen	Afstand	Vorige
✓	Park	0	-
✓	Kerk	700	Café
✓	Café	600	Park
	Supermarkt	1100	Kerk
✓	Zwembad	950	Kerk
	Hotel	1750	Zwembad

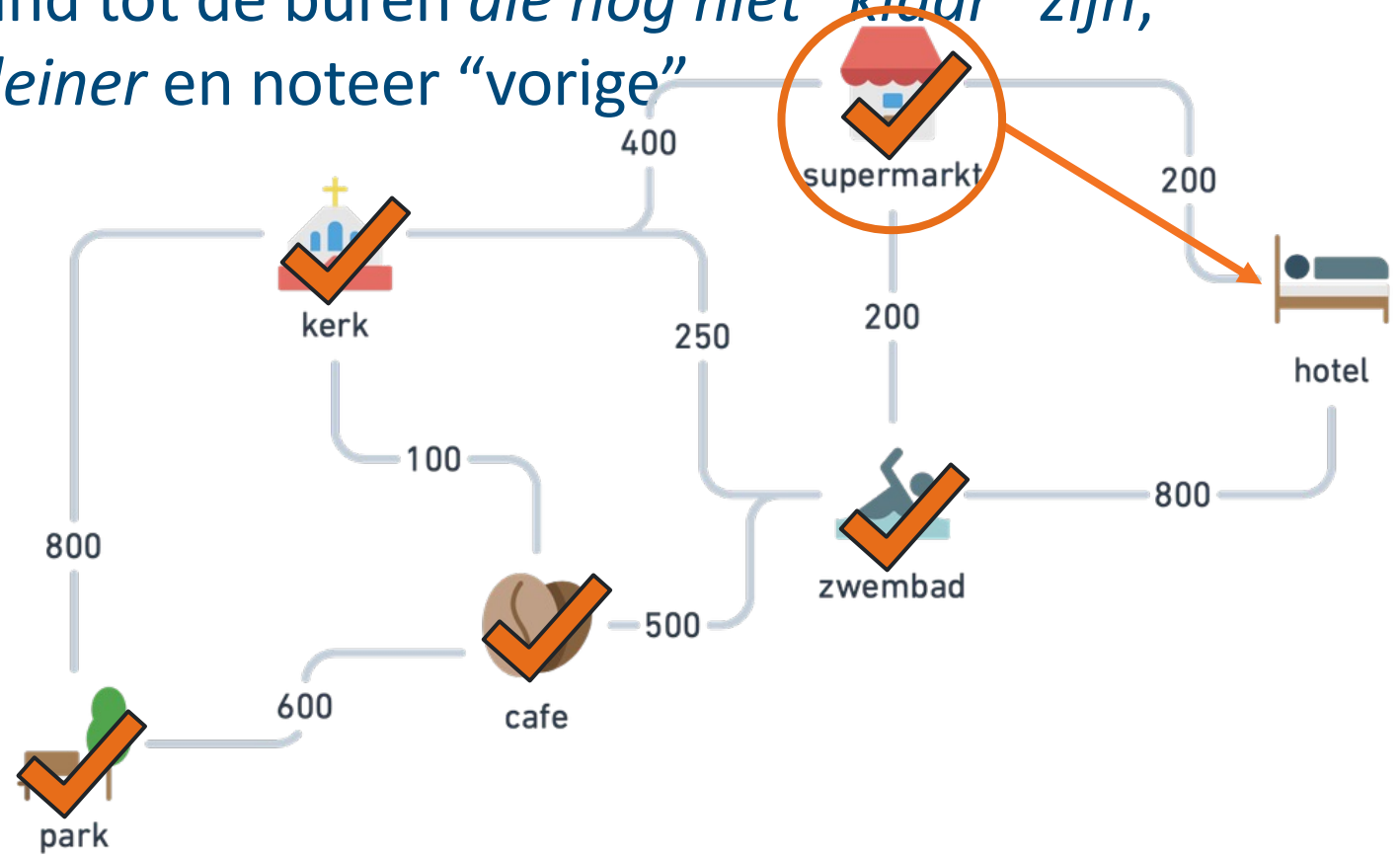


Dijkstra's algoritme

3. Bereken de *totale* afstand tot de buren *die nog niet "klaar" zijn*, update de tabel *mits kleiner* en noteer "vorige"

