Einführung in Algorithmen **Prüfungsstoff**

1.1 Du kannst aufgrund des Zitats aus dem Buch von R. Sedgewick und K. Wayne beschreiben, was ein Algorithmus ist.

1.2 Du kannst aufgrund der obigen Algorithmusdefinition beurteilen, ob es sich bei einer Problemlösugsbeschreibung um einen Algorithmus handelt.

2.1 Du kennst die korrekte englische Übersetzung und die entsprechenden Abkürzung für den grössten gemeinsamen Teiler.

2.2 Du kannst mit der klassischen Version des euklidischen Algorithmus den ggT zweier Zahlen schrittweise gemäss dem im Unterricht behandelten Verfahren berechnen.

2.3 Du kannst an einem Beispiel erklären, warum beim klassischen euklidischen Algorithmus keines der beiden Argumente negativ sein darf.

2.4 Du kannst den klassischen euklidischen Algorithmus in Python implementieren.

2.5 Du kannst den zentralen Nachteil des klassischen euklidischen Algorithmus beschreiben, der seine Effizienz reduziert und ein Beispiel angeben, wo dieser Nachteil gut sichtbar wird.

2.6 Du kannst mit der modernen Version des euklidschen Algorithmus den ggT von zwei Zahlen schrittweise berechnen.

2.7 Du kannst den modernen euklidischen Algorithmus in Python implementieren.

2.8 Du kannst den Worst Case beschreiben, bei dem sowohl der klassische als auch der moderne Algorithmus ein Maximum an Verarbeitungsschritten durchführen muss.

2.9 Du kannst die ersten 10 Werte der Fibonacci-Folge notieren.

3.1 Du weisst, dass die Laufzeitkomplexität ein Hilfsmittel ist, das die Komplexität eines Algorithmus' unabhängig von der Hardware und der Programmiersprache als Funktion der Anzahl n der Eingabedaten darstellt.

3.2 Du verstehst die Big-Oh-Notation und kannst solche Ausdrücke vereinfachen.

3.3 Du kennst von folgenden Algorithmen die Laufzeitkomplexität:

- Einen Wert in eine Liste schreiben, bzw. aus einer Liste lesen: $O(1)$
- Einen Wert in einer sortierten Liste suchen: $O(\log n)$
- Einen Wert in einer unsortierten Liste suchen: $O(n)$
- „Schnelle“ Sortieralgorithmen (Mergesort): $O(n \cdot \log n)$
- „Naive“ Sortieralgorithmen (Gnomesort, Insertionsort, Selectionsort, ...): $O(n^2)$
- Travelling Salesman Problem: $O(n!)$

3.4 Du kannst den Zeitbedarf von Algorithmen mit bekanntem $O\big(f(n)\big)$ abschätzen, wenn man die Eingabegrösse n verändert.

3.5 Du kannst angeben, welche Laufzeitkomplexitäten *nicht handhabbar* sind; d. h. für für grosse n nicht mehr praktisch berechenbar sind.

3.6 Du kannst die Laufzeitkomplexität einzelner Teile von Python-Codefragmenten abschätzen und so ihre Laufzeitkomplexität bestimmen.