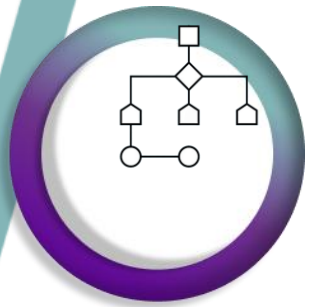
ALGORITHMUS

DEAF YOUNG CODE

Projektnummer: 2023-2-IT03-KA220-YOU-000179130

Co-funded by
the European Union

Fortgeschrittener Lehrplan



- Definiere einen Algorithmus technisch und erklären ihn einfach.
- Verstehe die Effizienz von Algorithmen

  (Bedeutung von "Zeitkomplexität" und "Raumkomplexität„)
- Identifiziere gängige Algorithmen-Typen:

  Sortieren, Suchen, rekursiv, iterativ, gierig usw.
- Analysiere einen Algorithmus Schritt für Schritt,

  um Korrektheit und Effizienz zu überprüfen.
- Schreibe einfache Algorithmen in Pseudocode oder Flussdiagrammen.

Was ist ein Algorithmus?

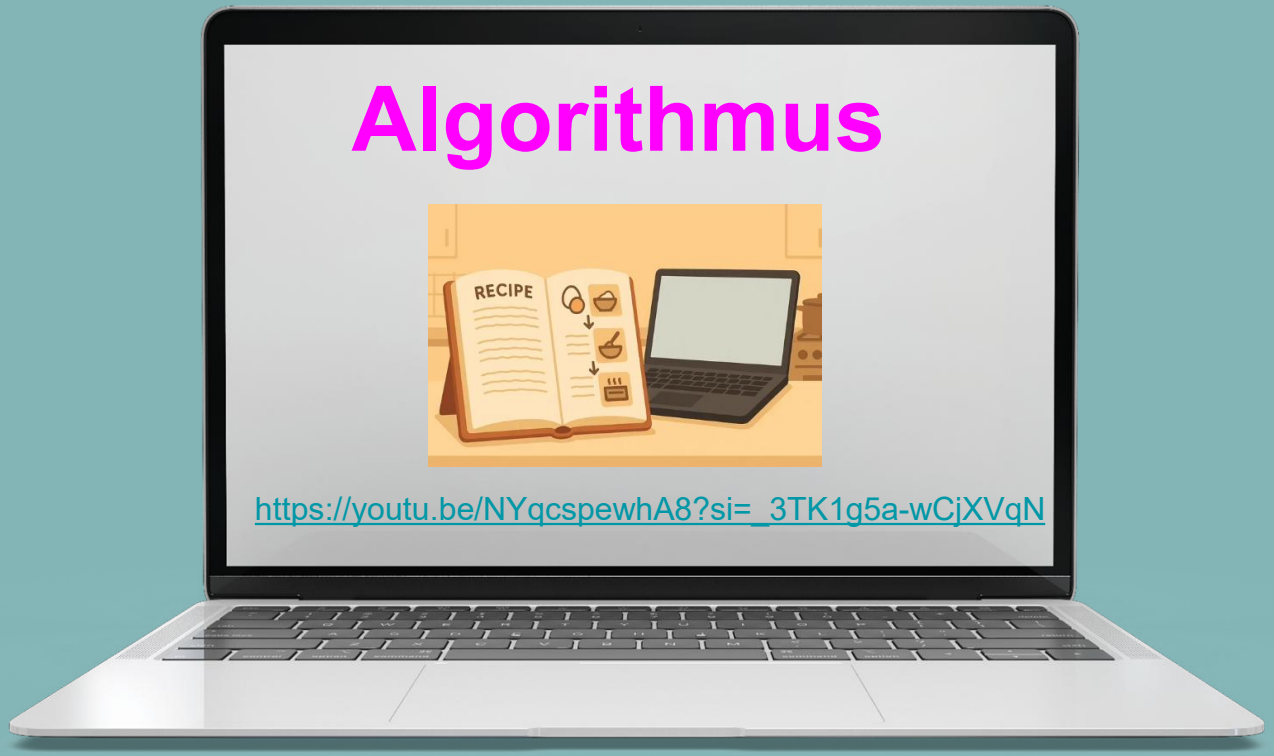
DEAF YOUNG CODE

Technische Definition

Ein Algorithmus ist eine

- endliche Folge
- genau definierter Anweisungen
- zur Lösung eines Problems.