✓?	Knopen	Afstand	Vorige
✓	Park	0	-
✓	Kerk	700	Café
✓	Café	600	Park
✓	Supermarkt	1100	Kerk
✓	Zwembad	950	Kerk
	Hotel	1300	Supermarkt

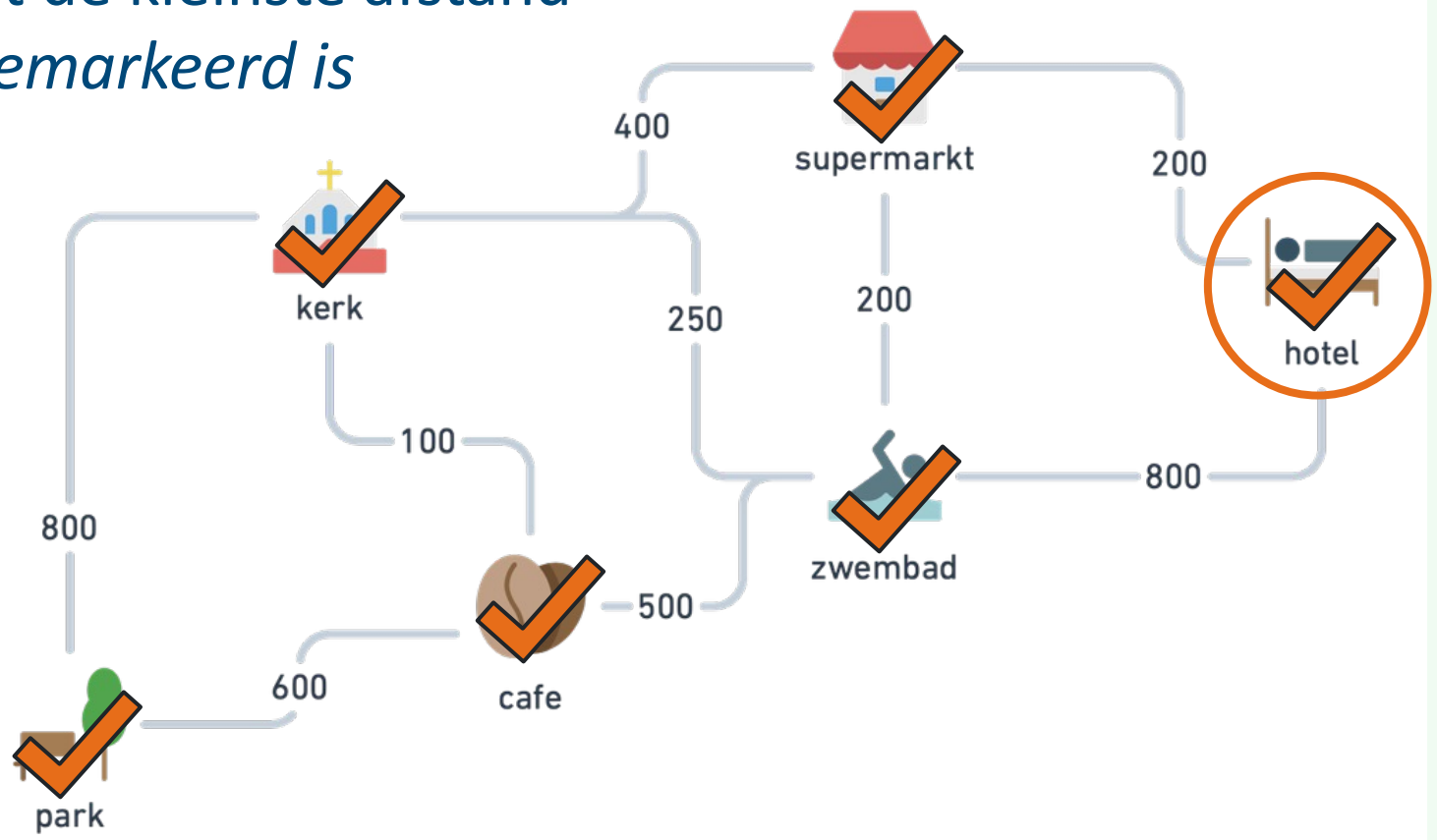
4. Markeer als "klaar"



