SoSe 2014
Prof. Dr. Margarita Esponda
ALP2
4. Übungsblatt (Abgabe: Mo., den 19.05.)

Ziel: Auseinandersetzung mit Sortieralgorithmen und die Analyse der Komplexität von Algorithmen.

1. Aufgabe (12 Punkte)

a) Definieren Sie eine **is_sorted** Hilfsfunktion, die bei Eingabe einer Liste kontrolliert, ob die Elemente der Liste sortiert sind. Das Ergebnis sollte gleich 1 oder -1 sein, wenn die Elemente jeweils in ansteigender bzw. absteigender Reihenfolge sortiert sind und 0, wenn die Elemente unsortiert sind.

b) Definieren Sie eine **generate_random_list** Hilfsfunktion, die bei Eingabe eines Wertebereichs **(a, b)** eine Liste mit zufälligen Zahlen zwischen **a** und **b** generiert.

c) Programmieren Sie eine rekursive Lösung des Mergesort-Algorithmus, in dem am Anfang ein Hilfsarray erzeugt wird, das genau so groß wie die zu sortierenden Zahlenmengen ist und das als Zwischenspeicherung von Teilmengen des Arrays innerhalb der merge-Funktion verwendet wird. Teilmengen, die kleiner gleich 9 sind, sollen nicht mehr mit Mergesort, sondern mit Bubblesort sortiert werden.

d) Schreiben Sie eine iterative Variante des Algorithmus, die die Zahlen mit Hilfe eines Hilfsarrays sortiert.

2. Aufgabe (4 Punkte)

Wir haben in der Vorlesung gesehen, dass sich beim Bubblesort Algorithmus die Positionen der Zahlen oft von der richtigen Endposition entfernen.

Programmieren Sie eine Variante des Bubblesort-Algorithmus, der diese Verschlechterung berechnen kann, indem aufgezählt wird, wie groß die Anzahl der Bewegungen, die in die falsche Richtung gehen, ist.

3. Aufgabe (6 Punkte)

a) Definieren Sie eine Variante des Countingsort-Algorithmus aus der Vorlesung, bei dem Sie die Elemente **in-Place** sortieren. Ihre Countingsort-Variante muss nicht stabil sein. Sie dürfen als einzigen zusätzlichen Speicher das Hilfsrarray C verwenden (siehe Vorlesungsfolien). Das Hilfsarray B darf nicht mehr verwendet werden.

b) Könnten Sie mit Ihrem Algorithmus Personen anhand ihres Geburtsdatum sortieren? Begründen Sie Ihre Antwort.

c) Analysieren Sie die Komplexität ihres Algorithmus.

4. Aufgabe (14 Punkte)

a) Zeigen Sie mit Hilfe eines Zahlenbeispiels, dass der Heapsort-Algorithmus nicht stabil ist.

Nehmen Sie an, wir bekommen Nachrichten, die mit Prioritäten, die in einem Wertbereich zwischen 1 und 50 liegen, abgearbeitet werden müssen. Die Abarbeitung der Nachrichten soll streng nach Prioritäten erfolgen. Bei Nachrichten mit der gleichen Priorität soll die zeitliche Reihenfolge des Empfangs einer Nachricht respektiert werden.

$$(p_1, t_1, m_1), (p_2, t_2, m_2),...,(p_i, t_i, m_i),...$$

b) Programmieren Sie zuerst mit Hilfe der **yield**-Anweisung eine Funktion **next_message**, die bei jedem Aufruf der Funktion eine neue Nachricht mit zufälliger Priorität und entsprechendem Zeitstempel produziert.

Simulieren Sie unter Verwendung einer ***heap***-Struktur (siehe Vorlesungsfolien) eine Warteschlange, bei der die Nachrichten nach Priorität und Zeit abgearbeitet werden (*Priority Queue*).

Eine Prioritätswarteschlange speichert Objekte nach einer Prioritätseigenschaft und entfernt immer als erstes das Objekt mit der höchsten Priorität.

Vorgehensweise:

c) Schreiben Sie zuerst folgende vier Hilfsfunktionen:

```
def insert ( message_queue, message )
    """ Eine neue Nachricht wird in die Prioritätswarteschlange eingefügt """
def is_empty ( message_queue )
    """ gibt einen Wahrheitswert als Rückgabewert zurück, je nachdem, ob die
        Prioritätswarteschlange leer ist oder nicht """
def remove_message ( message_queue )
    """ Die Nachricht mit der höchsten Priorität und die, die zeitlich zu erst
    produziert worden ist, wird aus der Prioritätswarteschlange entfernt und als
    Ergebnis der Funktion zurückgegeben """
def sort_messages ( message_queue )
```

d) Schreiben Sie eine Funktion **simulate_message_traffic**, die zufällig Nachrichten einfügt oder entfernt. Die Nachrichten sollen mit Hilfe der **next_message** Funktion produziert werden. Wenn die Prioritätswarteschlange zwischendurch leer ist, soll die **remove_message**-Funktion den Wert **None** zurückgeben.

Schreiben Sie getrennte Test-Funktionen für alle Aufgaben für Ihren Tutor und verwende Sie dabei die Hilfsfunktionen der 1. Aufgaben.

Wichtige Hinweise:

1) **Sie dürfen keine vorprogrammierte Python-Funktion verwenden, um Daten zu sortieren. Ziel dieses Übungsblattes ist es, Sortieralgorithmen selber zu programmieren.**
2) **Verwenden Sie geeignete Namen für Ihre Variablen und Funktionen, die den semantischen Inhalt der Variablen oder die Funktionalität der Funktionen darstellen.**
3) **Verwenden Sie vorgegebene Funktionsnamen.**
4) **Kommentieren Sie Ihre Programme.**
5) **Verwenden Sie geeignete Hilfsvariablen und definieren Sie sinnvolle Hilfsfunktionen in Ihren Programmen.**
6) **Löschen Sie alle Programmzeilen und Variablen, die nicht verwendet werden.**