Video


Algorithmus
RECIPE
https://youtu.be/NYqcspewhA8?si=_3TK1g5a-wCjXVqN

Schlüsselmerkmale

DEAF YOUNG CODE

1. **Eingabe**: Die Daten, mit denen du startest.
2. **Ausgabe:** Das Ergebnis oder die Lösung.
3. **Endlichkeit:** Muss nach einer begrenzten Anzahl von Schritten abgeschlossen werden.
4. **Definitivität:** Jeder Schritt ist klar definiert.
5. **Wirksamkeit:** Jeder Schritt kann in angemessener Zeit durchgeführt werden.

Algorithmen-Grundlagen

DEAF YOUNG CODE

Wie Algorithmen funktionieren

- Definiere das Problem klar → Was wollen wir lösen?
- Entwerfe ein Schritt-für-Schritt-Verfahren → Was sind die Anweisungen, um eine Lösung zu finden?
- Teste mit Beispielen → Überprüfe, ob der Algorithmus mit Beispieleingaben funktioniert.
- Effizienz analysieren → Wie schnell ist es? Wie viel Speicher verbraucht es?
- Verfeinern und optimieren → Kann es schneller sein oder weniger Speicher verbrauchen?

Algorithmusdiagramm (Flussdiagramm)

DEAF YOUNG CODE

Flussdiagramm = visuelle Darstellung, um einen Algorithmus Schritt für Schritt darzustellen. Es hilft, die Logik zu sehen und nicht nur zu lesen.

Element	Symbol	Funktion	Beispiel
Start / Ende	Oval 	zeigt an, wo der Algorithmus beginnt oder endet	„Start“ / „Ende“
Prozess / Schritt	Rechteck 	Handlung oder Anweisung	„Zwei Zahlen addieren“
Entscheidung / Bedingung	Diamant 	Frage, die den Ablauf verändert (Ja/Nein)	„Ist a > b?“
Input / Output	Parallelogram 	Daten, die eingehen oder ausgehen	„Eingabezahlen“ / „Ausgabeergebnis“
Pfeil	→ 	zeigt die Richtung des Flusses an	
Schleife / Verbindung	gebogener Pfeil Oder Beschriftung 	ein Schritt, der wiederholt wird, bis eine Bedingung erfüllt ist	“Wiederholen, bis sortiert“

Algorithmusdiagramm (Flussdiagramm)

DEAF YOUNG CODE

Ein Flussdiagramm lesen:

- Beginne → finde das "Start"-Oval.
- Eingabe → Schau dir an, welche Informationen ins System gelangen (zum Beispiel: Zahlen, Text).
- Prozesse → Folge jedem Rechteck. Das sind Aktionen oder Schritte.
- Entscheidung → Wenn du einen Diamanten erreichst, stellt er eine Ja/Nein-Frage.
- Wenn "Ja", folge einem Pfeil.
- Wenn "Nein", dann folge dem anderen.
- Loop → Wenn ein Pfeil zu einem früheren Schritt zurückgeht, bedeutet das "wiederholen".
- Ausgabe → Am Ende sehen Sie das Ergebnis oder die Antwort.
- Ende → Der Prozess stoppt.