Dijkstra's algoritme

2. Selecteer de knoop met de kleinste afstand
   die nog niet als klaar gemarkeerd is

✓?	Knopen	Afstand	Vorige
✓	Park	0	-
✓	Kerk	700	Café
✓	Café	600	Park
✓	Supermarkt	1100	Kerk
✓	Zwembad	950	Kerk
✓	Hotel	1300	Supermarkt



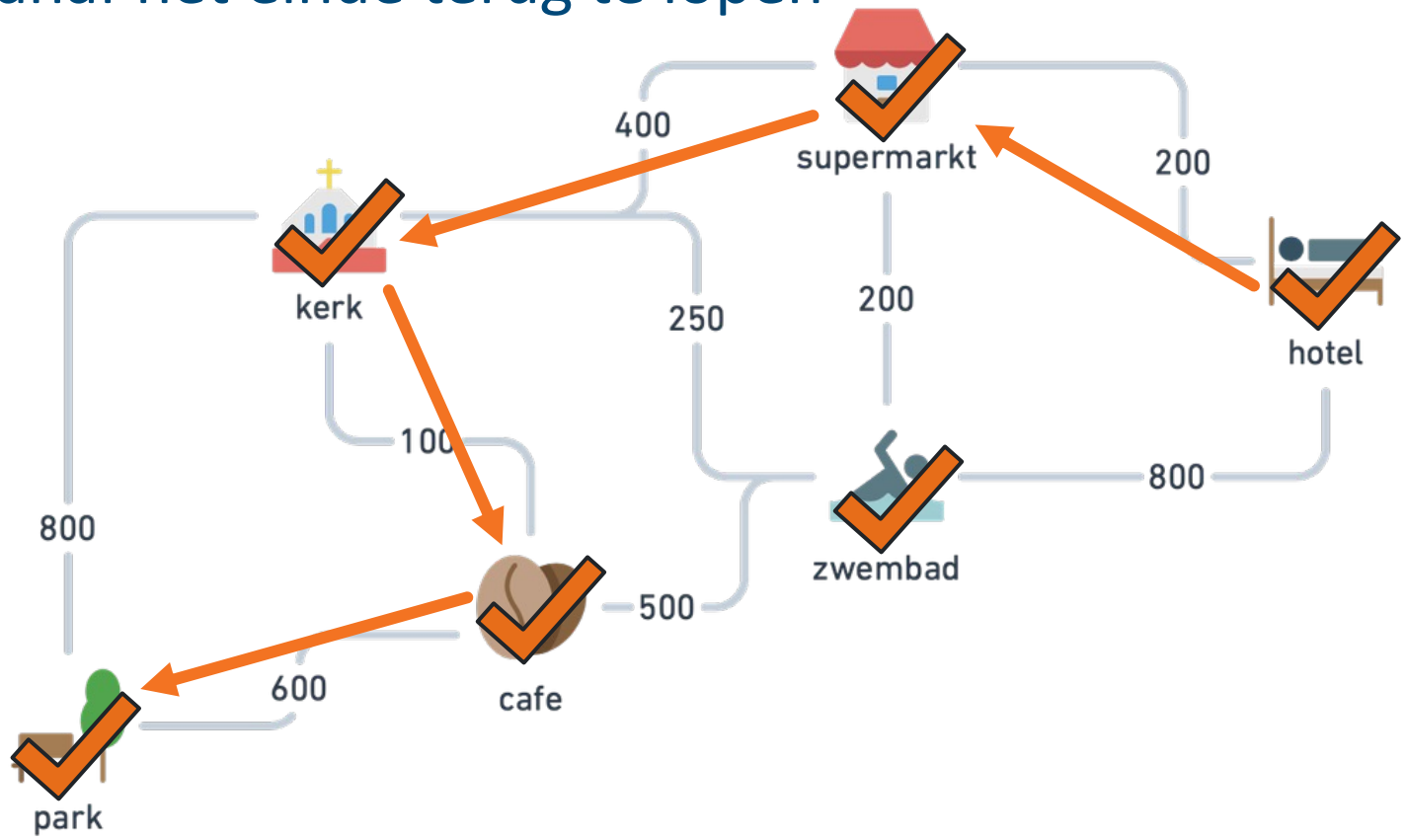
* Als dit het **einddoel** is, stop direct de herhaling

