PROGRAMMEREN 1

LES 7

UCLL HOGESCHOOL

OVERZICHT

- Sorteeralgoritmes
- Bubble sort
- Computationeel denken: abstractie en evaluatie

VORIGE LES

QUIZ

Wat is het verschil tussen een tuple en een lijst?

Een tuple is aanpasbaar, een lijst niet

Een lijst is aanpasbaar, een tuple niet

Een lijst kan duplicaten bevatten, een tuple niet

Elementen in een lijst zijn geordend, in een tuple niet

QUIZ

Gegeven onderstaande code:

```python
lijst = [4, 8, "Stop"]
```

Hoe selecteer je `"Stop"` uit deze lijst?

- lijst.select(2)
- lijst[3]
- lijst.select(3)
- lijst[2]

QUIZ

Gegeven onderstaande code:

```python
lijst = [4, 8, "Stop"]
```

Hoe pas je `8` in de lijst aan naar `"Start"`?

- lijst.change(2, "Start")
- lijst[1] = "Start"
- lijst.change(1, "Start")
- lijst[2] = "Start"

SORTEERALGORITMES

WAT EN WAAROM

SORTEREN IN COMPUTERS

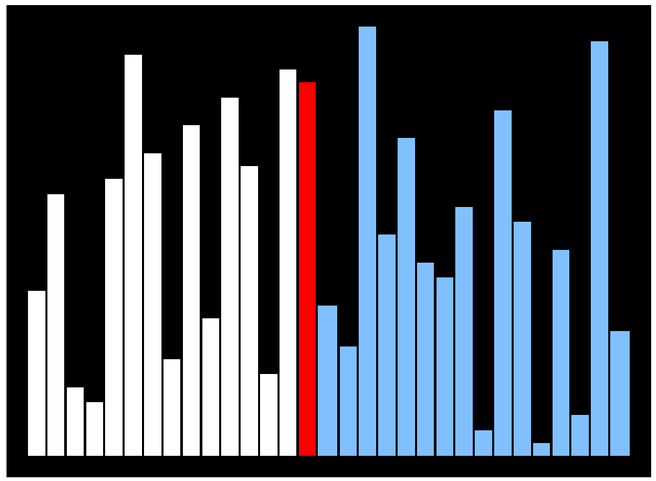
Sorteren is een **van de meest voorkomende opdrachten** die een computer uitvoert. Enkele voorbeelden:

- De bestanden in een folder ordenen op naam, datum van aanpassing, ...
- Bij het opstarten een alfabetisch geordende lijst van gebruikers tonen
- Je meest recent gebruikte programma's tonen
- Examenresultaten in een Excel-bestand ordenen van hoog naar laag

Sorteren dient vaak ook als opstap voor andere taken:

- De zoekfunctie van besturingssystemen werkt sneller als de gegevens op voorhand geordend zijn. Hetzelfde geldt voor gegevensbanken
- Op basis van de geordende examenresultaten kan je kwartielberekenen vlot maken (bv. gemiddelde score van 25% beste leerlingen)