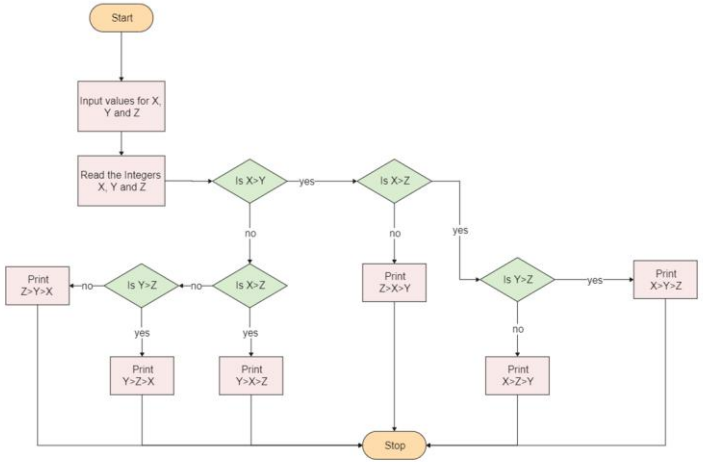
Beispiele: Nummern sortieren

Problem: Sortiere die Zahlen [4, 2, 7] vom kleinsten zum größten Wert.

1. Flussdiagramm

(Schritt für Schritt visuell)

```
Start
  |
  v
Eingabe Nummern: 4, 2, 7
  |
  v
Vergleiche ersten beiden Nummern:
ist 4 > 2? ja → tausche
  |
  v
Nummern jetzt: 2, 4, 7
  |
  v
Vergleiche : 4 and 7
ist 4 > 7? No → Ordnung halten  |
  v
Output sorted numbers: 2, 4, 7
  |
  v
End
```



Tipp für gehörlose Lernende:

Verwende für jeden Schritt Pfeile und Kästchen.

Markiere die Wechselaktion mit einer anderen Farbe.

Eingabe → Prozess zeigen → klar ausgeben.

Co-funded by the European Union

Beispiele: Nummern sortieren

Problem: Sortiere die Zahlen [4, 2, 7] vom kleinsten zum größten Wert.

2. Pseudocode (Simple Text Version)

```
Algorithm SortThreeNumbers:
  Input: a, b, c
  If a > b then
      Swap a and b
  If b > c then
      Swap b and c
  If a > b then
      Swap a and b
  Output: a, b, c
End Algorithm
```

Algorithmus Sortiere drei Zahlen:
Eingabe: a, b, c
wenn a > b, dann
  a und b vertauschen
wenn b > c, dann
  b und c vertauschen
wenn a > b, dann
  a und b vertauschen
Ausgabe: a, b, c
Ende Algorithmus

Notizen:

Pseudocode sind geschriebene Anweisungen, keine Programmiersprache.

Schritt-für-Schritt-Klarheit ist entscheidend.

Beispiele: Nummern sortieren

Problem: Sortiere die Zahlen [4, 2, 7] vom kleinsten zum größten Wert.

3. Diagramm (visueller Überblick)

```
[Input: 4,2,7]
      |
      v
[Compare 4 and 2]
      |
      v
[Swap if needed] ---> [Numbers now: 2,4,7]
      |
      v
[Compare 4 and 7]
      |
      v
[Swap if needed] ---> [Numbers now: 2,4,7]
      |
      v
[Output: 2,4,7]
```

[Eingabe: 4,2,7]
|
v
[Vergleiche 4 und 2]
|
v
[Bei Bedarf tauschen] --> [Zahlen jetzt: 2,4,7]
|
v
[4 und 7 vergleichen]
|
v
[Bei Bedarf tauschen] --> [Zahlen jetzt: 2,4,7]
|
v
[Ausgabe: 2,4,7]

Wichtige Punkte

Flussdiagramm = Schritt-für-Schritt-Karte des Algorithmus.

Enthält Entscheidungen, Schleifen, Eingänge, Ausgaben.

Super, um später Pseudocode oder Programme zu schreiben.

Diagramm = großes Bild, einfacher Überblick.

Zeigt Hauptaktionen und Ablauf ohne alle Schritte.

Das hilft, die Logik schnell zu verstehen, besonders für visuelle Lerner*innen.

