ALGORITMES & DATASTUCTUREN

Joost Visser (jwjh.visser@avans.nl)

Academie voor Deeltijd

Laatst geüpdatet: 29-04-2021

Les opnemen

Deze les wordt opgenomen, heeft hier iemand bezwaar tegen?

Les van vandaag

1. **Vakintroductie**
2. Wat zijn algoritmes en datastructuren?
3. Voorbeeld 1: Linear en binair zoeken
4. Tijds- en geheugencomplexiteit
5. Voorbeeld 2: Sorteren

Les van vandaag

1. **Vakintroductie**
    1. Even voorstellen
    2. Leeruitkomsten
    3. Toetscriteria
    4. Bronnen
2. Wat zijn algoritmes en datastructuren?
3. Voorbeeld 1: Linear en binair zoeken
4. Tijds- en geheugencomplexiteit
5. Voorbeeld 2: Sorteren

1. Vakintroductie

Even voorstellen

- *Naam:* Joost Visser
- *Leeftijd:* 25 jaar
- *Studie:* Computer Science master aan de TU/e.
- *Werkervaring*: Full-time algoritme developer bij SmartQare.
- *Specialisaties:* AI en algoritmes.

Af en toe heb ik meegedaan aan Europese algoritmewedstrijden (in Zweden en Engeland). Eén keer zelfs eerste van Nederland geworden.

Contact? → Kan via Teams, Email of WhatsApp.

Vragen tijdens de les? → Stel ze in de Teamschat

1. Vakintroductie

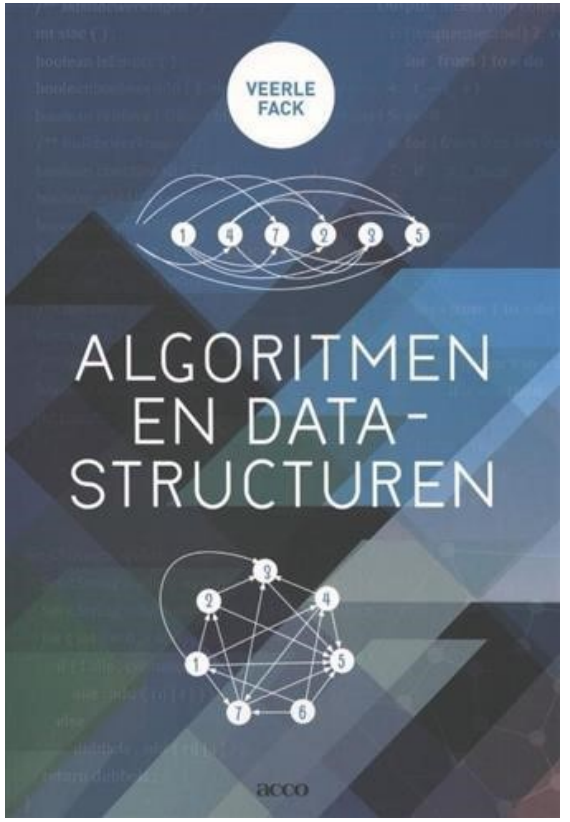
Leeruitkomsten

1. Je kan het juiste datastructuur kiezen bij specifieke situaties.
   - Deze komen uit het Collections Framework in Java.
2. Je bepaalt de rekencomplexiteit en geheugencomplexiteit van een algoritme.
   - Bijvoorbeeld: de algoritme heeft een *worst-case* uitvoeringstijd van
3. Je kan je eigen datastructuur ontwikkelen met generieke typen.
   - `PlatypusList<Platypus> platty = new PlatypusList<>();`
4. Je begrijpt standaard sorteeralgoritmes, binaire bomen en recursie.

1. Vakintroductie

Bronnen

1. Deze slides
2. LinkedIn Learning: Introductions to Data Structures & Algorithms
3. TutorialPoint
4. Algoritmen en Datastructuren, geschreven door Veerle Fack
5. Internet! Stackexchange
6. Kattis



1. Vakintroductie

Beoordeling: Toetscriteria

1. De meest voorkomende constructies van programmeertalen begrijpen en kunnen toepassen.
2. Bepalen van rekencomplexiteit en geheugencomplexiteit van algoritmes.
3. Begrijpen en toepassen van recursie.
4. Abstracte datastructuren en algoritmen kunnen kiezen en toepassen in praktische situaties (array, list, stack, queue, set, map, tree, hashing)
5. Generieke datatypen en methodes begrijpen en toepassen

1. Vakintroductie

Toets

Je wordt volledig beoordeeld aan de hand van de toets.

