## *Übungsaufgaben*

**Zu 1.**

a) Programmiere `public boolean istAufsteigendSortiert(List pList)`
pList soll hier nur Strings enthalten.

```
public boolean istAufsteigendSortiert(List pList){
    // Standardwert fuer ergebnis: true
    // Man geht davon aus, dass die Liste sortiert ist
    // und sucht dann einen Fehler.
    boolean ergebnis = true;
    pList.toFirst();
    while(pList.hasAccess()){
        String aktuell = (String) pList.getObject();
        pList.next();
        if(pList.hasAccess() == false){
            // Ende der Liste erreicht und keinen Fehler gefunden!
            ergebnis = true;
            return ergebnis;
        }
        String naechster = (String) pList.getObject();
        if(aktuell.compareTo(naechster)>0){
            // Aktuell steht im Alphabet NACH naechster!
            // Fehler gefunden!
            ergebnis = false;
            return ergebnis;
        }
    }
    // Ende der Liste erreicht und keinen Fehler gefunden
    ergebnis = true;
    return ergebnis;
}
```

b) Gesucht ist eine Methode
`public void loesche(List pList, List loeschListe)`
Die Methode soll aus pList alle Elemente löschen, die in loeschListe enthalten sind.

1. Beschreibe umgangssprachlich das Verfahren.

- loeschListe mit einer Schleife durchlaufen und jeweils das aktuelle Element (=aktuellZuLoeschen) auslesen.
  - In einer inneren Schleife pList durchlaufen und das jeweils aktuelle Element (=aktuell) mit aktuellZuLoeschen vergleichen.
    - Wenn aktuell == aktuellZuLoeschen, dann wird aktuell gelöscht.

2. Programmiere das Verfahren.

```
public void loesche(List pList, List loeschListe){
    loeschListe.toFirst();
    while(loeschListe.hasAccess()){
        Object aktuellZuLoeschen = loeschListe.getObject();
        pList.toFirst();
        while(pList.hasAccess()){
            Object aktuell = pList.getObject();
            if(aktuell == aktuellZuLoeschen){
                pList.remove();
            }
            else{
                pList.next();
            }
        }
    }
}
```

3. pList habe n Elemente und die loeschListe ungefähr n/10 Elemente.
   In welche Laudauklasse gehört die Laufzeit deines Verfahrens? Begründe!

Das Verfahren gehört in die Klasse O(n^2). Man hat n/10 Elemente, die zu löschen sind. Für jedes dieser Elemente muss man die ganze pList durchlaufen und jeweils vergleichen. D.h. man muss für jedes Element, das zu löschen ist, n Vergleiche durchführen.
D.h. insgesamt hat man (n/10)*n Vergleiche = n^2/10 Vergleiche
Damit gehört das Verfahren in die Klasse O(n^2).

4. *Kann es ein Verfahren geben, das diese Aufgabe in O(n*log(n)) Zeit erledigt?*

Ja. Man sortiert pList und loeschListe jeweils mit Mergesort. Aufwand dafür ist in der Klasse O(n*log(n)). Dann durchläuft man pList und loeschListe parallel von vorne bis hinten. Dadurch braucht man nur so viele Vergleiche, wie pList und loeschListe zusammen Elemente haben, d.h. n + n/10 Vergleiche. n + n/10 ist auch in der Klasse O(n*log(n)), d.h. insgesamt hat man ein Verfahren, das nur O(n*log(n)) Zeit braucht.

**Zu 2.**

Gegeben ist die Klasse Person (s.u.). Die Listen in den folgenden Aufgaben seien nur mit Personen gefüllt.

| **Person** |
|---|
| - name: String<br>- geburtstag: String |
| + Person(pName: String, pGeburtstag: String)<br>+ getName(): String<br>+ getGeburtstag(): String<br>+ toString(): String |

a) Programmiere `public void sortierenDurchAuswaehlen(List pList)`

Vgl. sibi-wiki.de und dort der Artikel zu List.

b) Programmiere `public List mergesort(List pList).`

Hier kann man voraussetzen, dass die Methode `public List merge(List l1, List l2)` und die Methode `public List teile(List pList)` schon gegeben ist.