Dijkstra’s algoritme

6. Bepaal de route door vanaf het einde terug te lopen

✓?	Knopen	Afstand	Vorige
✓	Park	0	-
✓	Kerk	700	Café
✓	Café	600	Park
✓	Supermarkt	1100	Kerk
✓	Zwembad	950	Kerk
✓	Hotel	1300	Supermarkt

Park > Café > Kerk > Supermarkt > Hotel



FUNDAMENT INFORMATICA

Dijkstra's algoritme – samengevat

V = aantal knopen (Engels: *vertices*)
E = aantal zijden (Engels: *edges*)

1. Maak een lijst van alle knopen, noteer afstand 0 bij begin, afstand ∞ bij de rest
2. Selecteer de knoop met de kleinste afstand *die nog niet als klaar gemarkeerd is*
   - Als dit het einddoel is, stop de herhaling
3. Voor alle buren die nog niet klaar zijn:
   - Bereken de *totale* afstand
   - *Mits nieuwe afstand kleiner*: noteer afstand en "vorige"
4. Markeer de geselecteerde knoop als klaar
5. Herhaal vanaf stap 2
6. Bepaal de route door vanaf het einde terug te lopen

✓?	Knopen	Afstand	Vorige
	Park	0	-
	Kerk	∞	?
	Café	∞	?
	Supermarkt	∞	?
	Zwembad	∞	?
	Hotel	∞	?

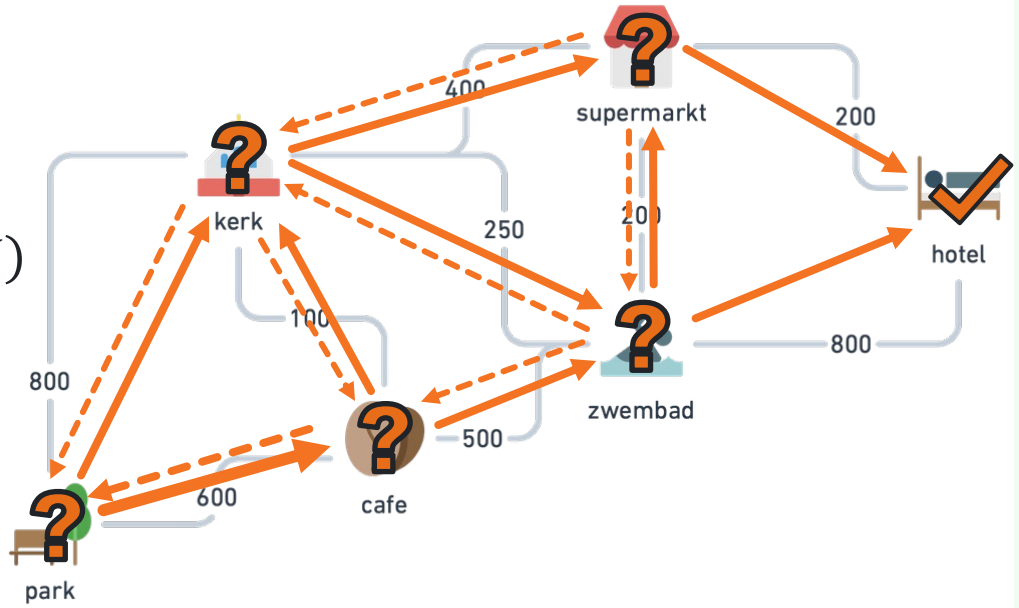
✓?	Knopen	Afstand	Vorige
✓	Park	0	-
	Kerk	700	Café
✓	Café	600	Park
	Supermarkt	∞	?
	Zwembad	1100	Café
	Hotel	∞	?

