Universität Potsdam
Institut für Informatik

GdP-Übung

Aufgaben für die Präsenzübung, Blatt 5

1 Effizienz von Algorithmen

1.1 O-Notation

1. Betrachten Sie die Funktion $f(n) = 255n^3 - 19n^2 + 0,003n - 1000$. Zeigen Sie
   (a) $f(n) \in O(n^3)$,
   (b) $f(n) \in \Omega(n^3)$.
2. Widerlegen Sie: Es gilt $n^2 \in O(n)$.
3. Geben Sie für jede der Funktionen auf der linken Seite an, in welchen der rechts genannten Funktionsklassen diese enthalten sind (Mehrfachzuordnungen sind möglich). Ein formaler Beweis ist nicht vonnöten.

$3n^2 + 30n + 300 = f(n)$	$\Omega(n^n)$
$4^n = g(n)$	$O(n)$
$5 + 0,001n^3 + 0,0025n = h(n)$	$\Theta(n^2)$
$n^2 + 3n - 3 = j(n)$	$O(3^n)$
	$O(n^2)$

1.2 Auswirkungen der Art der Datenrepräsentation

Je nach der Repräsentation eines Graphen (als Adjazenzliste oder -matrix) ergeben sich unterschiedliche Zeitkomplexitäten für die Ausführung bestimmter Algorithmen in Abhängigkeit von der Anzahl der Knoten $|V| = n$. Untersuchen Sie die Zeitkomplexität der Algorithmen zur Bestimmung des Ausgangsgrades sowohl für den Fall, dass der Graph durch seine Adjazenzlisten gegeben ist, als auch für den Fall seiner Repräsentation als Adjazenzmatrix. Betrachten Sie insbesondere auch den best und average case!

```
1 def agrad_liste(adj,j):
2     agrad = 0
3     for i in adj[j]:
4         agrad = agrad + 1
5     return agrad
```

```
1 def agrad_matrix(A,j):
2     agrad = 0
3     for i in range(len(A)):
4         agrad = agrad + A[j,i]
5     return agrad
```

1.3 Vergleich von Algorithmen

1. Bestimmen Sie die Zeitkomplexität im schlechtesten Fall für die Algorithmen `ggt_1` und `ggt_2` bezüglich $n = \max\{x, y\}$ als Größe der Eingabe.
2. Bestimmen Sie für `ggt_1` außerdem die Zeitkomplexität bezüglich der Bitlänge $n = \lfloor \log_2 x \rfloor + 1 + \lfloor \log_2 y \rfloor + 1$ als Größe der Eingabe.

```
1  def ggt_1(x,y):
2      while x != y:
3          if x > y:
4              x = x - y
5          else:
6              y = y - x
7      return x
```

```
1  def teilerlist(x):
2      L = []
3      for k in range(1,x+1):
4          if x % k == 0:
5              L.append(k)
6      return L
7
8  def ggt_2(x,y):
9      if x > y:
10         tmp = x; x = y; y = tmp
11     L1 = teilerlist(x)
12     L2 = teilerlist(y)
13     i = len(L1) - 1
14     while i >= 0:
15         if L1[i] in L2:
16             return L1[i]
17         i = i - 1
```

1.4 Aus Analyse folgt Verbesserung

Schauen Sie sich das Verhalten des folgenden Algorithmus an.
Finden Sie heraus, welche Funktion der Algorithmus berechnet, wenn die Eingabe eine Liste von Zahlen ist, und entwickeln Sie einen Algorithmus, der genau das gleiche Ergebnis liefert, jedoch eine bessere Zeitkomplexität aufweist!

```
1  def function(xs):
2      a = xs[0]
3      for i in xs:
4          b = True
5          for j in xs:
6              if i > j:
7                  b = False
8          if b:
9              a = i
10     return a
```