SORTEERALGORITMES



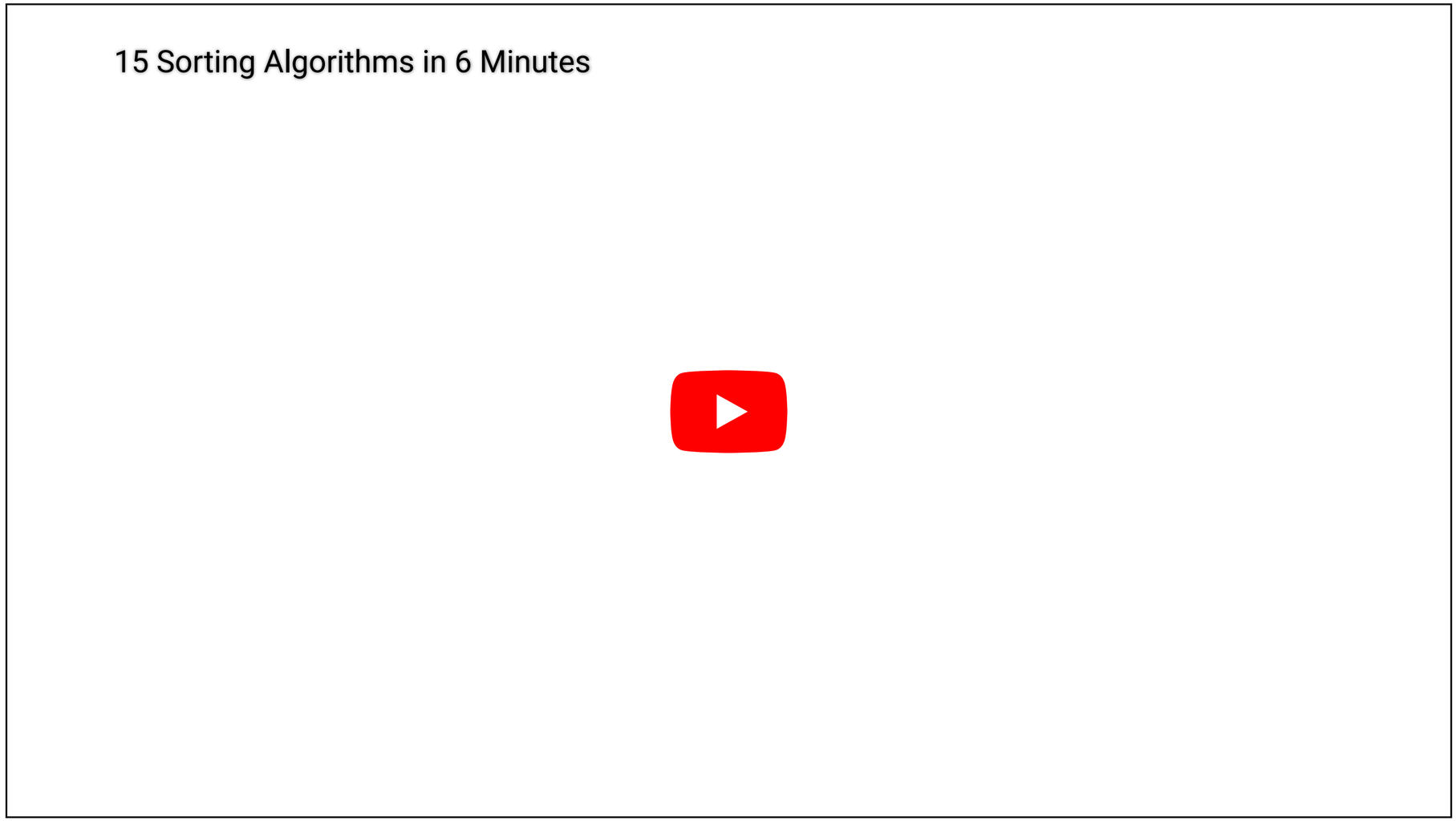
Omdat sorteren zo vaak gebeurt, hebben computerwetenschappers veel onderzoek besteed aan het uitvinden en perfectioneren van **zoekalgoritmes**.

In deze les gaan we één zoekalgoritme in detail bestuderen: **Bubble sort**.

In het project zullen jullie Bubble sort zelf implementeren én aanpassen aan de vereisten van de opdracht.

