MPGI 1 Probeklausur – Lösungen
(inoffizielle Lösungen - keine Garantie)

1. Aufgabe (Kundendatenbank)
   1. Induzierte Signatur

Es handelt sich um einen Produkttyp, da nur ein Konstruktor vorhanden ist (kunde).

Induzierte Signatur:
SORT Kunden
    (Sorte, Name)
FUN kunde: denotation ** nat ** Eissorte ** Betrag → Kunden
    (Konstruktor)
FUN name: Kunden → denotation
FUN alter: Kunden → nat
FUN lieblingssorte: Kunden → Eissorte
FUN umsatz: Kunden → Betrag
    (alles Selektoren)
FUN kunde? : Kunden → bool
    (Diskriminator)

   2. Erweitern des Datentyps

```
TYPE Kunden == kunde(name: denotation,
                     alter: nat,
                     lieblingssorte: Eissorte,
                     umsatz: Betrag)
               grosskunde(name: denotation,
                     firmenname: denotation,
                     lieblingssorte: Eissorte,
                     umsatz: Betrag)
```

Total ~ Funktionen die ohne Diskriminator funktionieren, und in beiden Fällen (kunde,grosskunde) einen Wert liefert.
=> name, lieblingssorte,umsatz

   3. Addition

```
FUN + : Betrag ** Betrag -> Betrag
DEF betrag(euro1,cent1) + betrag(euro2,cent2) ==
        IF ((cent1+cent2) > 100) THEN
            betrag((euro1+euro2+1),(cent1+cent2-100))
        ELSE
            betrag((euro1+euro2),(cent1+cent2))
        FI
```

4. Preiserhöhung

```
FUN sortenUmsatz: seq[Kunden] ** Eissorte -> Betrag
DEF sortenUmsatz(<>,_) == betrag(0,0)
DEF sortenUmsatz(k::R,Esort) ==
        IF lieblingssorte(k) = Esort THEN
            umsatz(k) + sortenUmsatz(R,Esort)
        ELSE
            sortenUmsatz(R,Esort)
        FI
```

5. Vergleichsoperator

```
FUN < : Betrag ** Betrag -> bool
DEF betrag(euro1,cent1) < betrag(euro2,cent2) ==
        IF euro1 < euro2 THEN
            true
        ELSE
            IF (euro1 = euro2) and (cent1 < cent2) THEN
                true
            ELSE
                false
            FI
        FI
```

6. Suchen von guten Kunden

```
FUN guteKunden: seq[Kunden] -> seq[denotation]
DEF guteKunden(<>) == <>
DEF guteKunden(k::R) ==
                IF kunde?(k) THEN
                    IF betrag(24,"50"!) < umsatz(k) THEN
                        name(k) :: guteKunden(R)
                    ELSE
                        guteKunden(R)
                    FI
                ELSE --grossKunde
                    IF betrag(1000,0) < umsatz(k) THEN
                        name(k) :: guteKunden(R)
                    ELSE
                        guteKunden(R)
                    FI
                FI
```

2. Aufgabe (Listenfunktionale)
    1. Zeiterfassung

```
FUN time: seq[auftrag] -> nat
DEF time(sA) == reduce(+,0)(map(\\a. bZeit(a))(sA))
```

2. Langsamer Service

```
FUN slowService: seq[auftrag] ** nat -> seq[auftrag]
DEF slowService(sA,m) == filter(\\a. bZeit(a) > m)(sA)
```

3. Langsamster Schalter

```
FUN slowest: seq[auftrag] -> (anliegen -> bool) -> nat
DEF slowest(A)(f) == LET B == filter(choose(f))(A) -- Liefert eine
liste die dem Anliegen entsprechen zurück
                                        C == reduce(\\a,b. IF bZeit(a) <
bZeit(b) THEN b ELSE a FI , ft(B))(B)
                                    IN
                                        schalter(C)
FUN choose : (anliegen -> bool) -> auftrag -> bool
DEF choose(f)(a) == f(art(a))
```

3. Aufgabe (Lambda-Kalkül)
   1. Induktiver Aufbau

a) gültig
b) nicht gültig, da $\Lambda(xy)$ nicht definiert
c) gültig
d) nicht gültig, da $(\Lambda . x)$ nicht definiert

   2. Substitution

$$
\begin{aligned}
&(\lambda y.(x\,y)(\lambda x.x))[(x\,y)/x] \quad * \\
&\equiv_{\alpha}(\lambda z.(x\,z)(\lambda x.x))[(x\,y)/x] \\
&=(\lambda z.((x\,z)(\lambda x.x))[(x\,y)/x]) \\
&=(\lambda z.(x\,z)[(x\,y)/x](\lambda x.x)[(x\,y)/x]) \\
&=(\lambda z.(x[(x\,y)/x]z[(x\,y)/x])(\lambda x.x)) \\
&=(\lambda z.((x\,y)z)(\lambda x.x))
\end{aligned}
$$