De toets zal uit twee delen bestaan:

1. Theorievragen (50%)
2. Praktijkopdracht (50%)

De toets van vorig jaar staat online als voorbeeld.

In tegenstelling tot vorig jaar kan de toets dit jaar **alleen in Java** worden gemaakt.

De exacte details van de toets worden in les 4 behandeld.

1. Vakintroductie

Lesplan

Les	Datum	Topic	Inhoud
1	4 mei 2021	Introductie	Inleiding, algoritme complexiteit, data-structuren, Linear zoeken, binair zoeken, Sorteren 1
2	11 mei 2021	Datastructuren 1	Abstracte datatypen, Array, List, Set, Map
3	18 mei 2021	Datastructuren 2	Stack, Heap, Deque, Priority Queue, Generieke datatypen
4	25 mei 2021	Algoritmes	Sorteren 2, recursie, binaire bomen, proeftoets

Vragen?

Les van vandaag

1. Vakintroductie
2. **Algoritmes en datastructuren**
   1. Wat is een algoritme?
   2. Wat is een datastructuur?
   3. Java datastructuren
3. Voorbeeld 1: Linear en binair zoeken
4. Tijds- en geheugencomplexiteit
5. Voorbeeld 2: Sorteren

2. Algoritmes en datastructuren

Wat is een algoritme?

Een algoritme is een recept voor het oplossen van een probleem, stap voor stap.

Verzin een algoritme voor de volgende problemen:

1. Gegeven twee nummers, tel ze bij elkaar op.
2. Zit de volgende getal in een gegeven lijst?
3. Wat is de snelste route van mijn huis naar Avans?

Vaak wil je iets van data tijdelijk opslaan voor je calculaties. Hiervoor kan je datastructuren gebruiken; een structuur om je data op te slaan.

2. Algoritmes en datastructuren

Wat is een datastructuur?

Vaak wil je iets van data tijdelijk opslaan voor je calculaties. Hiervoor kan je datastructuren gebruiken; een structuur om je data op te slaan.

Het simpelste is bijvoorbeeld om een lijst met getallen bij te houden, ook wel een array genoemd.

```
int [] inputNumbers = new int[20];
```

Echter zijn er ook andere mogelijkheden om een lijst met getallen op te slaan, zoals in een ArrayList, HashSet of zelfs een Binary Search tree.

Welke je kiest hangt af ondere andere af van (1) hoeveel ruimte je hebt, (2) hoe snel je bepaalde acties wilt doen en (3) welke acties je wilt doen.

2. Algoritmes en datastructuren

Wat is een datastructuur?

Vaak wil je iets van data tijdelijk opslaan voor je calculaties. Hiervoor kan je datastructuren gebruiken; een structuur om je data op te slaan.

Het simpelste is bijvoorbeeld om een lijst met getallen bij te houden, ook wel een array genoemd.