Berücksichtige bei der Programmierung die nötigen Abbruchbedingungen.

Vgl. sibi-wiki.de und dort der Artikel zu List.

**Zu 3.**

Gegeben ist folgender Algorithmus zum Sortieren einer Liste pList:

- Es wird eine leere Datenstruktur für das Ergebnis erstellt.
- Jetzt wird mit einer Schleife pList durchlaufen. Bei jedem Schleifendurchlauf wird...
  - das jeweils aktuelle Element aus pList entnommen (=aktuell)
  - dann mit dem mittleren Element von ergebnis verglichen; wenn aktuell kleiner ist als das mittlere Element, dann wir es mit dem Element bei ¼ verglichen, sonst mit dem Element bei ¾ .
  - Der Algorithmus setzt sich so fort, bis das Element eingefügt werden kann.

a) Welche Vergleichsoperationen sind nötig, wenn aktuell = 112 eingefügt werden soll und ergebnis schon die folgenden Elemente enthält:
17, 33, 47, 81, 107, 133, 147

Man muss 112 mit folgenden Elementen vergleichen: 81, 133, 107. Dann weiß man, dass man die 112 zwischen 107 und 133 einfügen kann.

b) Schätze die Anzahl der Operationen für n = 1024 Elemente geeignet ab. Berücksichtige dabei, dass es Entnahmeoperationen, Vergleichsoperationen und Einfügeoperationen gibt.
Für 1024 Elemente braucht man:

- 1024 Entnahmeoperationen
- 1024 Einfügeoperationen.

Die Anzahl der Vergleichsoperationen lässt sich (großzügig!) abschätzen, wenn man überlegt, wie viele Vergleiche man braucht, wenn man das letzte Element einfügt. Der Einfügevorgang entspricht hier der Suche in einem Telefonbuch mit 1024 Elementen. Man vergleicht erst mit dem Element in der Mitte; dadurch wird die Anzahl der Elemente, die noch in Frage kommen halbiert (=512). Dann vergleicht man wieder mit dem Element in der Mitte der verbleibenden Elemente, d.h. es wird wieder halbiert, usw.
D.h. insgesamt hat man jeweils einen Vergleich für folgende Element-Anzahlen:
1024, 512, 256, 128, 64, 32, 16, 8, 4, 1

d.h. 10 Vergleiche.
Insgesamt hat man also weniger als 10240 Vergleiche.

c) Entscheide anhand deiner Ergebnisse, ob das Verfahren zur Landau-Klasse O(n*log(n)) gehört.

Ja, das Verfahren ist O(n*log(n)). Begründung: Die meisten Operationen entstehen beim Vergleichen. Für 1024 Elemente braucht man weniger als 10240 Operationen.
$10240 = 1024 * 10 = 1024 * \log_2(1024)$.
Verallgemeinert auf n: Für n Elemente braucht man $n * \log_2(n)$

d) Warum kann man für dieses Verfahren nicht die Datenstruktur List verwenden?

Das Einfügen an einer bestimmten Position kann man mit der Datenstruktur List nicht realisieren, weil man hier Elemente nicht nach ihrer Nummer ansprechen kann (z.B. das 512te Element). Wenn man die Liste dafür jeweils von vorne aus durchlaufen würde, dann hätte man für jedes Einfügen einen Aufwand von jeweils n "Weiter-Geh-Operationen", was zu einer Laufzeit in der Klasse O(n^2) führen würde.

**Zu 4.**

a) Ein Algorithmus hat eine Anlaufzeit von 15 Sekunden. Dann braucht er $5*n^2 + 130*n$ Operationen, um die verlangte Aufgabe für n Elemente zu erledigen. Gehört dieser Algorithmus zur Landau-Klasse $O(n^2)$?

In der Anlaufzeit von 15 Sekunden werden eine zwar unbekannte, aber begrenzte Zahl von Operationen durchgeführt. Diese Zahl wird jetzt C genannt:

C: Anzahl der Operationen in der Anlaufzeit von 15s.
$f(n)$ ist dann: $5*n^2 + 130*n + C$
$f(n) / n^2 = 5 + 130/n + C/ n^2$
Für n gegen unendlich nähert sich $5 + 130/n + C/ n^2$ immer näher der 5, ist also kleiner unendlich.
Damit dieser Algorithmus zur Landau-Klasse $O(n^2)$

b) Überprüfe, ob $f(n) = 0{,}05*(\log(n))^2$ zur Landauklasse O(log(n)) gehört.

