Algorithmen und Programmierung 3, WS 2019/2020 — 1. Übungsblatt

Abgabe bis Freitag, 25. Oktober 2019, 12:00 Uhr, in die Fächer der Tutoren
Abzugeben sind diesmal die Aufgaben 1, 2, und 6. Null-Punkte-Aufgaben sind zur freiwilligen Vertiefung gedacht und werden nicht abgegeben.

---

1. (i) Vorübung: Wachstum von Laufzeiten, 4 Punkte
   Sortieren Sie die folgenden Laufzeiten aufsteigend: Wenn $f(n) = O(g(n))$ ist, aber nicht $g(n) = O(f(n))$, dann soll $f(n)$ vor $g(n)$ kommen.

   (a) n^3 (b) $\log_2 n$ (c) $1{,}8^n$ (d) n (e) 3^n (f) $\sqrt{n}$ (g) $n(\log_2 n)^2$ (h) n^2

   (ii) Beschleunigung der Hardware, 4 Punkte
   Wenn ein Programm eine Laufzeit $f(n)$ aus Aufgabe (i) hat und die Computer in zehn Jahren um den Faktor 1000 schneller werden, dann kann man Probleme *welcher Größe* in derselben Zeit lösen, in der man heute Probleme der Größe (1) $n = 20$, (2) $n = 1000$ lösen kann?

2. (2 Punkte) Die Computer werden immer schneller, die Speicherelemente immer billiger. Wird der Entwurf von effizienten Algorithmen angesichts dieser Entwicklung in Zukunft an Bedeutung verlieren? Bei welchen Computeranwendungen werden Effizienzfragen eine größere/kleinere Rolle spielen?

   Diskutieren Sie diese Fragen. (mindestens 5–10 Zeilen)

3. Korrektheit von Programmen, 0 Punkte

   Das folgende Programmstück ist ein Versuch, Sortieren durch Einfügen zu implementieren.

```
    void insertionSort(float[] a)
    { float x;
      for (int i = 0; i<a.length; i++)
      { † x = a[i];
        // Sortiere x zwischen a[0..i-1] ein
        // Bestimme zunächst die richtige Position j:
(*)     int j=i-1; † while (j>=0 && a[j-1]>x) j-- † ;
        // Nun verschiebe einen Teil des Feldes und füge x an dieser Stelle ein:
      † for (int k = i-1; k>=j; k--) a[k+1]=a[k];
        a[j]=x; †
      }
    }
```

   Stellen Sie dieses Programm richtig und fügen Sie an den durch † bezeichneten Stellen in Ihrem Programm Schleifeninvarianten in der Form vom Zusicherungen (assertions) ein, aus denen die Korrektheit des Programmes hervorgeht. (Ein formaler Beweis ist nicht erforderlich, aber die Zusicherungen müssen aussagekräftig sein und vor allem zutreffen.)

4. (0 Punkte) Kann man Sortieren durch Einfügen schneller machen, indem man die Einfügestelle `j` schneller findet als oben in Zeile (∗), zum Beispiel durch binäre Suche? Wie ist es im besten und im schlechtesten Fall?

5. (0 Punkte) Was passiert bei (i) *Sortieren durch Einfügen*, (ii) *Sortieren durch Auswahl*, wenn die gegebene Folge bereits (a) aufsteigend oder (b) absteigend sortiert ist? (c) Was passiert, wenn alle Elemente gleich sind? Ist es möglich, dass man bloß $O(n)$ Zeit benötigt?

6. Programmierexperiment zum Sortieren, 10 Punkte

   Schreiben Sie ein Programm in Java zum Sortieren von Zahlen, und testen Sie es mit 20 Millionen zufällig erzeugten Gleitkommazahlen einfacher Genauigkeit (`float`).

   - Programmieren Sie das Verfahren Quicksort. Wenn die Länge der zu sortierenden Teilfolge in der Rekursion unter eine Schranke b sinkt, dann soll aber auf eine einfacheres Sortierverfahren ihrer Wahl umgeschaltet werden (z. B. Sortieren durch Einfügen, Sortieren durch Auswahl, Bubblesort).
   - Das Pivotelement, das bei Quicksort zum Zerlegen der Folge verwendet wird, soll in jeder Teilfolge zufällig ausgewählt werden.
   - Messen Sie die Laufzeit. Zählen Sie auch die *Anzahl der Vergleiche* zwischen den Eingabezahlen mit und geben Sie sie aus.
   - Experimentieren Sie mit mindestens drei verschiedenen Werten von b, und versuchen Sie experimentell, den besten Wert zu finden.
   - Die Eingabewerte sollen gleichverteilt im Intervall $[0, 1]$ erzeugt werden.
   - Machen Sie jeweils (mindestens) zehn Durchläufe mit verschiedenen Datensätzen, und dokumentieren Sie die Ergebnisse. Geben Sie auch die Durchschnittswerte Ihrer Programmläufe an.