Wichtige Punkte

Zeitkomplexität (Big O-Notation):

Misst, wie schnell ein Algorithmus läuft, wenn die Eingabegröße wächst.

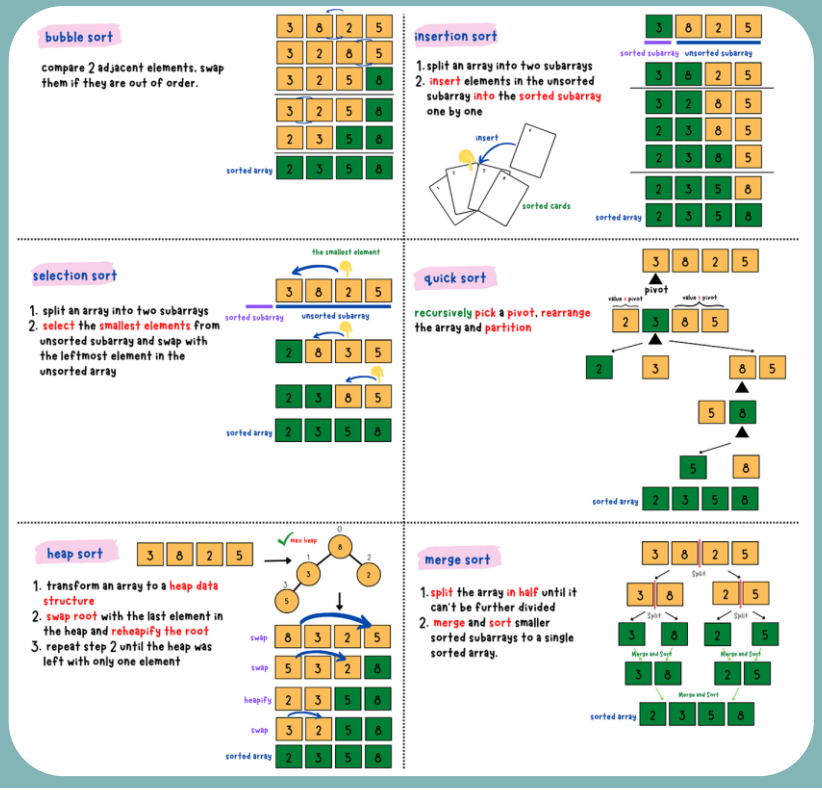
Raumkomplexität: Misst, wie viel Speicher ein Algorithmus verbraucht.

Häufige Algorithmustypen:

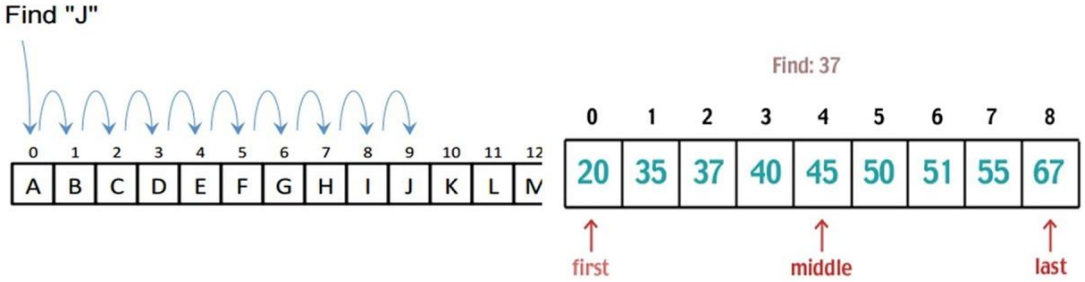
- **Sortieren**: Blasensortieren, Zusammenführen, Sortieren, Schnellsortieren.
- **Suche**: Lineare Suche, Binärsuche.
- **Rekursion**: Der Algorithmus ruft sich selbst auf (z. B. Fakultätsfaktor).
- **„Gierige“ Algorithmen**: Triff in jedem Schritt die beste Wahl (z. B. kürzester Weg).
- **Dynamische Programmierung**: Probleme in kleinere Teilaufgaben aufteilen.
- **Datenstrukturen**: Algorithmen benötigen oft Listen, Arrays, Bäume oder Graphen.