SORTEERALGORITMES VOORSTELLEN

Een speciale voorstelling: beeld en geluid



Filmpje op YouTube - bubble sort start op 4:01

COMPUTATIONEEL DENKEN

DEEL 4

ABSTRACTIE

DEELCOMPETENTIES CD

Iconen: CoDe-platform KU Leuven
Link naar de slides waarop deze leerstof gebaseerd is

Abstractie	Onnodige details weglaten
Veralgemening	Principes veralgemenen zodat ze breder toepasbaar zijn
Decompositie	Opdeling in kleinere, eenvoudigere problemen
Algoritmisch denken	Opstellen van een stappenplan voor het oplossen van een probleem
Evaluatie	Kritisch bekijken van oplossing

ABSTRACTIE

> *"Het achterwege laten van details die niet van belang zijn om tot een oplossing te komen."*

Veel probleemstellingen bevatten een grote hoeveelheid achtergrondinformatie. Enkele voorbeelden:

- Sommige oefeningen op Dodona vereisen wat wiskundige voorkennis
- Programma's schrijven voor een specifiek domein (archeologie, oncologie geneeskunde, onderwijs, ...)

Die informatie kan nuttig zijn om het probleem te begrijpen. Ze is echter soms **overbodig voor de oplossing**.

ABSTRACTIE: VOORBEELD

Wat hoort niet thuis in het rijtje?

In een restaurant worden borden op elkaar gestapeld en nadien afgewassen.

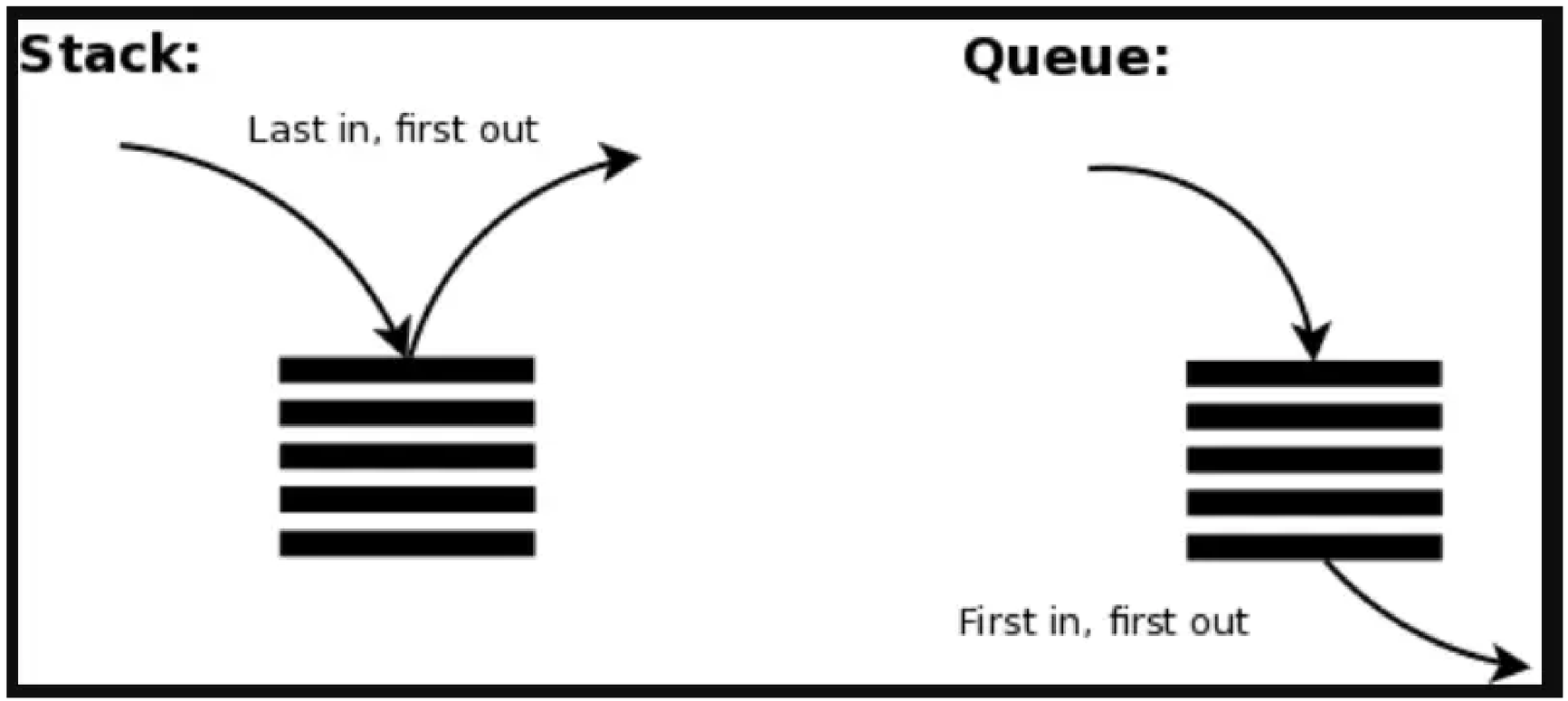
Klanten schuiven achter elkaar aan bij de kassa van een supermarkt.