Orde van Dijkstra's algoritme

V = aantal knopen (Engels: *vertices*)
E = aantal zijden (Engels: *edges*)

- Worstcasescenario:
  - Alle verbindingen aflopen: $O(E)$
  - Voor iedere verbinding, eventueel een getal aanpassen in de tabel: $O(1)$
    - Dus: $O(E) \cdot O(1) = \boldsymbol{O(E)}$
  - Alle knopen aflopen: $O(V)$
  - en daarna *ongeordend zoeken* door de lijst met alle knopen naar de kleinste afstand: $O(V)$
    - Dus: $O(V) \cdot O(V) = \boldsymbol{O(V^2)}$
- Dus: $\boldsymbol{O(V^2 + E)}$

✓?	Knopen	Afstand	Vorige
✓	Park	0	-
✓	Kerk	700	Café
✓	Café	600	Park
	Supermarkt	1100	Kerk
	Zwembad	950	Kerk
	Hotel	∞	?



FUNDAMENT INFORMATICA

Extra slides

Module G. Algoritmiek

FUNDAMENT INFORMATICA

Structuur

Fundament / SLO	Inhoud
1. Kwaliteitsaspecten	Wat maakt het ene algoritme ‘beter’ dan een ander algoritme?
2. Looptijden	Over groeigedrag en orde-analyses.
3. Grafen	Over algoritmen in de ‘graaf’-datastructuur, zoals Dijkstra’s.
4. Logica	Wetten van de logica. If-condities vereenvoudigen.

Materiaal	SLO	Fundament
Lesmateriaal	✓ Notion.so	✓ online
Werkvormen	✓ gevarieerd	✓ overgenomen, met instant-feedback, en sterk uitgebreid
Powerpoints	x	✓
Uitlegvideo’s	x	✓ PLUS-licentie
Planner	x	✓
Toetsmateriaal	x	✓ grote collectie

FUNDAMENT
INFORMATICA

Planning (van de Fundament implementatie)

Module	SLU (kort)	SLU (aanbevolen)	SLU (max)
G1. Kwaliteits-aspecten	30 minuten	1 lesuur	1 lesuur
G2. Looptijden	8 (2 wk)	16 (4 wk)	20
G3. Grafen	7 (2 wk)	12 (3 wk)	20
G4. Logica	4 (1 wk)	8 (2 wk)	10 of met extra module: 25+