   Zusatzaufgabe (0 Punkte): Bestimmen Sie jeweils die Anzahl der mehrfach vorkommenden Werte: Wieviele Zahlen kommen mindestens zweimal in der Eingabe vor?

   Anleitung zur Abgabe der Programmieraufgabe: Laden Sie die dokumentierten Quelltexte (zum Beispiel als `zip`-komprimierten Ordner) auf der Whiteboard-Seite der Veranstaltung[1] bis zum Abgabetermin (Freitag, 25. Oktober 2019, 12:00 Uhr) hoch. Fügen Sie eventuell eine `README`-Datei hinzu, wenn der Aufruf des Programmes und die Benützung nicht selbsterklärend ist.

   Geben Sie zusätzlich den ausgedruckten Quellcode und die Ergebnisse der Testläufe auf Papier ab.

7. Vergleich von asymptotischen Laufzeiten, 0 Punkte

   Welche der beiden folgenden Laufzeiten $f(n)$ und $g(n)$ ist für große Werte von n schneller, und welche für kleine n? Bei welchem Wert von n ändert sich die Antwort?

   (a) $f(n) = 10n(\log_2 n)^2$, $g(n) = 2n^{3/2}$
   (b) $f(n) = 5 \cdot 2^n$, $g(n) = 100n^2 \log_2 n$

8. Wiederholung der O-Notation, 0 Punkte

   (a) Beweisen Sie: Wenn $f(n) = O(g(n))$ ist, dann ist $f(n) + g(n) = O(g(n))$.
   (b) Beweisen Sie: $\max\{f(n), g(n)\} = \Theta(f(n) + g(n))$ für $f(n), g(n) > 0$.
   (c) Welche der folgenden Aussagen sind richtig (Begründen Sie Ihre Antworten):
   $n\sqrt{n} = O(n(\log n)^2)$; $n(\log n)^2 = O(n\sqrt{n})$; $n \log n \cdot (\log\log n) = O(n^2)$;
   $n^2 = O(n \log n \cdot (\log\log n))$; $n^2 \cdot 2^n = O(2^{n+2})$; $n^2 \cdot 2^n = O(3^n)$; $3^n = O(n^2 \cdot 2^n)$;

[1] `https://mycampus.imp.fu-berlin.de/x/sL9Pfe`

1. (i) Vorübung: Wachstum von Laufzeiten, 4 Punkte

Sortieren Sie die folgenden Laufzeiten aufsteigend: Wenn $f(n) = O(g(n))$ ist, aber nicht $g(n) = O(f(n))$, dann soll $f(n)$ vor $g(n)$ kommen.

(a) n^3 (b) $\log_2 n$ (c) $1{,}8^n$ (d) n (e) 3^n (f) $\sqrt{n}$ (g) $n(\log_2 n)^2$ (h) n^2

$$\log_2 n \overset{1}{<} \sqrt{n} \overset{2}{<} n \overset{3}{<} n(\log_2 n)^2 \overset{4}{<} n^2 \overset{5}{<} n^3 \overset{6}{<} 1{,}8^n \overset{7}{<} 3^n$$

1) zz. $\log_2 n \overset{!}{\leq} \sqrt{n}$

I.A. $n=4$:

$\log_2 4 \overset{!}{\leq} \sqrt{4} \quad / \cdot 2$

$4 \leq 2^2$ ✓

I.V. $\log_2 n \leq \sqrt{n}$ gelt

$\Leftrightarrow (\log_2 n)^2 \leq n$ ↑

I.S. $\log_2 (n+1) \leq \sqrt{n+1} \quad / ^2$

$(\log_2 (n+1))^2 \overset{\text{I.V.}}{\leq} (\log_2 n)^2 + 1 \quad / \sqrt{\ }$