Een printer voert een reeks printopdrachten uit.

Een reeks auto's staat aan te schuiven voor een rood licht.

ABSTRACTIE: VOORBEELD

Een stapel borden kan je modelleren als een **stack**. De andere opties kan je modelleren als een **queue**.



ABSTRACTIE: VOORBEELD 2

PROBLEEM A

Schrijf een programma dat de historische gegevens van de waarden van cryptomunten analyseert. Gegeven een lijst van historische waardes, berekent het programma de maximale theoretische winst over die periode.

PROBLEEM B

Schrijf een programma dat de suikerspiegel van een patiënt onderzoekt. Het berekent het verschil tussen de laagste glucoseconcentratie en de hoogste concentratie nadien, wanneer de patiënt een medicijn heeft gekregen.

ABSTRACTIE: VOORBEELD 2

PROBLEEM A

Schrijf een programma dat de historische gegevens van de waarden van cryptomunten analyseert. Gegeven een lijst van historische waardes, berekent het programma de maximale theoretische winst over die periode.

PROBLEEM B

Schrijf een programma dat de suikerspiegel van een patiënt onderzoekt. Het berekent het verschil tussen de laagste glucoseconcentratie en de hoogste concentratie nadien, wanneer de patiënt een medicijn heeft gekregen.

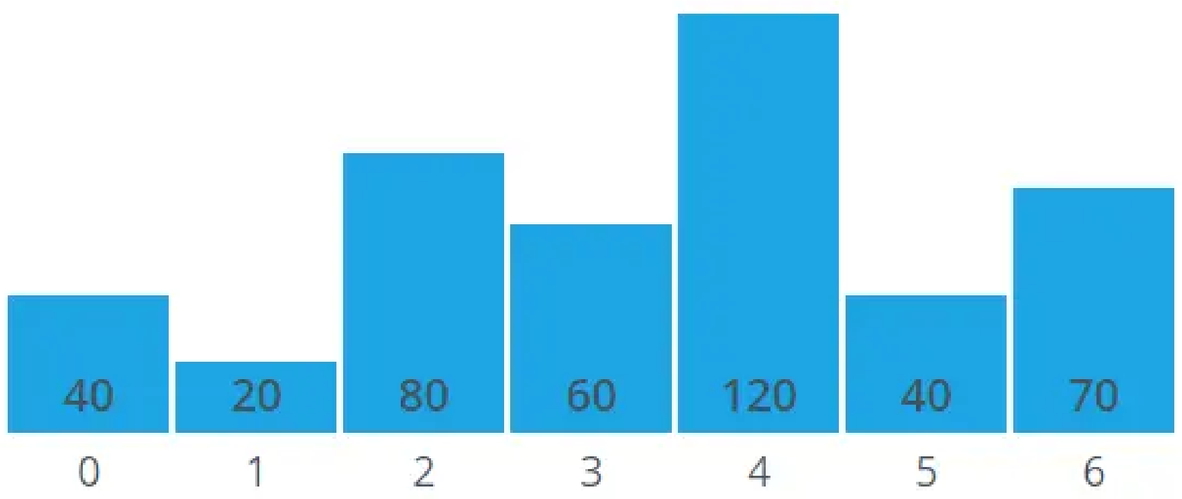
Deze twee problemen hebben dezelfde oplossing. Kan jij ze herleiden tot één algemene probleemstelling?

