TU Wien
Institut für Logic and Computation
Forschungsbereich Algorithms and Complexity

186.866 Algorithmen und Datenstrukturen VU

Übungsblatt 1

Deadline für dieses Übungsblatt ist **Montag, 21.03.2022, 20:00 Uhr**. Um Aufgaben für diese Übung anerkannt zu bekommen, gehen Sie folgendermaßen vor:

1. Öffnen Sie den TUWEL-Kurs der Lehrveranstaltung *186.866 Algorithmen und Datenstrukturen (VU 5.5)* und navigieren Sie zum Abschnitt *Übungsblätter*.

2. Teilen Sie uns mit, welche Aufgaben Sie gelöst haben **und** welche gelösten Aufgaben Sie gegebenenfalls in der Übungseinheit präsentieren können. Gehen Sie dabei folgendermaßen vor:

   - Laden Sie Ihre Lösungen in einem einzigen PDF-Dokument in TUWEL hoch.
     Link *Hochladen Lösungen Übungsblatt 1*
     Button *Abgabe hinzufügen*
     PDF-Datei mit Lösungen hochladen und *Änderungen sichern*.
   - Kreuzen Sie an, welche Aufgaben Sie gegebenenfalls in der Übung präsentieren können. Die Lösungen der angekreuzten Aufgaben müssen im hochgeladenen PDF enthalten sein.
     Link *Ankreuzen Übungsblatt 1*
     Button *Abgabe bearbeiten*
     Bearbeitete Aufgaben anhaken und *Änderungen speichern*.

Bitte beachten Sie:

- Bis zur Deadline können Sie sowohl Ihr hochgeladenes PDF, als auch Ihre angekreuzten Aufgaben beliebig oft verändern. Nach der Deadline ist keine Veränderung mehr möglich. Es werden ausnahmslos keine Nachabgabeversuche (z.B. per E-Mail) akzeptiert.

- Sie können Ihre Lösungen entweder direkt in einem Textverarbeitungsprogramm erstellen, oder aber auch gut leserliche Scans bzw. Fotos von handschriftlichen Ausarbeitungen hochladen (beachten Sie die maximale Dateigröße).

- Geben Sie Ihren Namen, Ihre Matrikelnummer und Ihre E-Mail-Adresse in den Ausarbeitungen an.

- Beachten Sie die Richtlinien für das An- und Aberkennen von Aufgaben (Details finden Sie in den Folien der Vorbesprechung).

Aufgabe 1. Gegeben ist folgendes Stable Matching Problem mit zwei Tabellen von Präferenzlisten.

	1.	2.	3.	4.
A	W	Z	X	Y
B	Z	W	X	Y
C	Z	X	Y	W
D	X	Z	Y	W

	1.	2.	3.	4.
W	B	D	A	C
X	C	A	B	D
Y	B	A	D	C
Z	A	D	B	C

Finden Sie ein Stable Matching mithilfe des Gale-Shapley-Algorithmus aus der Vorlesung. Lässt der Algorithmus offen, welches Element als nächstes betrachtet werden soll, dann gehen Sie in alphabetischer Reihenfolge vor. Geben Sie alle Zwischenschritte an.

Aufgabe 2. Ordnen Sie folgende Funktionen nach Dominanz, beginnend mit der asymptotisch am schwächsten wachsenden. Es genügt die Funktionen zu reihen, ein Beweis der Gültigkeit der Relationen ist nicht erforderlich.

$$\frac{n!}{12}, \quad \log^3(n), \quad (7/6)^n, \quad n^4 \cdot \log(n^3), \quad \frac{n}{4}^n, \quad (0{,}84)^{2n}, \quad n^5 + \ln^2(n), \quad \frac{n^7 + 2n^5 - n^3}{4n^2 + n^3}$$

Aufgabe 3. Für ein Problem mit Eingabegröße n gibt es einen etablierten Algorithmus A mit Worst-Case Laufzeit

$$f(n) = n^2 \cdot (n/\ln(n) + 4) + 10.$$

Im Folgenden wollen wir Worst-Case Laufzeiten alternativer Algorithmen betrachten und mithilfe der Θ, O, Ω Notationen mit dem etablierten Algorithmus vergleichen.

Als erstes wird Algorithmus N1 mit der Laufzeit

$$g_1(n) = n^3/4 + n + 10$$

vorgeschlagen. Ein alternativer Ansatz resultiert in einem Algorithmus N2 mit der Laufzeit

$$g_2(n) = \frac{(n^2 + n + 1) \cdot (1 + 4n)}{2 \cdot \sqrt[5]{n^2 + 3}}.$$

Der dritte Algorithmus N3 unterscheidet, ob $n \leq 110$ ist, in diesem Fall wird Algorithmus N1 ausgeführt, oder $n > 110$ ist, in diesem Fall wird Algorithmus A ausgeführt. Wir erhalten also:

$$g_3(n) = \begin{cases} n^3/4 + n + 10 & \text{falls } n \leq 110 \\ n^2 \cdot (n/\ln(n) + 4) + 10 & \text{sonst} \end{cases}$$

