ALP II, SS 2012
Prof. Dr. Margarita Esponda

4. Übungsblatt

Ziel: Auseinandersetzung mit Sortieralgorithmen.

1. Aufgabe (4 Punkte)

Schreiben Sie eine Variante des Quicksort-Algorithmus, der das mittlere Element des Array als Pivot-Element verwendet. Mit dieser Variante versuchen wir den quadratischen Aufwand zu umgehen, wenn die Zahlen bereits sortiert sind.

a) Ist Ihre Quicksort-Variante stabil? Begründen Sie Ihre Antwort.

b) Wann ist es im allgemeinen unwichtig, nicht stabile Algorithmen zu verwenden?

2. Aufgabe (5 Punkte)

a) Implementieren Sie eine Variante des Shellsort-Algorithmus, der anstatt des Insertionsort-Algorithmus den Bubblesort-Algorithmus für das Sortieren innerhalb der Segmente verwendet und Mersenne-Zahlen für die Segmentierung benutzt.

b) Wann es ist sinnvoll, den Insertsort- oder den Shellsort-Algorithmus zu verwenden? Begründen Sie Ihre Antwort.

3. Aufgabe (5 Punkte)

a) Programmieren Sie eine rekursive Lösung des Mergesort-Algorithmus in Python, indem ein Hilfsarray am Anfang erzeugt wird, der genau so groß wie die zu sortierenden Zahlenmengen ist, und der als Zwischenlagerung von Teilmengen des Arrays innerhalb der **merge**-Funktion verwendet wird. Teilmengen, die kleiner 8 sind, sollen nicht mehr mit Mergesort sondern mit Bubblesort sortiert werden.

b) **Freiwillige Aufgabe** (2 Bonuspunkte) Schreiben Sie eine iterative Variante des Mergesort-Algorithmus.

4. Aufgabe (6 Punkte)

Schreiben Sie eine eigene **isSorted**-Funktion und verwenden Sie diese in einer Test-Funktion, die Zufallszahlen erzeugt und unter Verwendung ihrer **isSorted**-Funktion und **assert**-Anweisungen ihrer Sortieralgorithmen aus den Aufgaben **1-3** testet und pragmatisch vergleicht.

5. Aufgabe (3 Punkte)

Schreiben Sie ein möglichst effizientes Python-Programm, das bei Eingabe einer Zahlenmenge, die zwei Zahlen findet, deren Werte am nächsten sind. Analysieren Sie die Komplexität Ihres Algorithmus.

6. Aufgabe (12 Punkte)

In der Linguistik-Forschung wird oft für die Untersuchung von Sprachpatterns innerhalb von Sprachen, die Häufigkeit des Vorkommens von Worten als erstes analysiert.

a) Schreiben Sie ein Python-Programm, das einen Text liest, und die Häufigkeit des Vorkommens der verschiedenen Wörter zuerst zählt und dann mit Hilfe eines Mergesort-Algorithmus nach ihrer Häufigkeit sortiert und in eine Liste zurückgibt.

b) Testen Sie Ihr Programm mit den auf unserer Webseite vorgegebenen Texten auf Englisch, Deutsch und Spanisch.

c) Versuchen Sie das Programm so gut wie möglich zu strukturieren. Schreiben Sie dafür verschiedene Funktionen. Z. B. eine Funktion für das Lesen des Textes, eine für die Berechnung der Häufigkeiten, eine für die Sortierung der Häufigkeiten und eine, die sich zum Schluss um die Ausgabe der Ergebnisse kümmert.

d) **Freiwillige Aufgabe** (3 Bonuspunkte). Sie können eine Funktion schreiben, die eine schöne graphische Ausgabe der Ergebnisse mit Hilfe der Python-Funktionen für die Generierung von PostScript-Dateien erstellt, schreiben. Die Funktion soll natürlich nur die häufigsten Worte graphisch darstellen.

7. Aufgabe (3 Punkte)

Ein Element einer Liste von n Objekten stellt die absolute Mehrheit der Liste dar, wenn das Element mindestens $\left(\frac{n}{2}+1\right)$ mal in der Liste vorkommt. Schreiben Sie

eine majority-Funktion, die mit linearem Aufwand das Majority-Element der Liste findet, wenn eine existiert oder sonst **None** zurückgibt.

8. Aufgabe (4 Punkte)

Schreiben Sie eine Variante des LRS-Algorithmus aus der Vorlesung, um in der folgenden DNA-Sequenz die längste sich wiederholende Subsequenz, die am Ende die Buchstaben TAC hat, zu finden.

AAGAGCAGTGCCTCCTTTGGTGAAGGTGACACATCATGTGACCTCTTCAGTGACCACTCTACGGTGT
CGGGCCTTGAACTACTACCCCCAGAAAGAGCAGACATCACCATGAAGTAAGAGCAGGGCTGAAGGA
TAAGCAGCCAATGGATGCACAAGGAGTTCGAACCTAAAGACGTATTGCCCAATGGGGATGGGACCT
ACCAGGGCTGGATAAAGAGCAGACCTTGGCTGTACCCCCTGGGGAAGAGCAGAGATATACGTACCA
GGTGGAGCACCCAGGCCTGGATCAGCCCCTCATTGTGATCTGAAGAAGGG

Wichtige Hinweise:

1) Sie dürfen keine vorprogrammierte Python-Funktion, um Daten zu sortieren, verwenden. Ziel dieses Übungsblatt ist es, Sortieralgorithmen selber zu programmieren.

2) Verwenden Sie geeignete Namen für Ihre Variablen und Funktionsnamen, die den semantischen Inhalt der Variablen oder die Funktionalität der Funktionen darstellen.

3) Verwenden Sie vorgegebene Funktionsnamen, falls diese angegeben werden.

4) Kommentieren Sie Ihre Programme.

5) Verwenden Sie geeignete Hilfsvariablen und Hilfsfunktionen in Ihren Programmen.

6) Löschen Sie alle Programmzeilen und Variablen, die nicht verwendet werden.

7) Schreiben Sie getrennte Test-Funktionen für alle 5 Aufgaben für Ihren Tutor.