ABSTRACTIE: PROBLEMEN HERLEIDEN TOT ESSENTIE

Als we voorgaande problemen reduceren tot hun essentie, komen we tot het volgende:

Gegeven een lijst van nummers, geef het grootst mogelijke verschil terug tussen een lage waarde en een hoge waarde die ergens erna komt.

Wat is de oplossing van dit probleem voor de lijst hieronder?



Merk op dat er geen enkele domeinspecifieke kennis meer overblijft. De details zijn weggeabstraheert.

ABSTRACTIE: ANDERE INTERPRETATIE

Een tweede interpretatie van abstractie, is het **expres verbergen van de details van hoe iets werkt**.

Een paar voorbeelden:

- Hoe werkt een afwasmachine eigenlijk? Wat gebeurt er als je een bepaald programma selecteert?
- Hoe komt het dat een smartphone zowel met wifi als met mobiele gegevens op het internet kan? Als je beiden op hebt staan, welke van de twee wordt er dan gebruikt?
- Levende wezens zijn opgebouwd uit molecules, maar toch wordt dit 'detail' genegeerd in sommige disciplines binnen levenswetenschappen

In al deze situaties heeft men er bewust voor gekozen om details te verbergen.

ABSTRACTIE: PRAKTISCH

Neem de tijd om een probleem te begrijpen, maar zorg dat je het kan herleiden tot zijn essentie.

Gebruik functies om details van je code te verbergen en te vervangen door simpele, begrijpbare namen. Bijvoorbeeld:

Python

```
# De variabele 'belgie' een complexe datastructuur
# waar we de details niet van kennen
hoofdstad = geef_hoofdstad(belgie)
# Wat verwacht je dat de waarde van 'hoofdstad' is?
```

Python

ABSTRACTIE: PRAKTISCH

Neem de tijd om een probleem te begrijpen, maar zorg dat je het kan herleiden tot zijn essentie.

Gebruik functies om details van je code te verbergen en te vervangen door simpele, begrijpbare namen. Bijvoorbeeld:

Python

```
# De variabele 'belgie' een complexe datastructuur
# waar we de details niet van kennen
hoofdstad = geef_hoofdstad(belgie)
# Wat verwacht je dat de waarde van 'hoofdstad' is?
```

Python

```
# De details van de functie
def geef_hoofdstad(land):
  return land["steden"][0]["label"]
```

SORTEREN:

PROBLEEMSTELLING

SORTEREN: VOORAF

We hanteren volgend plan van aanpak:

1. We formuleren het probleem op een **abstracte** manier
2. We proberen het probleem met de hand op te lossen, om te begrijpen welke stappen er nodig zijn
3. We vertalen onze stappen naar een echt algoritme
4. We **evalueren** ons algoritme: is er ruimte voor verbetering? Zijn er beperkingen?
5. (Eventueel) we programmeren het algoritme in Python

Programmeren is pas de laatste stap. Het is strikt genomen zelfs niet nodig om het probleem en de oplossing te begrijpen!

SORTEREN: FORMULERING VAN HET PROBLEEM

Hoe formuleren we het probleem van sorteren op een abstracte manier? Kunnen we details weglaten?

SORTEREN: FORMULERING VAN HET PROBLEEM

Hoe formuleren we het probleem van sorteren op een abstracte manier? Kunnen we details weglaten?