Vergleichsbasierte Sortierung

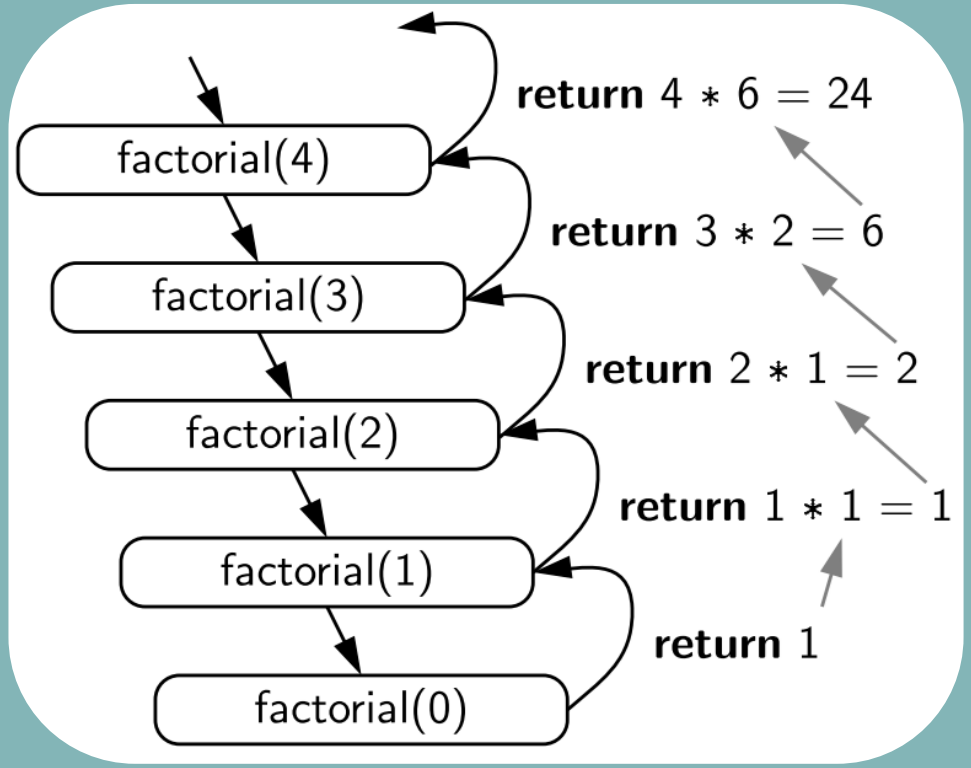
DEAF YOUNG CODE



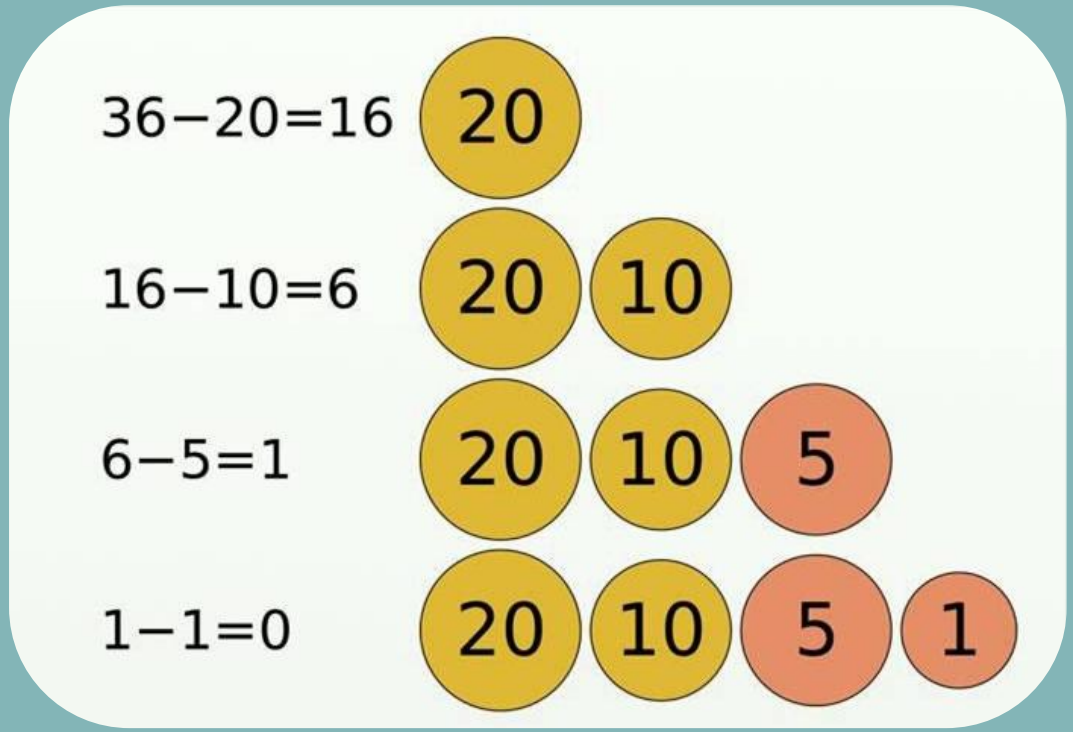
Vergleichsbasierte Sortierung



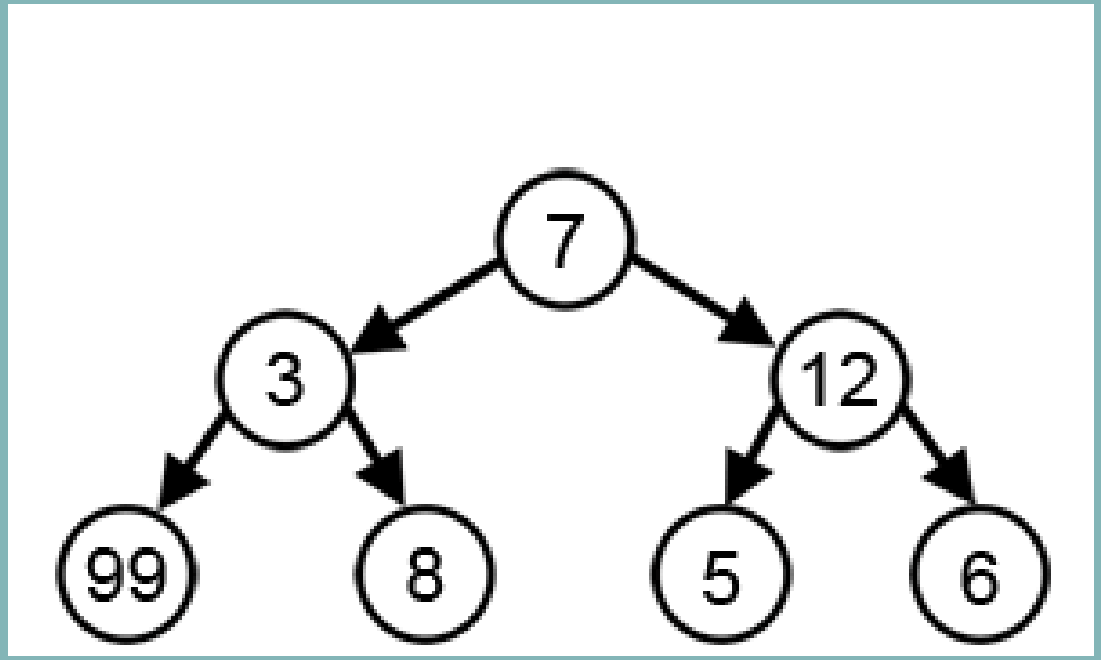
Vergleichsbasierte Sortierung



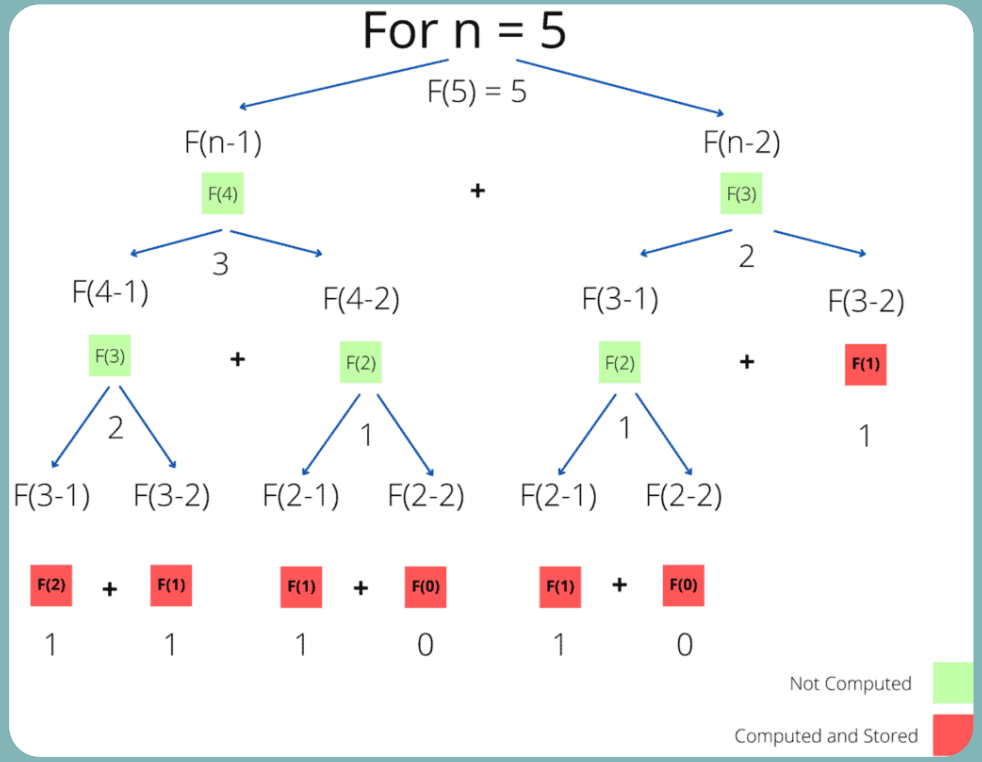
Vergleichsbasierte Sortierung



Vergleichsbasierte Sortierung



Vergleichsbasierte Sortierung



Interaktive Übungsidee (praktisches Lernen)

DEAF YOUNG CODE

Ziel:

- Hilf den Lernenden, ein Algorithmus-Design auf unterhaltsame und visuelle Weise zu üben.
