Blatt 8

Vorkurs Bauingenieurwesen - Informatik - 28.06.24

Aufgabe 0

Erstelle eine neue Textdatei, die auf `.py` endet, etwa mit dem Namen `blatt08.py`. In diese Datei sollen die Lösungen der folgenden Aufgaben geschrieben werden.

Aufgabe 1

Kopiere die Funktion `tausche` in die Datei aus Aufgabe 0.

```
def tausche(liste: list, i1: int, i2: int):
    tmp = liste[i1]
    liste[i1] = liste[i2]
    liste[i2] = tmp
```

Mache dich in der REPL (dem Taschenrechnermodus) damit vertraut was diese Funktion tut, indem du sie für ein paar verschiedene Eingaben ausprobierst. Tipp: das Argument `liste` sollte eine Variable sein.

Aufgabe 2

Schreibe eine Funktion `tausche_test` ohne Argumente die eine Liste von `int` zurückgibt:

```
def tausche_test() -> list[int]:
```

In der Funktion soll eine Variable beliebigen Namens erzeugt werden, die mit dem Wert `[3, 0, -1, 3, 8, 2]` initialisiert wird. In den folgenden Zeilen soll diese Variable durch wiederholtes Aufrufen der `tausche` Funktion aus Aufgabe 1 sortiert werden. Am Ende von `tausche_test` soll die sortierte Liste zurückgegeben werden.

Aufgabe 3

Ein Verfahren um eine Liste zu sortieren heißt *bubblesort*. Schreibe eine Funktion `bubblesort` die eine Liste `liste` von Ganzen Zahlen übergeben bekommt und ihre Einträge der Größe nach aufsteigend sortiert. Die Idee ist wie folgt. Wir gehen `len(liste)` Mal Paar (zwei Einträge, deren Indizes nur um 1 verschieden sind) für Paar durch `liste` durch und tauschen die Reihenfolge eines Paares, wenn der vordere Eintrag größer als der hintere Eintrag ist. Wieso reichen die `len(liste)` Wiederholungen dieser Prozedur aus, um die Liste garantiert zu sortieren?

Die Sortierung soll *in place* erfolgen, also die Originaliste modifizieren, statt eine neue Liste zu erstellen.

```
def bubblesort(liste: list[int]):
```

(Tipp: zwei geschachtelte `for`-Schleifen)

Aufgabe 4

Ein Verfahren um eine Liste zu sortieren heißt *insertionsort*. Die einfacher umzusetzende Variante füllt eine neue Liste nach und nach mit den Elementen der zu sortierenden Liste, wobei ein Element immer direkt am richtigen Ort eingefügt wird.

```
def insertionsort(liste: list[int])
    -> list[int]:
```

Der Ablauf des Verfahrens sieht im Detail also so aus:

1. Lege eine neue Variable `ergebnis` an, die zu Beginn den Wert `[]` hat.
2. Für jeden Eintrag `x` in `liste`:
   (a) Finde den größten Index `i` in `ergebnis`, so dass entweder `i == 0` oder `x` größergleich dem Eintrag `ergebnis[i - 1]` ist.
   (b) Füge mit `ergebnis.insert(i, x)` den Wert `x` in `ergebnis` ein (`x` steht danach an Index `i`).
3. Gebe zum Schluss `ergebnis` zurück.

Schritt 2a kann wiederum in mehrere Teilschritte unterteilt werden:

1. Lege eine neue Variable `i` an. Diese hat zu Beginn den Wert `len(ergebnis)`.
2. Solange `i > 0` und `ergebnis` an Position `i - 1` einen Eintrag größer als `x` stehen hat:
   (a) Ziehe `1` von `i` ab.

Aufgabe 5

Ein drittes Verfahren eine Liste zu sortieren heißt *mergesort*. Schreibe eine Funktion `mergesort`, die eine Liste `liste` von Ganzen Zahlen nach diesem Verfahren sortiert.

```
def mergesort(liste: list[int])
    -> list[int]:
```

Im Gegensatz zu Bubblesort soll `mergesort` nicht die Originalliste modifizieren, sondern eine neue sortierte Liste zurückgeben.
Mergesort kann in die folgenden Schritte unterteilt werden:

1. Wenn `len(liste) <= 1`: gebe `liste` zurück, diese ist schließlich schon sortiert.
2. Teile `liste` in zwei (fast) gleich große Listen auf.
3. Sortiere beide Teillisten (durch jeweils einen Aufruf von `mergesort`)
4. Erstelle eine (zu Beginn leere) Ergebnisliste
5. Füge Element für Element immer das kleinste noch nicht hinzugefügte Element beider Teillisten zur Ergebnisliste hinzu
6. gebe die Ergebnisliste zurück

Schritt 5 ist dabei der namensgebende Schritt von Mergesort. Umgesetzt werden kann er mit einer Indexvariablen für jede Listenhälfte. Für beide Hälften speichern die jeweiligen Indizes an welcher Stelle das erste Element steht, dass noch nicht in die Ergebnisliste übertragen wurde. Wenn also das nächste Element übertragen werden soll, wird geguckt welche Listenhälfte am jeweiligen Index ein kleineres Element hat. Dieses wird dann in die Ergebnisliste kopiert und der Index einen weitergezählt.
Wenn ein Index so groß wie die Länge dieser Hälfte ist, muss zum Abschluss noch der Rest der anderen Hälfte in die Ergebnisliste übertragen werden.

Aufgabe 6

Was sind Vor- und Nachteile von Bubblesort und Mergesort im Vergleich zueinander?

Aufgabe 7

Die Einträge der Originalliste `liste` werden in der bisherigen Implementierung von Insertionsort der Reihe nach gelesen und dann in `liste` (während des Sortierens) nicht weiter genutzt. Schreibe eine zweite Version von Insertionsort, die keine neue Liste `ergebnis` zurückgibt, sondern wie unsere Bubblesortimplementierung nach und nach das Argument `liste` sortiert. Konzeptionell spielt dann der (größer werdende) Anfang von `liste` die Rolle von `ergebnis`. Um `x` an die richtige Stelle zu bringen kann es immer mit dem Element links von der aktuellen Position getauscht werden, bis dieses nicht mehr größer als `x` ist.

```
def insertionsort_2(liste: list[int]):
```