$\log_2 (n+1) \leq \log_2 (n) + 1 \quad / 2^x$

$n+1 \leq n + 2^1$

$n+1 \leq n+2$

2) zz. $\sqrt{n} \leq n$

$\limsup_{n\to\infty} \left| \frac{\sqrt{n}}{n} \right| \quad / ^2$

$\limsup_{n\to\infty} \left| \frac{n}{n^2} \right|$

$\limsup_{n\to\infty} \left| \frac{1}{n} \right| < \infty$

3) zz. $n \leq n(\log n)$

$\limsup_{n\to\infty} \left| \frac{n}{n(\log n)} \right|$

$\limsup_{n\to\infty} \left| \frac{1}{\log n} \right| < \infty$

4) zz. $n(\log n)^2 \leq n^2$

$\limsup_{n\to\infty} \frac{n(\log n)^2}{n^2}$

$= \limsup_{n\to\infty} \frac{(\log n)^2}{n} \quad / \sqrt{\ }$

$= \limsup_{n\to\infty} \frac{\log n}{\sqrt{n}}$ im 1. bewiesen

5) zz. $n^2 \leq n^3$

$\limsup_{n\to\infty} \left| \frac{n^2}{n^3} \right|$

$\limsup_{n\to\infty} \left| \frac{1}{n} \right| < \infty$

6) zz. $n^3 < 1{,}8^n$

I.A. $n=14$

$14^3 < 1{,}8^{14}$

$2744 < 3748$

I.V. $n^3 < 1{,}8^n$

I.S. zz. $(n+1)^3 < 1{,}8^{n+1}$

$(n+1)^3 < 1{,}8^n \cdot 1{,}8$

$(n+1)^3 \overset{\text{I.V.}}{<} n^3 \cdot 1{,}8$

2. Induktion:

I.A. $n=14 \quad 15^3 < 14^3 \cdot 1{,}8$ ✓

I.V. $(n+1)^3 < n^3 \cdot 1{,}8$

I.S. zz. $(n+2)^3 < (n+1)^3 \cdot 1{,}8$

$n^3 + 6n + 6n^2 + 8 < (n^3 + 3n + 3n^2 + 1) \cdot 1{,}8$

$n^3 + 6n + 6n^2 + 8 \overset{\text{I.V.}}{<} (3n + 3n^2 + 1) \cdot 1{,}8 + (n+1)^3$

$n^3 + 6n + 6n^2 + 8 < (3n + 3n^2 + 1) \cdot 1{,}8 + n^3 + 3n + 3n^2 + 1 \quad / -(n^3 + 3n + 3n^2 + 1)$

$3n + 3n^2 + 7 < (3n + 3n^2 + 1) \cdot 1{,}8$

$3n + 3n^2 + 7 < 5{,}4n + 5{,}4n^2 + 1{,}8 \quad / -(3n + 3n^2 + 1{,}8)$

$5{,}2 < 2{,}4n + 2{,}4n^2$

7) zz. $1{,}8^n \leq 3^n$

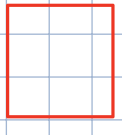
$\limsup_{n\to\infty} \frac{1{,}8^n}{3^n} \quad / \sqrt[n]{\ }$

$\limsup_{n\to\infty} \frac{1{,}8}{3} < \infty$

(ii) Beschleunigung der Hardware, 4 Punkte

Wenn ein Programm eine Laufzeit $f(n)$ aus Aufgabe (i) hat und die Computer in zehn Jahren um den Faktor 1000 schneller werden, dann kann man Probleme *welcher Größe* in derselben Zeit lösen, in der man heute Probleme der Größe (1) $n = 20$, (2) $n = 1000$ lösen kann?

$O(n)$	$n=20$	$n=1000$
$\log n$	$2^{20.000}$	$2^{1.000.000}$
$\sqrt{n}$	20.000^2	$1.000.000^2$
n	20.000	$1.000.000$
$n(\log n)^2$	$2^{\sqrt{(20.000)}}$	$2^{\sqrt{(1.000.000)}}$
n^2	$\sqrt[2]{20.000}$	$\sqrt[2]{1.000.000}$
n^3	$\sqrt[3]{20.000}$	$\sqrt[3]{1.000.000}$
$1,8^n$	$\log_{1,8} 20.000$	$\log_{1,8} 1.000.000$
3^n	$\log_3 20.000$	$\log_3 1.000.000$



Inh.