$\alpha - Konversion\ ist\ bei\ *\ notendig,$
$da\ das\ y\ aus\ (x\,y)\ gebunden\ werden\ würde.$

4. Aufgabe (Aufwand)
   1. O-Kalkül

Konstante Faktoren, tragen nicht zur Aufwandsklasse bei, somit können Sie vernachlässigt werden. Es folgt:

$$\lim_{n\to\infty}\left(\frac{n^2-n}{n^2}\right)=\lim_{n\to\infty}\left(\frac{n^2}{n^2}-\frac{n}{n^2}\right)=1-\lim_{n\to\infty}\left(\frac{1}{n}\right)=1-0=1$$

$\Rightarrow Durch\ Grenzwertbetrachtung\ ist\ gezeigt\ f(n)\gg g(n)\Rightarrow f\in\theta(g)$

Alternative: siehe nächste Seite.

Es muss aufgrund der Definitionen gelten :

$$1)\ ist\ f \in O(g) \Rightarrow \lim_{n\to\infty}\left(\frac{f(n)}{g(n)}\right) = 0$$

$$2)\ ist\ f \in \Omega(g) \Rightarrow \lim_{n\to\infty}\left(\frac{f(n)}{g(n)}\right) = \infty$$

$$3)\ ist\ f \in \theta(g) \Rightarrow \lim_{n\to\infty}\left(\frac{f(n)}{g(n)}\right) = konstant$$

$$f \in \theta(g)\ bedeutet\ auch,\ f \in O(g) \subset \Omega(g)$$

$$\lim_{n\to\infty}\left(\frac{f(n)}{g(n)}\right) = \lim_{n\to\infty}\left(\frac{\pi * n^2 - n}{e * n^2 + 4}\right) = \lim_{n\to\infty}\left(\frac{\frac{\pi - n}{n^2}}{\frac{e+4}{n^2}}\right) = \ldots = \frac{\pi}{e} = konstant$$

$$\Rightarrow f \in \theta(g)\ und\ somit\ gilt\ auch\ f \in O(g)\ und\ f \in \Omega(g)$$

2. Rekkurrenzgleichungen

$$A_{\text{to set}}(n) = A_{\text{in?}}(n) + A_{\text{to set}}(n-1) \qquad A_{\text{in?}}(n) = A_{\text{in?}}(n-1)$$

Die Funktion in? *kann nach der* 1.Rek. *Gleichung mit dem Aufwand* $O(n)$ *abgeschätzt werden.*

Dieses setzen wir nun in *die Funktion* to set *ein. Daraus folgt* :

$$A_{\text{to set}}(n) = n + A_{\text{to set}}(n-1)$$

Nach der erste Rek.Gleichung folgt :

$$\text{k=1} \Rightarrow O(n^{k+1}) \Rightarrow O(n^2).$$

Somit hat *die Funktion quadratischen Aufwand* .

5. Aufgabe (Bäume)
   1. Allgemeine Fagen

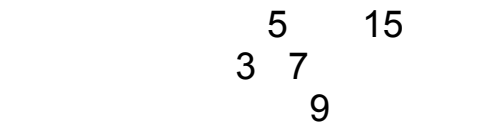
      Ja, Ja, Nein, Nein

   2. Traversierung

      a) A,D,O,P,I,W,M,X,Y
      b) P,O,I,D,W,A,X,M,Y
      c) Höhe: 3 , Größe: 9
      d) <A,D,M,O,W,X,Y,P,I>

   3. AVL-Bäume

      1. Einfügen der 10,5,15,3,7,9 =>



      AVL-Bed. (Höhe) verletzt: Doppelrotation

      Fertig.

4. Bäume

a) Kein Heap, da nicht immer das größte/kleinste Element in der Wurzel steht, Ist ein AVL-Baum, da Balanciert und Suchbaum
b) Maxheap, da jeweils das größte Element in den Knoten steht, linksvoll, Kein Avl, da kein Suchbaum
c) Kein Heap, da nicht immer das größte Element in den Knoten steht (siehe Knoten 34,35), Kein AVL da das Element die Suchbaumeigenschaft verletzt.

6. Aufgabe (Sortierverfahren)
   1. Insertionsort
   2. Quicksort
7. Aufgabe (Ein- und Ausgabe)

```
FUN konkatenator: com[void]
DEF konkatenator ==
        ask("Zahl > ") &
        (\\n:nat. hkon(n)) &
        (\\A:seq[denotation]. succeed(reduce(++,"")(A)) &
        (\\c:denotation. writeLine(c)))

FUN hkon: nat -> com[seq[denotation]]
DEF hkon(0) == succeed(<>)
DEF hkon(n) ==
       hkon(n-1) &
       (\\A. ask("> ") &
       (\\a:denotation. succeed(a::A)))
```