- Stärke das Verständnis von Schritten, Entscheidungen, Schleifen und Effizienz.

Interaktive Übungsidee (praktisches Lernen)

DEAF YOUNG CODE

Schritt 1:
Wähle eine einfache Aufgabe.

Schritt 2:
Erstelle ein Flußdiagramm

Interaktive Übungsidee (praktisches Lernen)

DEAF YOUNG CODE

Schritt 1:

Beispiele für alltägliche Aufgaben:

- Tee machen
- Bücher nach Größe sortieren
- Morgenroutine (aufwachen → Zähne putzen → frühstücken → mich anziehen)
- Einen Rucksack packen
- Früchte nach Größe bestellen

Interaktive Übungsidee (praktisches Lernen)

Schritt 2: Zeichne ein Flussdiagramm

Anweisungen für die Lernenden:
Start → Zeichne das Startsymbol (oval).
Eingang / Ausgabe → Zeigt die Daten oder das Ergebnis (Parallelogramm).
Aktionen / Schritte → Verwende Rechtecke für jede Aktion.
Entscheidungspunkte → Diamanten für Ja/Nein-Fragen verwenden.
Loops → Zeichnen Pfeile, die bei Bedarf zu wiederholten Schritten zurückführen.
Ende → Zeige das Endergebnis oder den Abschluss (Oval).