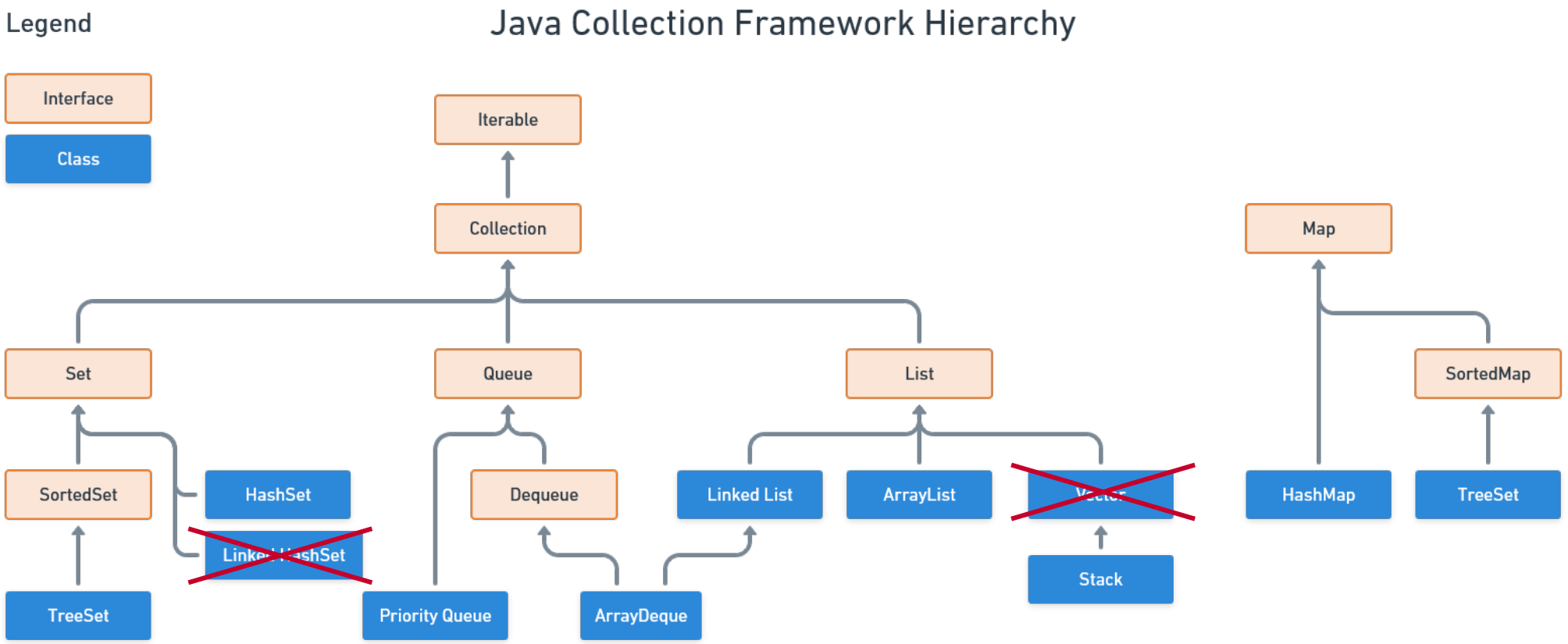
```
int [] inputNumbers = new int[20];
```

In principe kan elke datastructuur de volgende CRUD acties doen:

1. Create → Data maken en toevoegen
2. Read → Data uitlezen
3. Update → Data aanpassen
4. Delete → Data verwijderen

2. Algoritmes en datastructuren

Java datastructuren



Les van vandaag

1. Vakintroductie
2. Wat zijn algoritmes en datastructuren?
3. **Voorbeeld 1: Zoeken**
    1. Wat is het zoekprobleem?
    2. Linear zoeken
    3. Binair zoeken
4. Tijds- en geheugencomplexiteit
5. Voorbeeld 2: Sorteren

3. Voorbeeld 1: Zoeken

Wat is het zoekprobleem?

Zoekprobleem: vind een waarde, zoeksleutel, in een verzameling gegevens

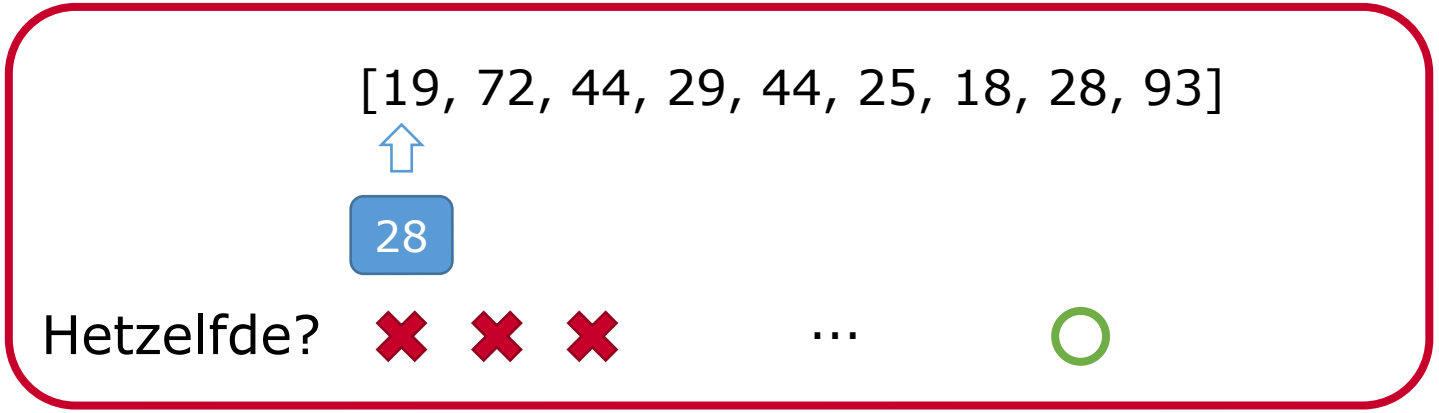
Voorbeelden:

1. Vind 28 in [19, 72, 44, 29, 44, 25, 18, 28, 93]
2. Vind 72 in [18, 19, 25, 28, 29, 44, 44, 72, 93]
3. Vind Henk in [“Piet”, “Jan”, “Katja”, “Annabel”, “Frans”, “Lieke”]

3. Voorbeeld 1: Zoeken

Linear zoeken

1. Vind 28 in [19, 72, 44, 29, 44, 25, 18, 28, 93]



Hoe *snel* is dit algoritme?

… hoe meet je de snelheid van een algoritme?

3. Voorbeeld 1: Zoeken

Linear zoeken

1. Vind 28 in [19, 72, 44, 29, 44, 25, 18, 28, 93]

Hoe *snel* is dit algoritme?

… hoe meet je de snelheid van een algoritme?

1. In de *worst-case* scenario, hoe snel is het algoritme?
2. Neem aan dat één comparison '1' tijdseenheid kost.

$$T = 9$$

3. Wat nou als de lijst een aritraire lengte heeft?

$$T(n) = n$$

3. Voorbeeld 1: Zoeken

Linear zoeken

1. Vind 28 in [19, 72, 44, 29, 44, 25, 18, 28, 93]

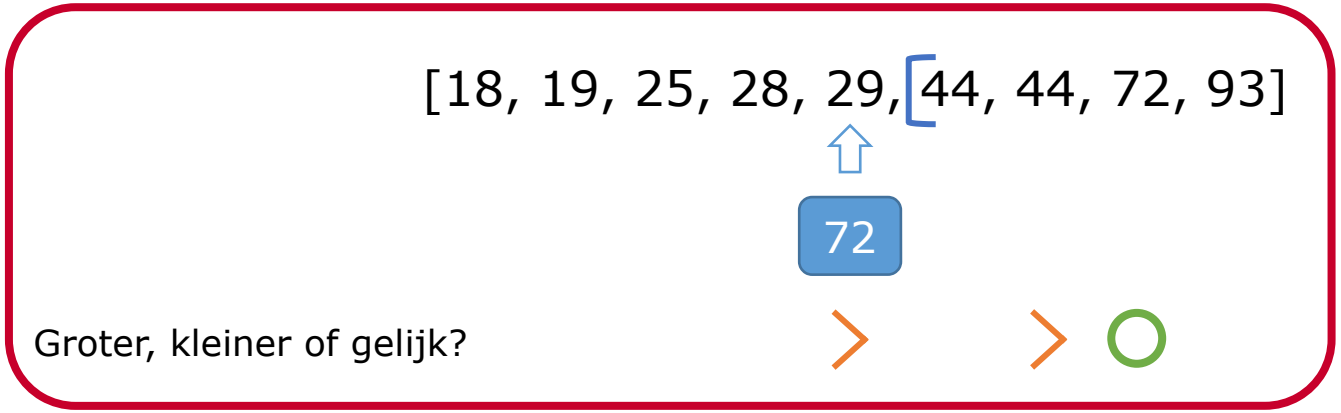
```java
/**
 * Finds a number in an unordered list.
 * Returns index or -1 if not found.
 */
public int findNumber(int list[], int number) {
    for (int i = 0; i < list.length; i++) {
        if (list[i] == number) {
            return i;
        }
    }
    return -1;
}
```

3. Voorbeeld 1: Zoeken

Binair zoeken

2. Vind 72 in [18, 19, 25, 28, 29, 44, 44, 72, 93]

Iemand een idee voor een snellere algoritme dan linear zoeken?



Hoe weten we of het getal er niet in zit?

