Musterlösung Übung 5

Algorithmen, Komplexität von Algorithmen

1. Betrachten Sie den folgenden Algorithmus in Pseudocode:

```
multi := 1;
FOR i := 1 TO 10 DO
 multi := multi * i;
```

   (a) Welchen Wert hat die Variable `multi` nach der Schleife?

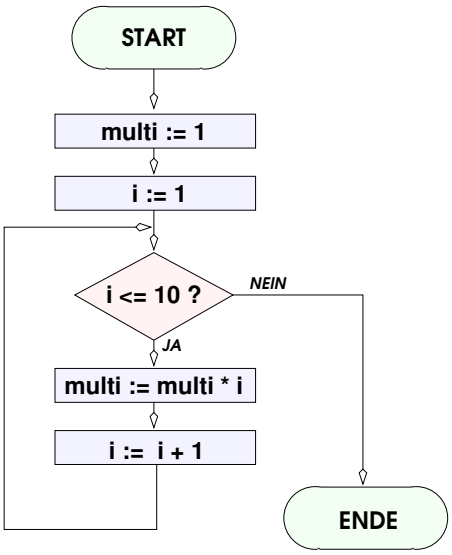
   Die Schleife wird verlassen, wenn `i` *den Wert 10 erreicht hat und der Anweisungsblock anschließend das letzte Mal ausgeführt wurde.* `i` *hat zu diesem Zeitpunkt also den Wert 11. Bis dahin ist* `multi` *auf den Wert* $1 \cdot 2 \cdot 3 \cdot 4 \cdot 5 \cdot 6 \cdot 7 \cdot 8 \cdot 9 \cdot 10$ *angewachsen, also* $10! = 3628800$.

   (b) Wie verhält sich der Algorithmus bezüglich

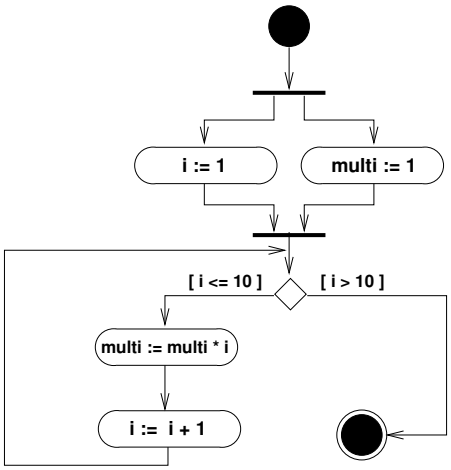
   i. Statischer Finitheit,
   ii. Dynamischer Finitheit,
   iii. Terminierung?

   Der Algorithmus ist ***statisch finit****, da er sich mit endlich vielen Befehlen genau aufschreiben lässt,* ***dynamisch finit****, da er nur endlich viele Variablen/Resourcen beansprucht, und er* ***terminiert*** *nach genau 10 Durchläufen der* `FOR`*-Schleife (ansonsten würde man auch nicht auf das Ergebnis in der vorigen Aufgabe kommen).*

   (c) Stellen Sie den Algorithmus als Flußdiagramm (bzw. alternativ als UML Aktivitätsdiagramm) dar.



Flussdiagramm



(UML) Aktivitätsdiagramm

2. Entwerfen Sie einen Algorithmus in Pseudocode, der mittels eines Schleifenkonstruktes eine Variable beginnend von 100 bei jedem Durchlauf um 10 erniedrigt. Ist der Algorithmus **statisch finit**, **dynamisch finit**, **terminierend**?

```
variable := 100;
WHILE ( 1 = 1 ) DO
 variable := variable - 10;
```

Der Algorithmus ist ***statisch finit****, denn er lässt sich vollständig aufschreiben. Er ist ebenfalls (mit der Einschränkung, dass* `variable` *eine endliche Variable ist) auch* ***dynamisch finit****, aber, da sowohl in der umgangssprachlichen als auch in der Pseudocode-Version des Algorithmus kein Abbruchkriterium für die Wiederholung der Subtraktionsanweisung vorgesehen ist,* ***nicht terminierend****.*

Siehe auch ***http://de.wikipedia.org/wiki/Algorithmus#Finitheit***

3. Formulieren Sie den „umgangssprachlichen“ Algorithmus aus der Vorlesung, der den größten gemeinsamen Teiler zweier Zahlen nach Euklid berechnet, in Pseudocode.

   (a) Sei A die Größere der beiden Zahlen A und B (entsprechend vertauschen, falls dies nicht bereits so ist)

   (b) Setze A auf den Wert $A - B$

   (c) Wenn A und B ungleich sind, dann fahre fort mit Schritt 1, wenn sie gleich sind, dann beende den Algorithmus: Diese Zahl ist der größte gemeinsame Teiler.

```
A := EINGABE_WERT1;
B := EINGABE_WERT2;

WHILE A != B DO
 BEGIN
  IF A < B THEN
   BEGIN
    T := A;
    A := B;
    B := T;
   END;
  A := A - B;
 END;

GGT := A;
```

4. Der folgende Algorithmus soll den Mittelwert von drei Zahlen a, b und c berechnen.

```
a := 1;
b := 3;
c := 5;
mittel := 0;
mittel := mittel + a + b + c;
mittel := mittel / 4;
```

   Ist der Algorithmus korrekt? Warum [nicht]? Wie müsste er ggf. geändert werden?

   *Da es sich um **drei** Werte handelt (*`mittel` *selbst zählt nicht mit), von denen der Mittelwert berechnet werden soll, muss die Summe der Werte* `a`, `b` *und* `c` *durch 3 geteilt werden, nicht durch 4. D.h. der Algorithmus ist zwar **syntaktisch korrekt**, aber er befolgt nicht die Anweisungen der umgangssprachlichen Aufgabenbeschreibung, und ist somit **semantisch inkorrekt**. In der letzten Anweisung muss durch 3 statt durch 4 geteilt werden, damit der Algorithmus korrekt arbeitet und das gewünschte Ergebnis liefert.*

5. Schreiben Sie einen Algorithmus, der einen Zahlenwert (a) n-mal zu einer Variable sum, die zu Beginn den Wert 0 hat, hinzuaddiert. Welcher Komplexitätsklasse gehört der Algorithmus an?

```
a := KONSTANTE;
n := EINGABEWERT;
sum := 0;

FOR i := 1 TO n DO
 sum := sum + a;
```

   Die Komplexitätsklasse des Algorithmus ist, da die Anzahl der auszuführenden Aweisungen (wegen der `FOR`*-Schleife) linear von der Variablen n abhängt, $O(N)$.*

6. Verbessern Sie die Effizienz des Algorithmus aus der vorigen Aufgabe, indem Sie den Algorithmus syntaktisch so umformulieren, dass das Ergebnis bei gegebenem a und n zwar immer noch das gleiche ist, aber eine niedrigere Komplexitätsklasse erreicht wird.

   Mathematisch ist

$$\sum_{i=1}^{n} a \qquad = \qquad n \cdot a$$

   Daraus vereinfacht sich der vorige Algorithmus zu

```
a := KONSTANTE;
n := EINGABEWERT;

sum := a * n;
```

   mit einer Komplexitätsklasse von $O(1)$, da die Anzahl der Anweisungen nicht mehr von irgendeiner Variablen abhängig ist.