Aufgabenblatt 6

Allgemeine Informationen zum Aufgabenblatt:

- Die Abgabe erfolgt in TUWEL. Bitte laden Sie Ihre Python-Datei bis spätestens **Freitag, 29.05.2020 20:00 Uhr** in TUWEL hoch.
- Beachten Sie bitte folgende Punkte
  - Ihre Programme müssen ausführbar sein, d.h. einzelne Programme sollten z.B. keine Syntaxfehler oder Typfehler enthalten.
  - Bei kleinen logischen Fehlern (falsche Ergebnisse) werden wir keine Punkte abziehen. Daher sollten Sie immer versuchen, alle Aufgaben vollständig abzugeben.
  - Die Angabedatei muss nicht umbenannt werden.
  - Wir haben diesmal Hinweise zu den Typen (type hints) in den vorgegebenen Code inkludiert (siehe `https://www.python.org/dev/peps/pep-0484/`). Damit werden die Typen von Parametern und der Rückgabe dokumentiert. Ein Beispiel:

    ```
    def gcd(a: int, b: int) -> int:
    ```

    Damit wird beschrieben, dass die Funktion `gcd` zwei Parameter vom Typ `int` besitzt und die Funktion wiederum einen `int`-Wert zurückliefert.

In diesem Aufgabenblatt werden folgende Themen behandelt:

- Klassen
- O-Notation
- Algorithmen

Aufgabe 1 (1 Punkt)

In dieser Aufgabe müssen Sie eine Klasse für Brüche implementieren. Ein Bruch hat zwei Instanzvariablen numerator (Zähler) und denominator (Nenner) nach dem Schema $\frac{numerator}{denominator}$. Das Erzeugen eines Bruches ist schon in der Angabedatei vorgegeben. Dabei gehen wir davon aus, dass beim Nenner nicht 0 übergeben wird. Sie dürfen an dieser Stelle auch eine zusätzliche Behandlung dafür einbauen, aber der kleine Test in der Funktion `test` erfordert das nicht.
Beschreiben Sie kurz in einem Kommentar, was `__init__` macht, welche weitere Funktion aufgerufen wird und was am Ende abgespeichert wird. Achten Sie dabei darauf, was zur Klasse gehört und was sich außerhalb der Klasse befindet.
Sie müssen nun noch folgende Operationen implementieren:

- Die Ausgabe (Zähler/Nenner) des Inhalts eines Bruchobjekts.
- Addition von zwei Brüchen. Die Berechnung folgt den folgende Regeln (siehe z. B. `https://de.wikipedia.org/wiki/Bruchrechnung#Addieren_und_Subtrahieren`) und am Ende gibt diese Methode ein neues Bruchobjekt zurück.
- Invertieren des Bruches (Methode `inverse`), d.h. der Zähler wird zum Nenner und der Nenner zum Zähler. Es wird wiederum ein neues Bruchobjekt zurückgegeben.

In der Methode `test` finden Sie Anweisungen, von denen einige auskommentiert sind. Wenn alles richtig implementiert wird, dann können Sie alle diese Zeilen ausführen und der Test sollte folgende Ausgabe erzeugen:

```
1/4
3/7
19/28
7/3
1/2
```

Bei der letzten Ausgabe gibt es noch eine Zusatzfrage im Kommentar, die Sie beantworten müssen.

Aufgabe 2 (1 Punkt)

Sie haben fünf Funktionen mit asymptotischen oberen Schranken gegeben:

a) $\log n + 5$ ist in $O(n)$

b) $n^2 \times n + 3$ ist in $O(n^2)$

c) $n^3 + n \times n + 4$ ist in $O(n^4)$

d) $n^2 + 2^{10}n$ ist in $O(n^2)$

e) 3^n ist in $O(2^n)$

Geben Sie bei jeder Funktion in der Abgabedatei an, ob die obere Schranke korrekt ist. Falls dies nicht der Fall ist, dann geben Sie eine kurze Erklärung dafür. Ein Beispiel für eine lineare Funktion ist schon vorgegeben.

Aufgabe 3 (1 Punkt)

Sie haben einige Funktionen gegeben, die jeweils Berechnungen durchführen. Für jede einzelne Funktion müssen Sie zwei Punkte beantworten:

- Beschreiben Sie kurz (maximal 2-3 Sätze) was die Funktion macht.
- Geben sie für jede Funktion die kleinste asymptotische obere Schranke für die Laufzeit an und erklären Sie ganz kurz, wie Sie zu Ihrem Ergebnis kommen.

Es ist wieder ein einfaches Beispiel vorgegeben.

Aufgabe 4 (1 Punkte)

Sie haben eine alternative Implementierung für die Ermittlung der maximalen Abschnittssumme gegeben. Beantworten Sie folgende Fragen:

- Wie geht diese Funktion vor? Wie wird die maximale Abschnittssumme ermittelt?
- Geben Sie für die Laufzeit die kleinste asymptotische obere Schranke an.
- Wie unterscheidet sich diese Implementierung bezüglich Speicheranforderungen von den in der Vorlesung gezeigten Implementierungen?
- Wie verhält sich der Algorithmus, wenn wir eine Liste mit einem Element übergeben?
- Wie verhält sich der Algorithmus, wenn wir eine leere Liste übergeben?

Aufgabe 5 (1 Punkte)

In dieser Aufgabe sollen Sie folgende einfache Algorithmen, die in der Vorlesung besprochen wurden, verbessern:

- `unique` (Folie 55): Diese Funktion überprüft, ob jedes Element nur einmal in einer Liste vorkommt und gibt `True` zurück, wenn dies stimmt, oder `False`, wenn ein Element doppelt vorkommt. Die Laufzeit dieser Funktion ist in $O(n^2)$. Implementieren Sie eine schnellere Variante `my_unique`, deren Laufzeit in $O(n \log n)$ ist. Sie dürfen nur mit der Liste (bzw. der Länge der Liste) und Methoden aus der Klasse `list` arbeiten. Erklären Sie in dem zusätzlichen Kommentar, warum diese Implementierung die geforderte obere Schranke für die Laufzeit besitzt.

- `disjoint` (Folie 56): Diese Funktion überprüft, ob die Schnittmenge der Elemente in drei Sequenzen leer ist. Trifft dies zu, dann wird `True` zurückgegeben, ansonsten `False`. Die Laufzeit dieser Funktion ist in $O(n^3)$. Implementieren Sie eine schnellere Variante `my_disjoint`, deren Laufzeit in $O(n^2)$ ist. Sie können dabei davon ausgehen, dass keine der drei Sequenzen Duplikate enthält. Erklären Sie auch in dem zusätzlichen Kommentar, warum diese Implementierung die geforderte obere Schranke für die Laufzeit besitzt.