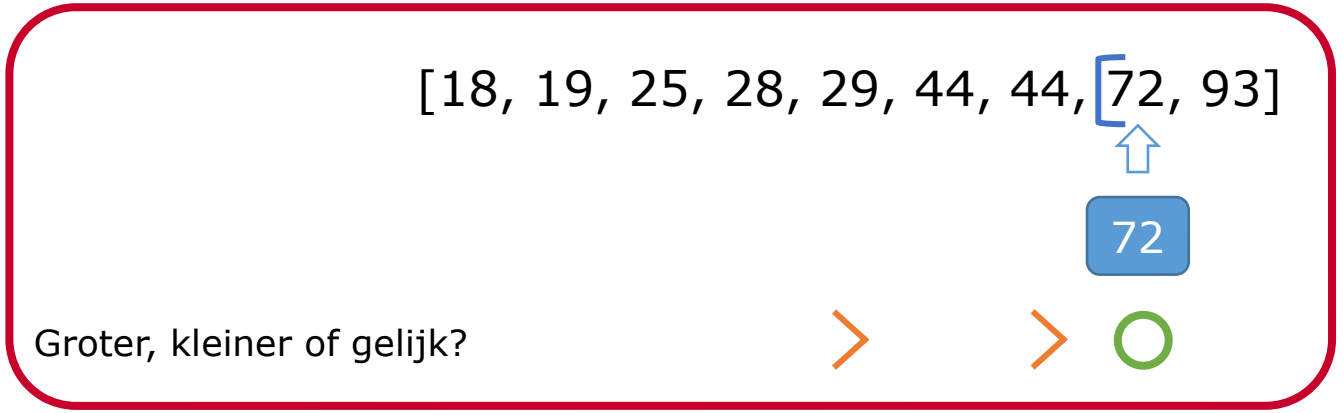
In de *worst-case* scenario, hoe snel is dit algoritme?

3. Voorbeeld 1: Zoeken

Binair zoeken

2. Vind 72 in [18, 19, 25, 28, 29, 44, 44, 72, 93]

Iemand een idee voor een snellere algoritme dan linear zoeken?



We halveren elke keer onze *range*, zodra die 1 is weten we het antwoord.

$$T(n) = \log_2 n$$

Hoe weten we of het getal er niet in zit?

In de *worst-case* scenario, hoe snel is dit algoritme?

Note: dit is geen formeel bewijs.
Formele bewijzen vallen buiten de scope van dit vak.
Bekijk het boek voor een formelere bewijs.

3. Voorbeeld 1: Zoeken

Binair zoeken

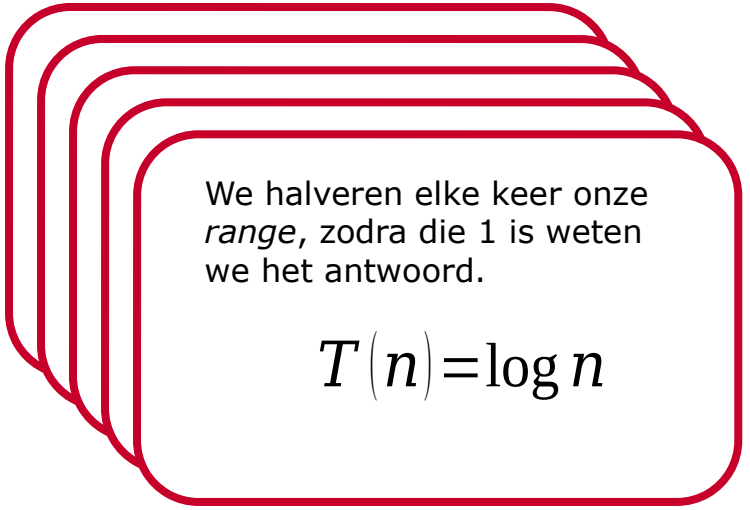
2. Vind 72 in [18, 19, 25, 28, 29, 44, 44, 72, 93]

Logaritme intermezzo!

$$a^x = b \Leftrightarrow b = \log_a x$$

Omdat logaritme met *base 2* enorm vaak voorkomen in Computer Science, laten we de 2 in weg.

De running time wordt dan



Note: dit is geen formeel bewijs.
Formele bewijzen vallen buiten de scope van dit vak.
Bekijk het boek voor een formelere bewijs.

3. Voorbeeld 1: Zoeken

Binair zoeken

2. Vind 72 in [18, 19, 25, 28, 29, 44, 44, 72, 93]

Hoeveel scheelt het qua snelheid?

n	Linear zoeken	Binair zoeken
10	10 microseconde	3.32 microseconde
1 000	1 milliseconde	9.97 microseconde
1 000 000	1 seconde	19.93 microseconde
1 000 000 000	16.67 minuten	29.90 microseconde
1 000 000 000 000	11.57 **dagen**	39.86 microseconde
1e18	31.71 miljard jaar	59.79 microseconde

Handige praktijktip: een computer doet ongeveer 1 000 000 computations per seconde

Les van vandaag

1. Vakintroductie
2. Algoritmes en datastructuren
3. Voorbeeld 1: Linear en binair zoeken
4. **Tijds- en geheugencomplexiteit**
    1. Tijdscomplexiteit
    2. Big-O notatie
    3. Big-O notatie in de praktijk
    4. en
    5. Geheugencomplexiteit
5. Voorbeeld 2: Sorteren

4. Tijds- en geheugencomplexiteit

Tijdscomplexiteit

Hoe snel is een algoritme?