$0{,}05*(\log(n))^2 / \log(n) = 0{,}05*\log(n)$.
$0{,}05*\log(n)$ geht gegen unendlich, wenn n gegen unendlich geht.
D.h. f(n) gehört NICHT zur Klasse O(log(n)).

c) Überprüfe, ob $f(n) = (\log(n))^2$ zur Landauklasse O(n) gehört.

$(\log(n))^2 / n$ geht für große n gegen 0 (Taschenrechner!). D.h. f(n) gehört zu O(n).

d) Weise nach, dass $f(n) = \log_{1,1}(n)$ zur Landauklasse $O(\log_2(n))$ gehört.
Nutze dabei die folgende Gesetzmäßigkeit aus: $\log_a(n) = \log_2(n)/\log_2(a)$

$\log_{1,1}(n) / \log_2(n) = (\log_2(n) / \log_2(1,1)) / \log_2(n) = 1 / \log_2(1,1)$
$1 / \log_2(1,1)$ ist konstant – auch für große n. Damit ist f(n) in der Klasse $O(\log_2(n))$

**e) <u>Erkläre anhand dieses Beispiels, warum bei der Landau-Klasse O(log(n)) die Angabe der Logarithmus-Basis egal ist.</u>**

Die Rechnung oben kann man für 2 beliebige Logarithmus-Basen durchführen. D.h. allgemein ist:
$\log_a(n) / \log_b(n) = (\log_b(n) / \log_b(a)) / \log_b(n) = 1 / \log_b(a)$
Und damit gehört $\log_a(n)$ zur Klasse $O(\log_b(n))$.

f)
*Ein Programmierer hat die Methode*
***`public int[] meinLieblingsSortierVerfahren(int[] array)`***
*entwickelt. Sie testen den Algorithmus auf einem Computer und erhalten folgende Messwerte:*

| *Anzahl der Elemente* | *100* | *1000* | *10000* | *100000* | *1000000* |
|---|---|---|---|---|---|
| *Zeit (in Mikrosekunden[1])* | *470* | *6063* | *81341* | *1032977* | *12754334* |

**Aufgabe:**
Untersuche, ob es sich bei dem Algorithmus um ein Verfahren handeln kann, das in O(n*log(n)) Zeit sortiert. Begründe deine Aussage.

Man ergänzt die Tabelle um zwei Zeilen: In der einen trägt man O(n*log(n)) ein, wobei man der Einfachheit halber den Logarithmus zur Basis 10 nimmt. (Dass die Basis egal ist, wurde ja in Aufgabe e nachgewiesen!).
Dann geht man davon aus, dass in jeder Mikrosekunde eine feste Zahl von Operationen berechnet wird.
Wenn der Algorithmus in der Klasse $O(n*\log_{10}(n))$ sein soll, dann muss der Quotient aus Zeit und $O(n*\log_{10}(n))$ konstant sein. Dafür berechnet man diesen Quotienten in der 4. Zeile.

| *Anzahl der Elemente* | *100* | *1000* | *10000* | *100000* | *1000000* |
|---|---|---|---|---|---|
| *Zeit (in Mikrosekunden[2])* | *470* | *6063* | *81341* | *1032977* | *12754334* |
| *$O(n*\log_{10}(n))$* | *200* | *3000* | *40000* | *500000* | *6000000* |
| ***Zeit / $O(n*\log_{10}(n))$*** | ***2,35*** | ***2,02*** | ***2,03*** | ***2,07*** | ***2,13*** |

Der Quotient in der 4. Zeile ist annähernd konstant, d.h. der Algorithmus kann in der Klasse O(n*log(n)) sein.

1 Eine Mikrosekunde ist $10^{-6}$ Sekunden.
2 Eine Mikrosekunde ist $10^{-6}$ Sekunden.