O-Notation

$$\exists k>0,\ \exists x_0>0,\ \forall x>x_0 : |f(x)| \le k \cdot |g(x)|$$

Bsp. $a \in \mathbb{R} \cup \{-\infty, \infty\}$ $x \in \mathbb{R}$

$$\limsup \left|\frac{f(x)}{g(x)}\right| < \infty \Rightarrow f \in O(g)$$

$$x(\log 30),\ \frac{1}{x},\ \sqrt[3]{x},\ \frac{x}{3}$$

Logarithmusregeln

$$\log(ab) = \log a + \log b$$
$$\log\left(\frac{a}{b}\right) = \log a - \log b$$
$$\log(a^b) = b \cdot \log a$$
$$\log(a^{\frac{1}{b}}) = \log(\sqrt[b]{a}) = \frac{1}{b} \cdot \log a$$
$$a^{\log_a b} = b$$
$$\log_a b = \frac{\log_c b}{\log_c a}$$

```
Länge der Liste: 20000000
Durchläufe: 10
Schranke: 10

Durchlauf 1.
Vergleiche: 675034901
Dauer: 3s 217ml 241ms 892ns

Durchlauf 2.
Vergleiche: 705149880
Dauer: 3s 272ml 965ms 653ns

Durchlauf 3.
Vergleiche: 692527929
Dauer: 3s 323ml 3ms 593ns

Durchlauf 4.
Vergleiche: 684822940
Dauer: 3s 280ml 839ms 85ns

Durchlauf 5.
Vergleiche: 691415956
Dauer: 3s 283ml 656ms 684ns

Durchlauf 6.
Vergleiche: 695236047
Dauer: 3s 249ml 994ms 357ns

Durchlauf 7.
Vergleiche: 721799273
Dauer: 3s 338ml 947ms 978ns

Durchlauf 8.
Vergleiche: 704246500
Dauer: 3s 246ml 749ms 677ns

Durchlauf 9.
Vergleiche: 692111372
Dauer: 3s 268ml 741ms 300ns

Durchlauf 10.
Vergleiche: 690691831
Dauer: 3s 242ml 162ms 32ns

Durchschnittliche Vergleiche: 695303662
Durchschnittliche Dauer: 3s 272ml 430ms 32ns
```

```
Länge der Liste: 20000000
Durchläufe: 10
Schranke: 100

Durchlauf 1.
Vergleiche: 1816755830
Dauer: 5s 451ml 205ms 152ns

Durchlauf 2.
Vergleiche: 1811189155
Dauer: 5s 481ml 276ms 141ns

Durchlauf 3.
Vergleiche: 1802212251
Dauer: 5s 624ml 628ms 561ns

Durchlauf 4.
Vergleiche: 1805580658
Dauer: 5s 589ml 909ms 843ns

Durchlauf 5.
Vergleiche: 1821312056
Dauer: 5s 484ml 680ms 664ns

Durchlauf 6.
Vergleiche: 1820288620
Dauer: 5s 501ml 390ms 675ns

Durchlauf 7.
Vergleiche: 1814846085
Dauer: 5s 513ml 397ms 609ns

Durchlauf 8.
Vergleiche: 1810830613
Dauer: 5s 502ml 743ms 514ns

Durchlauf 9.
Vergleiche: 1841329183
Dauer: 5s 513ml 900ms 315ns

Durchlauf 10.
Vergleiche: 1804089175
Dauer: 5s 516ml 220ms 338ns

Durchschnittliche Vergleiche: 1814843362
Durchschnittliche Dauer: 5s 517ml 935ms 338ns
```

```
Länge der Liste: 20000000
Durchläufe: 10
Schranke: 1000

Durchlauf 1.
Vergleiche: 13725791496
Dauer: 23s 961ml 616ms 603ns

Durchlauf 2.
Vergleiche: 13726719495
Dauer: 24s 58ml 734ms 310ns

Durchlauf 3.
Vergleiche: 13770251862
Dauer: 23s 944ml 595ms 94ns

Durchlauf 4.
Vergleiche: 13736766307
Dauer: 23s 804ml 143ms 29ns

Durchlauf 5.
Vergleiche: 13734423020
Dauer: 23s 831ml 438ms 270ns

Durchlauf 6.
Vergleiche: 13681323810
Dauer: 23s 782ml 524ms 72ns

Durchlauf 7.
Vergleiche: 13747902436
Dauer: 23s 960ml 337ms 505ns

Durchlauf 8.
Vergleiche: 13694374862
Dauer: 23s 801ml 986ms 121ns

Durchlauf 9.
Vergleiche: 13740813582
Dauer: 23s 891ml 380ms 559ns

Durchlauf 10.
Vergleiche: 13756624288
Dauer: 23s 875ml 369ms 787ns

Durchschnittliche Vergleiche: 13731499115
Durchschnittliche Dauer: 23s 891ml 212ms 787ns
```

$\Rightarrow$ je niedriger die Schranke desto besser, da bubblesort $C(n) = n^2$

1)

$$\lim_{n\to\infty} \frac{\log_2 n}{\sqrt{n}}$$

$$\lim_{n\to\infty} \frac{\log_2 n}{n^{\frac{1}{2}}} \quad / \text{ ableiten}$$

$$\lim_{n\to\infty} \frac{n \cdot \ln(2)}{\frac{1}{2} n^{-\frac{1}{2}}}$$

$$\lim_{n\to\infty} \frac{n \cdot \ln(2)}{\frac{1}{2\sqrt{n}}}$$

$$\lim_{n\to\infty} \frac{2\sqrt{n}}{n \cdot \ln(2)} \quad / \text{ ableiten}$$

$$\lim_{n\to\infty} \frac{1}{\sqrt{n} \cdot \ln(2)}$$