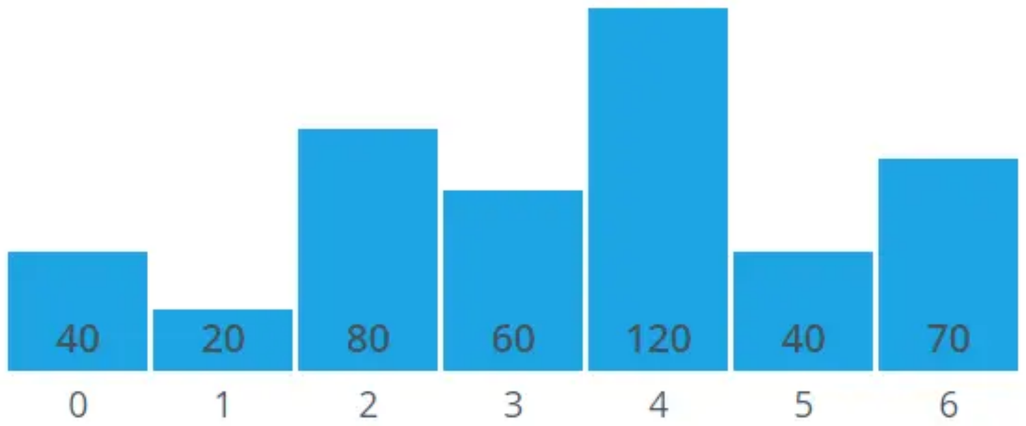
Gegeven een lijst van nummers, sorteer deze lijst van laag naar hoog.

Merk op: we laten bewust een aantal details achterwege (abstractie):

- We focussen enkel op simpele nummers. We kunnen ons probleem later uitbreiden naar letters, datastructuren, ... maar dit is niet deel van de essentie van het probleem
- We sorteren van laag naar hoog. Omgekeerd zou ook kunnen, maar dit is een detail dat niet relevant is om het probleem te begrijpen

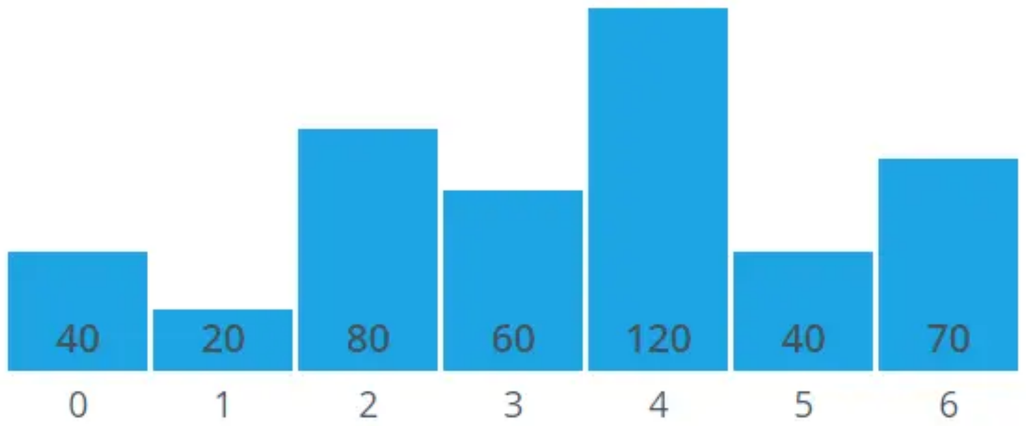
ZELF AAN DE SLAG

Gegeven onderstaande lijst. Sorteer deze met de hand (papier, digitaal kladblok, Excalidraw, ...).