Tipp: Verwende Farben oder Klebezettel für Aktionen, Entscheidungen und Ausgaben.

Interaktive Übungsidee (praktisches Lernen)

DEAF YOUNG CODE

Schritt 3:

Teilen und vergleichen

Jede Teilnehmer*in oder Gruppe zeigt ihr Diagramm.

Diskutieren:

Wurden alle wichtigen Schritte berücksichtigt?

Gibt es unnötige Schritte, die entfernt werden können?

Kann die Aufgabe schneller oder effizienter erledigt werden??

Interaktive Übungsidee (praktisches Lernen)

DEAF YOUNG CODE

Schritt 4: Effizienz (optional)

Führe die Idee der Algorithmuseffizienz ein:
Weniger Schritte = schnellerer Algorithmus
Wiederholende unnötige Schritte = langsamerer Algorithmus

Interaktive Übungsidee (praktisches Lernen)

DEAF YOUNG CODE

Schritt 4: Effizienz (optional)

Beispiel: Sortieren von 3 Büchern

Schüler A: Prüft alle Paare zweimal → 6 Schritte

Schüler B: Smart swaps nur → 3 Schritte.

Fragen: Welcher Algorithmus ist besser? Warum?

Zeigen Sie, dass selbst einfache alltägliche Aufgaben "effizientere" Algorithmen haben können.

Interactive Exercise Idea (Hands-On Learning)

DEAF YOUNG CODE

Schritt 5: Kinästhetisches Lernen (praktischer Bonus)

Verwende echte Gegenstände: Bücher, Becher, Klebezettel.

Die Lernenden spielen die Schritte durch Körperbewegung nach und folgen ihrem eigenen Flussdiagramm.

Hilft visuellen und taktilen Lernern, Abfolge, Schleifen und Entscheidungen zu verstehen.

Aktivität

Quiz

Frage 1

Was ist ein Algorithmus?

A. Computermarke
B. Detaillierte Anleitung zur Lösung eines Problems
C. Datentyp
D. Passwort

Frage 2

Welches dieser Beispiele ist ein gewöhnliches Beispiel für einen Algorithmus?

A. Kochrezept
B. Fußball
C. Musikvideo
D. Bild

Frage 3

DEAF YOUNG CODE

Wie nennen wir die Informationen, die wir in einen Algorithmus eingeben?

A. Ausgang
B. Eingang
C. Fehler
D. Übersichtliches Diagramm

Frage 4

DEAF YOUNG CODE

**Was passiert,
wenn ein Algorithmus einen Fehler (Bug) hat?**

A. Es läuft schneller
B. Es stoppt oder liefert falsche Ergebnisse
C. Es wird sicherer
D. Es verschwindet

Frage 5

Wie ist die richtige Reihenfolge des Algorithmus?

A. Ausgang → Prozess → Eingang
B. Prozess → Eingang → Ausgang
C. Eingang → Prozess → Ausgang
D. Eingang → Ausgang → Prozess

Vielen Dank für deine Aufmerksamkeit!

DEAF YOUNG CODE

… bleib dran

für die nächste Präsentation!

Von der Europäischen Union finanziert. Die geäußerten Ansichten und Meinungen entsprechen jedoch ausschließlich denen des Autors bzw. der Autoren und spiegeln nicht zwingend die der Europäischen Union oder der Europäischen Exekutivagentur für Bildung und Kultur (EACEA) wider.
Weder die Europäische Union noch die EACEA können dafür verantwortlich gemacht werden.

Projektnummer: 2023-2-IT03-KA220-YOU-000179130

Co-funded by the European Union