Halteproblem und Church'sche These

Aufgabe 1 (Halteproblem)

Das Halteproblem (Alan Turing) lautet:

> *Es gibt kein Programm, welches feststellt, ob ein beliebiges Programm ein Ergebnis erzielt oder unendlich lange ausgeführt wird.*

Erklären Sie anhand des folgenden Beispiels, warum das Halteproblem nicht lösbar ist.

Sei p ein Programm und x eine Zeichenfolge als Eingabe für p.

Frage: Gibt es eine *boolesche Funktion*, die entscheidet, ob p angewandt auf x hält?

Bei positiver Antwort hätte man ein sehr mächtiges Programm, welches entscheidet, ob ein beliebiges Programm p bei Eingabe einer beliebigen Zeichenkette x hält oder nicht hält.

Da das Programm p selbst ein Text ist, können wir die gesuchte boolesche Funktion wie folgt schreiben.

```
function haelt (p, x : TEXT) : boolean;
begin
  if  <p terminiert bei x>
    then    haelt := TRUE
    else    haelt := FALSE
end;
```

Untersuchen Sie nun folgendes Programm *seltsam*, wenn für x das Programm p = *seltsam* selbst eingegeben wird. Erklären Sie den Widerspruch, der durch diese durchaus legitime Eingabe erzeugt wird.

```
program seltsam;
function haelt …;
begin
  lies(p);
  while  haelt(p, p) do;
  writeln(‚fertig’)
end.
```

Aufgabe 2 (Church'sche These)

Erklären und diskutieren Sie die Church'sche These.

Aufgabe 3 (asymptoische Laufzeitanalysen)

Betrachten Sie die drei Algorithmen *Methode-1*, *Methode-2* und *Methode-3* für das Problem c $\in$ S. Welche asymptotische Laufzeit habe diese drei Algorithmen? Analysieren Sie den Pseudocode und schreiben Sie die Laufzeit in O-Notation auf.

Algorithmus 1:

```
Output:  true, falls c ∈ S, sonst false
Methode-1: contains
    var b : bool;
    b := false;
    for  i := 1 to n do
        if  S[i] = c  then  b := true  endif;
    endfor;
    return  b;
    end  contains.
```

Algorithmus 2:

```
Output:  true, falls c ∈ S, sonst false
Methode-2: contains
i := 1;
while S[i]  ≠ c  and  i ≤ n  do  i := i+1;  { stop, wenn c gefunden }
if  i ≤ n     then return true
              else return  false
```

Algorithmus 3:

Input: S: aufsteigend sortiertes Array von Integerzahlen
low, high: unterer und oberer Indes des zu Bereichs von S
c: integer

Output: true: falls c im Bereich S[low] ..S[high] vorkommt
false: sonst

Methode 3: binäre Suche

Aufgabe 4

Warum gilt folgendes? $\log_3 n = O(\log n)$

$$\log_{10} n = O(\log n)$$

Aufgabe 5 (Logarithmieren)

Gegeben ist die Polynomfunktion $y = bx^c$

Welches Ergebnis erhält man durch Logarithmieren dieser Funktion?