FUNDAMENT INFORMATICA

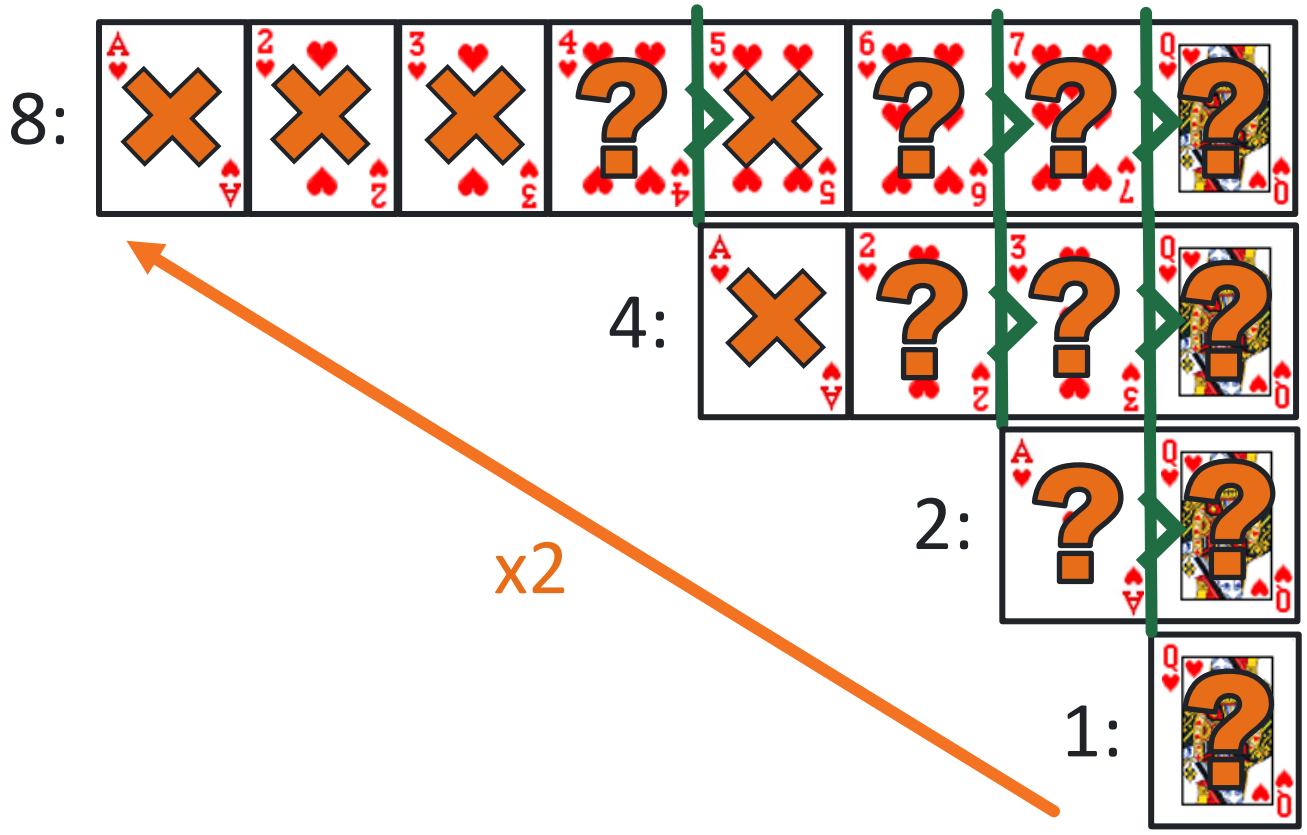
** We hebben de weekplanning ingeschat op basis van 1 week = 4 SLU, wat volgens SLO het aantal SLU is voor het vak informatica bij een gelijke verdeling van de SLU over alle leerjaren.*

Orde van binair zoeken

$$raden = \log_2(n) + 1$$

$$n_{kaarten} = 2^{raden-1}$$

- Hoe vaak moeten we raden om ❤️V te vinden?



4x $n = 2^3$

3x $n = 2^2$

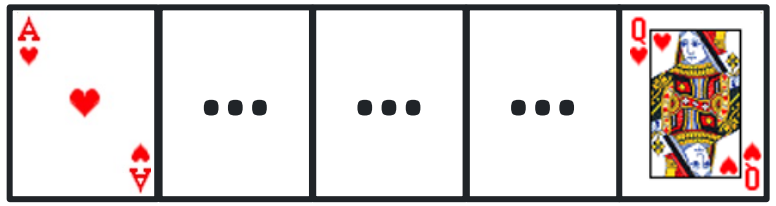
2x $n = 2^1$

1x $n = 2^0$

+1

Orde van binair zoeken

- Hoe vaak moeten we raden om ❤️V te vinden?
- In *het algemene geval* geldt:

n:  $\log_2(n) + 1$

- $O(\log_2(n) + 1)$
- Maar orde zegt alleen iets over “hele grote *n*”, dus +1 is verwaarloosbaar:
- $\boldsymbol{O(\log_2(n))}$