- Lineaire search →
- Binaire search →

Dit was **met** de aanname dat één operatie ‘1’ tijd kostte.

Hoe kunnen we dit wat wiskundig netter oplossen?

4. Tijds- en geheugencomplexiteit

Big O-notatie

Met een asymptotische analyze!

Stel we hebben een running time functie van:

Voor is

- Waarvan komt door de

We zijn vaak alleen geïntereseerd in de hoogste macht van , omdat voor een hogere deze vaak de volledige running time bepaalt.

$$T(n) = O(n^3)$$

4. Tijds- en geheugencomplexiteit

Big O-notatie

$$T(n) = O(n^3)$$

De bovengrens van de orde van toename is .

- Als er extra samples komen, dan groeit de running time met operaties.

De wiskundige definitie is dat er een constante bestaat zodat:

- Voor die groot genoeg is:

Een aantal voorbeelden:

1. is sneller dan (voor grote , worst-case)
2. is sneller dan (voor grote , worst-case)
3. is sneller dan (voor grote , worst-case)

4. Tijds- en geheugencomplexiteit

Big O-notatie

Waar of niet waar?

1. Een algoritme is **altijd** langzamer dan .
2. Orde is langzamer dan voor grote in de worst-case.
3. Met samples kan je makkelijk een algoritme gebruiken.

Eigenlijk in plaats van ; daar komen we straks op.

4. Tijds- en geheugencomplexiteit

Big-O notatie in de praktijk

Hoe snel is een algoritme?

- Lineaire search →
- Binaire search →

Dit was **met** de aanname dat één operatie ‘1’ tijd kostte.

operatie

Hoe kunnen we dit wat wiskundig netter oplossen?

$$T(n) = O(n)$$

4. Tijds- en geheugencomplexiteit

Big-O notatie in de praktijk

Hoe bereken je de complexiteit van Lineair Zoeken?

$O(n)$
$O(1)$
$O(1)$

```java
/**
 * Finds a number in an unordered list.
 * Returns index or -1 if not found.
 */
public int findNumber(int list[], int number) {
    for (int i = 0; i < list.length; i++) {
        if (list[i] == number) {
            return i;
        }
    }
    return -1;
}
```

$$T(n) = O(n) \cdot O(1) + O(1) = O(n)$$

4. Tijds- en geheugencomplexiteit

Big-O notatie in de praktijk

Hoe snel is het volgende algoritme?

```
public int findNumber(int list[], int number) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            step(i, j);
        }
    }
    return -1;
}
```

$$T(n) = O(n^2)$$

4. Tijds- en geheugencomplexiteit

Big-O notatie in de praktijk — oefeningen

```
for (int i = 0; i < n; i++) {
    for (int j = 0; j < 100; j++) {
        step(i, j);
    }
}

for (int i = 0; i < n; i++) {
    for (int j = 0; j < i; j++) {
        step(i, j);
    }
}

for (int i = 0; i < n; i++) {
    for (int j = 0; j < K; j++) {
        step(i, j);
    }
}
```

Nog vragen?

4. Tijds- en geheugencomplexiteit

en

Bij asympotische tijdsanalyze zie je ook af en toe en naast :

1. Orde → Bovengrens van de orde van toename is
   - "Het is maximaal orde "
2. Orde → Ondergrens van de orde van toename is
   - "Het is minimaal orde "
3. Orde → Boven- en ondergrens van de orde van toename is
   - "Het algoritme heeft orde "

4. Tijds- en geheugencomplexiteit

Geheugencomplexiteit

Net als dat tijd een belangerijke resource is, kan geheugen soms ook een probleem leveren.

We kunnen op eenzelfde manier de tijd noteren:

$$G(n) = O(n)$$

Als we bijvoorbeeld een array van getallen willen bijhouden, dan is ons geheugenverbruik (en , maar werken met big- is makkelijker).