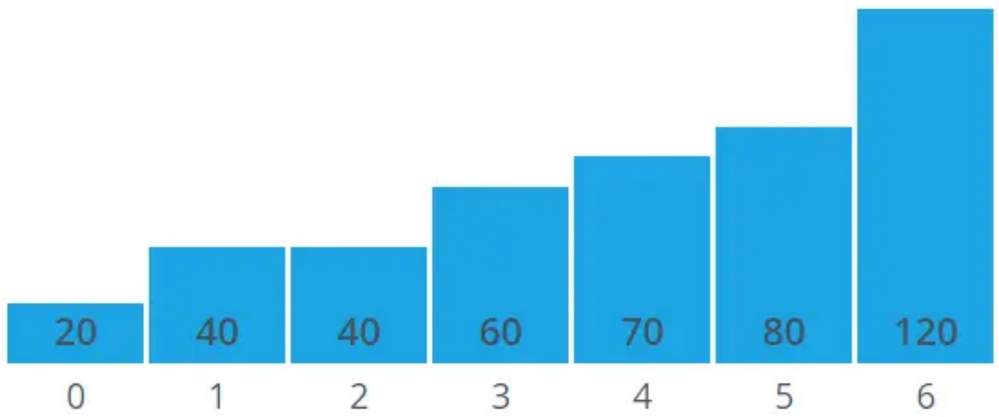


ZELF AAN DE SLAG

Gegeven onderstaande lijst. Sorteer deze met de hand (papier, digitaal kladblok, Excalidraw, ...).



Oplossing:



Wat maakt het zo makkelijk?

- We hebben een overzicht van alle waardes
- De waardes worden visueel voorgesteld (gekleurde balk)

Jammer genoeg is dit niet algemeen toepasbaar. Wat als we 10 000 cijfers moeten sorteren? De balken worden onleesbaar en het overzicht valt weg.

We moeten een strategie vinden die algemeen werkt.

SORTEREN: KAARTENSPEL

Sorteer een reeks kaarten van laag naar hoog. Meer specifiek:

1. De kaarten liggen ondersteboven
2. In één beurt mag je het volgende doen:
    1. Twee kaarten omdraaien, en
    2. De kaarten van plaats wisselen (indien je wilt)
3. Beurten zijn niet gratis: na elke beurt moet je een opdracht doen
4. Probeer in zo min mogelijk beurten de reeks gesorteerd te hebben
5. Roep "KLAAR" als je denkt dat je reeks gesorteerd is

SORTEREN: KAARTSPEL EN COMPUTERS

Het kaartspel is een abstractie van hoe sorteeralgoritmes op computers werken:

- De computer 'ziet' niet alle data (zeker als het om 1000'en elementen gaat)
- De computer kan individuele elementen selecteren
- De computer kan elementen van plaats wisselen
- Operaties op een computer vragen tijd en energie. We willen liefst zo snel mogelijk klaar zijn

BUBBLE SORT: ANIMATIE

De animatie in de link voert Bubbel sort stap voor stap uit. Probeer op basis hiervan te begrijpen hoe het algoritme werkt.

https://www.hackerearth.com/practice/algorithms/sorting/bubble-sort/visualize/

BUBBLE SORT: CONCEPTEN

In 1 iteratie gebeurt het volgende:

- We starten aan het begin van de lijst
- We controleren of elk element in de lijst groter is dan het volgende element.
  Indien ja, wissel de elementen van plaats
- Op het einde van de iteratie staat het grootste element gegarandeerd helemaal rechts (het 'borrelt' op)

Na 1 iteratie staat 1 element op de juiste plaats. Om **alle** elementen juist te zetten, hebben we evenveel iteraties nodig als er elementen in de lijst zijn.

BUBBLE SORT V1: ALGORITME IN PSEUDO-CODE

```Python
lijst = de te sorteren lijst
n = lengte van lijst

herhaal n keer:
  voor elk element e in lijst:
    als e groter is dan het volgende element e+1:
      dan wissel e en e+1 van plaats
```

Probeer op basis van deze pseudo-code Bubble sort te implementeren in Python.

BUBBLE SORT: OPTIMALISATIES

We hebben iets dat werkt, maar kan het beter? **Absoluut.**

Om te begrijpen hoe we dit algoritme kunnen verbeteren, nemen we eerst een stap terug naar computationeel denken, meer bepaald **evaluatie**.

