Aufgabenblatt 7

Allgemeine Informationen zum Aufgabenblatt:

- Die Abgabe erfolgt in TUWEL. Bitte laden Sie Ihre Python-Datei bis spätestens **Freitag, 12.06.2020 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.

In diesem Aufgabenblatt werden folgende Themen behandelt:

- O-Notation
- Suchalgorithmen
- Sortieralgorithmen
- Klassen

Aufgabe 1 (1 Punkt)

Verändern Sie den in der Vorlesung präsentierten Algorithmus für die binäre Suche (Folie 15, Suchalgorithmen) folgendermaßen: Es wird der **kleinste** Index >= 0 zurückgegeben, an dem der gesuchte Wert `elem` in der Liste vorkommt. Ansonsten wird -1 zurückgeliefert. Implementieren Sie Ihre Lösung in die Funktion `binary_search`. Die Laufzeit muss in $O(\log n)$ liegen! Bei Aufruf der Testfunktion `test_search` muss folgende Ausgabe erfolgen:

```
1
12
0
-1
18
-1
```

Aufgabe 2 (1 Punkt)

Sie haben eine $n \times n$-Matrix gegeben, bei der jede Zeile und jede Spalte aufsteigend sortiert ist. Ein Beispiel:

```
1 3 3 4 5
1 3 4 4 6
2 3 4 5 6
3 5 5 6 8
4 5 8 9 9
```

Implementieren sie eine Funktion `matrix_search(matrix: list, value: int) -> bool` die `True` zurückliefert, wenn der Wert in der Matrix `matrix` (Liste von Listen) vorhanden ist. Ansonsten wird `False` zurückgeliefert. Der Aufruf der Testfunktion `test_matrix_search` muss dann folgende Ausgabe erzeugen:

```
[False, True, True, True, True, True, True, False, True, True, False]
```

Die Laufzeit zum Finden des Elements muss in $O(n)$ liegen, d.h. ein naives zeilenweises Durchsuchen ist hier nicht erlaubt (da in $O(n^2)$). Geben Sie im zusätzlichen Kommentar kurz an, warum die Laufzeit Ihrer Implementierung linear ist.

Aufgabe 3 (1 Punkt)

Implementieren Sie in der Funktion `sort_two_values(data: list) -> None` einen Sortieralgorithmus für eine Liste, bei der wir davon ausgehen können, dass nur genau zwei verschiedene Werte darin vorkommen. Z. B. muss die Liste

```
[1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0]
```

nach der Sortierung folgendermaßen aussehen:

```
[0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1]
```

Der Algorithmus muss auch auf Listen mit anderen Datentypen funktionieren, z. B.

```
["Bob", "Alice", "Alice", "Bob", "Alice"]
```

wird zu

```
['Alice', 'Alice', 'Alice', 'Bob', 'Bob']
```

Die Laufzeit muss in $O(n)$ liegen. Geben Sie im zusätzlichen Kommentar kurz an, warum die Laufzeit Ihrer Implementierung linear ist. Sie dürfen hier keine zusätzlichen Datenstrukturen verwenden. Der Algorithmus muss nur durch entsprechende Vertauschungen in der Liste realisiert werden.

Aufgabe 4 (1 Punkt)

In dieser Aufgabe sollten Sie auf die Listenmethode `sort` bzw. die Funktion `sorted` zurückgreifen. Sie müssen dabei folgende Aufgaben lösen:

- Legen Sie eine Liste mit 10 zufälligen Werten im Intervall [0, 10] an und geben Sie diese aus. Sortieren Sie dann diese Liste aufsteigend und geben Sie die Liste im neuen Zustand aus. Beispiel für eine Ausgabe:

  ```
  [0, 8, 9, 1, 4, 8, 4, 6, 7, 2]
  [0, 1, 2, 4, 4, 6, 7, 8, 8, 9]
  ```