4. Tijds- en geheugencomplexiteit

Geheugencomplexiteit

```java
/**
 * Finds a number in an unordered list.
 * Returns index or -1 if not found.
 */
public int findNumber(int list[], int number) {
    for (int i = 0; i < list.length; i++) {
        if (list[i] == number) {
            return i;
        }
    }
    return -1;
}
```

Wat is het geheugengebruik van dit algoritme?

$$G(n) = O(n)$$

Wat is het extra geheugengebruik van dit algoritme?

$$G(n) = O(1)$$

4. Tijds- en geheugencomplexiteit

Samenvatting

Om de snelheid van een algoritme te berekenen, gebruiken we de big-O notatie, wat de bovengrens van de orde van toename is:

1. Relatief makkelijke manier om de algoritmesnelheid in te schatten
2. Kan schatten hoe groot , de verzamelingsgrootte, mag zijn
3. Kan snelheid tussen algoritmes vergelijken

$$T(n) = O(\log n)$$

Voor een aantal voorbeelden hebben we deze snelheid bepaald.

Het geheugengebruik kan op eenzelfde manier berekend worden

$$G(n) = O(n)$$

Les van vandaag

1. Vakintroductie
2. Algoritmes en datastructuren
3. Voorbeeld 1: Linear en binair zoeken
4. Tijds- en geheugencomplexiteit
5. **Voorbeeld 2: Sorteren**
    1. Het sorteerprobleem
    2. Selection Sort
    3. Counting Sort

5. Voorbeeld 2: Sorteren

Wat is het sorteerprobleem?

Sorteerprobleem: Herschik een gegeven array zodat de elementen in stijgende (of dalende) volgorde staan.

Voorbeelden:

1. Ongesorteerd: [19, 72, 44, 29, 44, 25, 18, 28, 93]
   Gesorteerd: [18, 19, 25, 28, 29, 44, 44, 72, 93]
2. Ongesorteerd: [“Piet”, “Jan”, “Katja”, “Annabel”, “Frans”, “Lieke”]
   Gesorteerd: [“Annabel”, “Frans”, “Jan”, “Katja”, “Lieke”, “Piet”]

Waarom analyseren we dit?

3. Makkelijk, duidelijk voorbeeld voor het illustreren van tijds- en geheugencomplexiteit.
4. Goed te zien wanneer een antwoord correct is.
5. Goed om algoritmes te vergelijken met elkaar.
6. In sommige situaties is het sneller om je eigen algoritmes te gebruiken.

Voorbeeld 2: Sorteren

Selection Sort

Input: [19, 72, 44, 29, 44, 25, 18, 28, 93]

Selection Sort v1

```
1. Create new outputArray[n]
2. For i = 0 to n-1:
    1. Find minValue in inputArray
    2. outputArray[i] = minValue
    3. Set minValue in inputArray
       to
3. Return outputArray
```

Uitvoeringstijd? $O(n^2)$

```
1. O(n)
2. O(n)
    1. O(n)
    2. O(1)
    3. O(1)
3. O(1)
```

Extra geheugenverbruik?

$O(n)$

Kan dit beter?

Ja! Inline sorteren.
Extra geheugenverbruik van

Output: [18, 19, 25, 28, 29, 44, 44, 72, 93]

Voorbeeld 2: Sorteren

Selection Sort

Input: [19, 72, 44, 29, 44, 25, 18, 28, 93]

Selection Sort v2

```
1. For i = 0 to n-1:
    1. k = findMinValueIndex(i,n)
    2. Swap(i, k)
2. Return inputArray
```

Zie https://visualgo.net/en/sorting

Uitvoeringstijd? $O(n^2)$

```
1. O(n)
    1. O(n)
    2. O(1)
2. O(1)
```

Extra geheugenverbruik?

$O(1)$

Output: [18, 19, 25, 28, 29, 44, 44, 72, 93]

Voorbeeld 2: Sorteren

Selection Sort

Soortgelijke algoritmes

1. Insertion Sort
   - Sorteer van links naar rechts: eerste twee elementen, dan derde, dan vierde, enzovoorts.

2. Bubble Sort
   - Vergelijkbaar met Selection Sort, maar gebruikt meer swaps in plaats van zoeken naar de min. (Gaat ook uit van max → min sorteren)

Zie https://visualgo.net/en/sorting voor een visualisaties.

