Universität Potsdam
Institut für Informatik

GdP-Rechnerübung

Aufgabenblatt 10
(zuletzt aktualisiert: 6. September 2023)

Lernziele (zum Abhaken): Die Student:innen können...

☐ die Nachteile der Benutzung von globalen Variablen erläutern.

☐ einen Algorithmus unter der Beachtung der Nachteile sowie der Verwendung von globalen Variablen implementieren.

☐ einen Algorithmus mit einer vorgegebenen Zeiteffizienz implementieren.

☐ die Zeiteffizienz eines selbst erstellten Programmes bestimmen.

☐ für einen vorgegebenen Algorithmus den worst case ermitteln.

☐ innerhalb einer vorgegebenen Umgebung die Laufzeit eines Programmes ermitteln.

☐ ein Testszenario für den Vergleich von zwei Algorithmen entwickeln und implementieren.

19 Globale Variablen

1. Schreiben Sie ein Python-Programm `add.py`.
   Dieses soll eine Funktion `add(a, b)` enthalten, die die Summe von `a` und `b` berechnet. Dabei sollen `a` und `b` vom Benutzer eingegeben werden. Die Funktion soll das Ergebnis **nicht** zurückgeben, sondern in eine globale Variable `c` speichern.

2. Welche Nachteile ergeben sich durch das Benutzen globaler Variablen?

   _____

   _____

   _____

20 Effizienz

1. Gegeben ist eine sortierte Liste `L` bestehend aus den Zahlen $a_1, a_2, a_3, \ldots, a_n$ mit $a_m \leq a_{m+1}$ für alle $m \in [1, n-1]$ und eine Zahl `targetSum`. Es soll überprüft werden, ob die Liste zwei Zahlen a_i und a_j mit $i \neq j$ enthält, die addiert `targetSum` ergeben.

   a) Implementieren Sie hierfür in Python zwei zweistellige Funktionen f_1 und f_2 mit

      i. $t_{f_1} \in \theta(n^2)$

      Hinweis: Vergleichen Sie die Summe jedes Paars von Elementen mit `targetSum`.

      ii. $t_{f_2} \in \theta(n)$

      Hinweis: Nutzen Sie die Eigenschaft, dass die Liste sortiert ist und vergleichen Sie die Summe aus a_1 und a_n zuerst. Überlegen Sie sich, welche Elemente Sie als nächstes vergleichen, wenn das Ergebnis zu klein oder zu groß ist.

   Die Funktionen sollen jeweils einen Wahrheitswert (`True` oder `False`) zurückgeben.

b) Vergleichen Sie die Laufzeiten beider Funktionen im *worst case*.

i. Unter welchen Bedingungen tritt der *worst case* für die beiden Funktionen ein?

ii. Verwenden Sie jetzt das Python-Modul `profile` wie folgt, um die tatsächliche Laufzeit der Funktion auf Ihrem Rechner zu messen.[1] Benutzen Sie hierzu eine Beispielliste wie z.B.:

```
1 profile.run("sum1([1, 2, 3], 4)")
```

Welche Gesamtlaufzeit wird ausgegeben?

iii. Um Eingabelisten für die Funktionen zu erzeugen, schreiben Sie eine dreistellige Funktion, die eine sortierte Liste aus `n` ganzen Zufallszahlen (zwischen den Intervallgrenzen `start` und `stop`) generiert.

Hinweis: Importieren Sie dafür das Modul `random`*. Es bietet die Funktion* `randint(a, b)`*, die eine zufällige Zahl* n *mit* $a \leq n \leq b$ *zurückgibt:*

```
1 random.randint(start, stop)
```

Eine Liste können Sie mit der Funktion `sort()` *sortieren:*

```
1 L = [3,2,5,1]
2 L.sort()      # -> L = [1,2,3,5]
```

iv. Testen Sie nun Ihre beiden Funktionen mit längeren Listen: Wie lange brauchen sie jeweils für eine sortierte Liste mit 4000 zufälligen Elementen im *worst case*?

Benutzen Sie `profile.run()`, um die Laufzeiten zu messen!

[1] Angenommen, Sie haben Ihre Funktion `sum1` genannt.