- Wie im vorhergehenden Punkt, aber jetzt soll absteigend sortiert werden. Beispiel für eine Ausgabe:

  ```
  [5, 5, 1, 10, 2, 6, 6, 5, 9, 2]
  [10, 9, 6, 6, 5, 5, 5, 2, 2, 1]
  ```

- Erstellen Sie eine aufsteigend sortierte Kopie der vorgegebenen Liste `cities` von Städten und geben Sie diese neue Liste aus. Ausgabe:

  ```
  ['Amsterdam', 'Bratislava', 'Dublin', 'Helsinki', 'Rome', 'Vienna']
  ```

- Erstellen Sie eine nach Länge des Stadtnamens aufsteigend sortierte Kopie der vorgegebenen Liste `cities` von Städten und geben Sie diese neue Liste aus. Ausgabe:

  ```
  ['Rome', 'Vienna', 'Dublin', 'Helsinki', 'Amsterdam', 'Bratislava']
  ```

- Erstellen Sie eine nach Länge des Stadtnamens absteigend sortierte Kopie der vorgegebenen Liste `cities` von Städten und geben Sie diese neue Liste aus. Ausgabe:

  ```
  ['Bratislava', 'Amsterdam', 'Helsinki', 'Vienna', 'Dublin', 'Rome']
  ```

- Sie haben ein Dictionary `cities_2` gegeben. Jeder Eintrag hat als Schlüssel den Namen der Stadt und als Wert die Einwohnerzahl. Darauf aufbauend wird ein neues sortiertes Dictionary `sorted_cities` erzeugt.
  - Beschreiben Sie, welche Schritte hier durchgeführt werden, damit dieses Dictionary erzeugt wird. Nach welchem Kriterium werden die einzelnen Einträge sortiert?
  - Erzeugen Sie dann ein weiteres Dictionary, dessen Einträge nach der Einwohnerzahl sortiert sind und geben Sie dieses aus.

Aufgabe 5 (2 Punkte)

Implementieren Sie eine Klasse `Student`, die für das Verwalten von Daten über Studierenden verwendet wird. Es werden folgende Daten gespeichert:

- `name` ist der Name als String. Vereinfacht gehen wir davon aus, dass der Vorname gespeichert wird.
- `first_test` sind die Punkte, die beim ersten Test erreicht wurden (ganze Zahl).
- `second_test` sind die Punkte, die beim zweiten Test erreicht wurden (ganze Zahl).

Folgende Methoden sollten implementiert werden:

- Mit `__init__` werden die drei Datenattribute mit Werten belegt.
- Mit `__lt__` werden zwei Student-Objekte verglichen. Dabei wird zuerst nach den Punkten im ersten Test und danach nach den Punkten im zweiten Test verglichen.
- Mit `__str__` wird der Inhalt eines Objekts in der Form `Name: first_test/second_test` ausgegeben.

In der Funktion `test` ist eine Liste von Namen vorgegeben. Ihre Aufgaben sind nun:

- Erzeugen Sie zu jedem Namen ein Student-Objekt und geben Sie es in eine Liste `students`. Bei jedem Objekt werden die Punkte zufällig bestimmt:
  - Für den ersten Test liegen die Punkte im Intervall [90, 100].
  - Für den zweiten Test liegen die Punkte im Intervall [50, 100].
- Geben Sie die Liste zeilenweise folgendermaßen formatiert aus:

```
Student results:
Alice: 99/53
Bob: 91/57
Claire: 90/98
Donald: 90/71
Eric: 91/86
Jessica: 92/86
Leo: 97/72
Robyn: 97/64
William: 95/96
Zoe: 94/66
```

- Erzeugen Sie eine aufsteigend sortierte Kopie der Liste durch einen Aufruf mit `sorted` (benutzt `__lt__`).
- Geben Sie auch die sortierte Kopie zeilenweise folgendermaßen formatiert aus.

```
Sorted results:
Donald: 90/71
Claire: 90/98
Bob: 91/57
Eric: 91/86
Jessica: 92/86
Zoe: 94/66
William: 95/96
Robyn: 97/64
Leo: 97/72
Alice: 99/53
```