Voorbeeld 2: Sorteren

Counting Sort

Input: [9, 2, 4, 9, 4, 5, 8, 8, 3]

Constraint: data zit tussen en . ()

Counting Sort

```
1. int[k] counts = new int[k]
2. For i = 0 to n-1:
    1. counts[inputArray[i]] ++;
3. i = 0;
4. for j = 0 to k-1:
    1. while (counts[j] > 0)
        1. inputArray[i] = j
        2. i ++;
        3. counts[j] --;
5. Return inputArray
```

Uitvoeringstijd?

```
1. O(1)
2. O(n)
3. O(1)
4. O(n+k)
5. O(1)
```

$O(n+k)$

Extra geheugengebruik?

$O(k)$

Nog vragen?

Output: [2, 3, 4, 4, 5, 8, 8, 9, 9]

4. Voorbeeld 2: Sorteren

Samenvatting

Sorteerprobleem: Herschik een gegeven array zodat de elementen in stijgende (of dalende) volgorde staan.

Selection sort: Van links naar rechts, vind de minimale element en zet het links neer.

- Uitvoeringstijd:
- Extra geheugengebruik:

Counting sort: Tel van 0 tot hoe vaak ze voorkomen.

- Uitvoeringstijd:
- Extra geheugengebruik:
- Werkt alleen als alle getallen tussen de 0 en zitten.

Bekijk https://visualgo.net/en/sorting voor visualisaties van de algoritmes.

Les van vandaag

1. Vakintroductie
2. Algoritmes en datastructuren
3. Voorbeeld 1: Linear en binair zoeken
4. Tijds- en geheugencomplexiteit
5. Voorbeeld 2: Sorteren

Les van vandaag

Samenvatting

Algoritme: recept voor het oplossen van een programmeerprobleem.
Data-structuur: verschillende structuren om je data op te slaan.

Zoekprobleem: vindt een element in een array
1. Linear zoeken → [Gesorteerde en ongesorteerde lijst]
2. Binair zoeken → [Alleen gesorteerde lijst]

Voor de uitvoeringstijd van een algoritme gebruiken we de big-O notatie.
- Orde betekent dat in de worst-case en voor grote de orde van toename is.
- Dit is equivalent aan *hoogste macht* in de uitvoeringstijd functie .

We gebruiken dezelfde notatie voor het geheugengebruik en extra geheugengebruik.

Sorteerprobleem: zet de elementen van een array op volgorde van klein naar groot.
3. Selection Sort:
4. Counting Sort:

Opdrachten

Informatie

Oefenfiles voor deze opdrachten staan op Brightspace.

De opdrachten zijn optioneel, maar zeer aangeraden.
Het antwoord zal later deze week online komen te staan.
Bij twijfel kan je je opgave submitten via Brightspace, dan kijk ik ernaar.

Deze week zijn er twee oefenopdrachten:

1. Ontbinden van priemgetallen
2. Sorteren van vogelbekdieren

Opdrachten

Opdracht 1: Ontbinden van priemgetallen

In encryptie worden vaak grote priemgetallen gebruikt. Er wordt gebruik gemaakt van het problem van ontbinden in factoren van een product van twee grote priemgetallen.

Een algoritme voor ontbinden in factoren:

```
// zoek een priemfactor van n
// return de kleinste factor, of n als n priem is
int zoekFactor(int n) {
    for (int i = 1; i < n; i++)
        if (n % i == 0)
            return i;
    return n;
}
```

Opgave delen:

1. Schrijf de uitvoeringstijd van het huidige algoritme op.
2. De kan beter. Verbeter het algoritme. Wat is nu de uitvoeringstijd?
3. Naast de formele uitvoeringstijd, voeg een teller toe die telt hoe vaak de for-loop uitgevoerd wordt voor en na de verbetering.

Opdrachten

Opdracht 2: Sorteren van vogelbekdieren

Perry vindt het leuk om vogelbekdieren te houden. Er komt een nieuwe lading vogelbekdieren binnen en hij zou ze graag willen sorteren op naam en op lengte.

Implementeer deze functionaliteit met je eigen sorteerfunctie!

Input: array aan Vogelbekdieren [vogelbekdier1, vogelbekdier2, …]

- String vogelbekdier1.naam bevat de naam van het vogelbekdier1
- int vogelbekdier1.lengte bevat de lengte in cm van het vogelbekdier1.

Output (print naar System.out):

- Array van vogelbekdieren, gesorteerd op naam
- Array van vogelbekdieren, gesorteerd op lengte