Der vierte Algorithmus N4 unterscheidet, ob n gerade ist, in diesem Fall wird Algorithmus N1 ausgeführt, oder n ungerade ist, in diesem Fall wird Algorithmus N2 ausgeführt. Wir erhalten also:

$$g_4(n) = \begin{cases} n^3/4 + n + 10 & \text{falls } n \text{ gerade} \\ \dfrac{(n^2 + n + 1) \cdot (1 + 4n)}{2 \cdot \sqrt[5]{n^2 + 3}} & \text{sonst} \end{cases}$$

Kreuzen Sie in der folgenden Tabelle die zutreffenden Felder an und begründen Sie Ihre Antworten (formaler Beweis ist nicht notwendig). Nutzen Sie die Tabelle dann, um die Algorithmen zu vergleichen.

$f(n)$ ist in	$\Theta(.)$	$O(.)$	$\Omega(.)$	keines
$g_1(n)$				
$g_2(n)$				
$g_3(n)$				
$g_4(n)$				

Hinweis: Setzen Sie statt dem Punkt die entsprechende Funktion (g_1, g_2, g_3 bzw. g_4) ein. Beispielsweise ist die Zelle links oben als „$f(n)$ ist in $\Theta(g_1(n))$“ zu lesen.

Aufgabe 4.

(a) Beweisen oder widerlegen Sie, dass für die folgende Funktion $f(n)$ die Beziehung $f(n) = O(n^3 \log_2 n)$ gilt:

$$f(n) = \begin{cases} n^2 + \sqrt[3]{n^9} \cdot \log_4(n^5), & \text{falls } n \text{ gerade} \\ (n^2 + 5n) + \frac{n^4}{2^n} & \text{falls } n \text{ ungerade} \end{cases}$$

Hinweis: Bedenken Sie, dass für einen Beweis gegebenenfalls auch geeignete Werte für die Konstanten c und n_0 angegeben werden müssen.

(b) Beweisen oder widerlegen Sie, dass wenn $f_1(n) = \Theta(g_1(n))$ und $f_2(n) = \Theta(g_2(n))$ dann gilt auch $f_1(n) \cdot f_2(n) = \Theta(g_1(n) \cdot g_2(n))$.

Anmerkung: Wir nehmen an, dass alle Funktionen die nicht-negativen ganzen Zahlen als Definitions- und Wertebereich haben.

Aufgabe 5. Bestimmen Sie die Laufzeiten der unten angegebenen Algorithmen in Abhängigkeit von n in Θ-Notation. Verwenden Sie hierfür möglichst einfache Terme. Überlegen Sie bei jedem Algorithmus, ob eine Unterscheidung in Best-Case und Worst-Case Laufzeit notwendig ist und geben Sie gegebenenfalls beide Laufzeiten an.

(a)
```
a ← 0
for i = 1, ..., 4
    a ← a + 1
return a * a
```

(b)
```
b ← 1,5
for i = 1, ..., n − 1
    if A[i − 1] ≠ A[i] then
        for j = 1, ..., ⌊n/2⌋
            b ← b + i * j
return b
```

(c)
```
c ← 5
for i = 1, ..., ⌈√n⌉
    c ← c + 1
for i = 1, ..., n
    j ← n − i
    while j ≤ n
        c ← c * i
        j ← j + 1
return c
```

(d)
```
d ← 1,5
i ← n
while i ≥ 2
    d ← d + 1
    i ← i/2
return d
```

Aufgabe 6. Wir wollen die Laufzeit für den folgenden Algorithmus in Abhängigkeit von n in Θ-Notation ermitteln.

```
j ← 1
w ← 24
while j < n
    j ← 3 * j
    w ← w + n
ℓ ← 1
for i = 1, ..., w
    j ← w
    do
        j ← j + 3
        ℓ ← ℓ + √i
    while j < 3 * w
    ℓ ← ⌊ℓ⌋
return ℓ
```

(a) Bestimmen Sie die Laufzeit und den Rückgabewert für den folgenden Teil des Algorithmus in Abhängigkeit von n in Θ-Notation.

```
j ← 1
w ← 24
while j < n
    j ← 3 * j
    w ← w + n
return w
```

(b) Bestimmen Sie die Laufzeit für den folgenden Teil des Algorithmus in Abhängigkeit von w in Θ-Notation.

```
ℓ ← 1
for i = 1, ..., w
    j ← w
    do
        j ← j + 3
        ℓ ← ℓ + √i
    while j < 3 * w
    ℓ ← ⌊ℓ⌋
return ℓ
```

(c) Bestimmen Sie die Laufzeit für den gesamten Algorithmus in Abhängigkeit von n in Θ-Notation, indem Sie die bisherigen Ergebnisse kombinieren.

Verwenden Sie für alle Ergebnisse möglichst einfache Terme.