COMPUTATIONEEL DENKEN

DEEL 5

EVALUATIE

DEELCOMPETENTIES CD

Iconen: CoDe-platform KU Leuven
Link naar de slides waarop deze leerstof gebaseerd is

Abstractie	Onnodige details weglaten
Veralgemening	Principes veralgemenen zodat ze breder toepasbaar zijn
Decompositie	Opdeling in kleinere, eenvoudigere problemen
Algoritmisch denken	Opstellen van een stappenplan voor het oplossen van een probleem
Evaluatie	Kritisch bekijken van oplossing

EVALUATIE

> "*Kritisch nakijken van de bekomen oplossingsmethode*"

Een probleem heeft vaak meerdere oplossingen. Hoe weet je of je de beste oplossing hebt? Is er zelfs een 'perfecte' oplossing?

In sommige gevallen zal je oplossingen tegen elkaar moeten afwegen. De verschillende sorteeralgoritmes hebben elk hun sterktes en zwaktes op het vlak van:

- Geheugengebruik: heb ik extra datastructuren nodig? Moet ik data dupliceren?
- Tijd: hoe snel is het algoritme klaar?

Hou er ook rekening mee dat er **best case** en **worst case** scenarios zijn. Sommige algoritmes zijn zeer efficiënt in de best case, maar verschrikkelijk slecht in de worst case.

EVALUATIE: PRAKTISCH

Stel jezelf de volgende vragen over je oplossing:

1. Hoe snel werkt het in de worst case?
2. Hoe snel werkt het in de best case?
3. (Zelfde vragen voor geheugen)
4. Doet het algoritme onnodig extra werk? Is er een manier om dit te omzeilen?

BUBBLE SORT: OPTIMALISATIES

EVALUATIE VAN BUBBLE SORT V1

Hoe snel werkt het in best case?

Best case = alles is al gesorteerd

Toch zal het algoritme volledig uitgevoerd worden. Dit is onnodig extra werk.

Oplossing: hou een variabele bij die aangeeft of er een wissel is opgetreden. Als je in 1 iteratie geen enkele wissel doet, dan is de lijst volledig gesorteerd.

Python

EVALUATIE VAN BUBBLE SORT V1

Hoe snel werkt het in best case?

Best case = alles is al gesorteerd

Toch zal het algoritme volledig uitgevoerd worden. Dit is onnodig extra werk.

Oplossing: hou een variabele bij die aangeeft of er een wissel is opgetreden. Als je in 1 iteratie geen enkele wissel doet, dan is de lijst volledig gesorteerd.

Python

```
lijst = de te sorteren lijst
n = lengte van lijst

herhaal n keer:
  is_gewisseld = False
  voor elk element e in lijst:
    als e groter is dan het volgende element e+1:
      dan wissel e en e+1 van plaats
      is_gewisseld = True
  als not is_gewisseld:
    stop
```

Ander onnodig extra werk?

Na `k` iteraties staan de laatste `k` elementen al op hun plaats. Toch controleren we in 1 iteratie alle `n` elementen.

Oplossing: sla de laatste `k` elementen over.

Python

Ander onnodig extra werk?

Na `k` iteraties staan de laatste `k` elementen al op hun plaats. Toch controleren we in 1 iteratie alle `n` elementen.

Oplossing: sla de laatste `k` elementen over.

Python

```
lijst = de te sorteren lijst
n = lengte van lijst

voor k van 0 tot n:
  is_gewisseld = False
  voor i van 0 tot (n - 1 - k):
    e = lijst[i]
    als e groter is dan het volgende element e+1:
      dan wissel e en e+1 van plaats
      is_gewisseld = True
  als not is_gewisseld:
    stop
```

BUBBLE SORT: V2

Gebruik voorgaande info om een tweede, meer optimale versie van Bubble sort te schrijven.

Deze versie kan je gebruiken als basis voor deel 3 van het project. Denk wel na: dit algoritme is nog niet exact wat je nodig hebt. Wat ga je nog moeten aanpassen?