Max Wisniewski , Alexander Steen

Tutor: Lena Schlipf

Aufgabe 1

In dieser Aufgabe von wir zwei $n \times n$ Matrizen schneller als mit der schulmethode Berechnen.

(a) Erklären Sie in einem Satz, wie viele Additionen und Multiplikationen asymptotisch nötig sind, um zwei $n \times n$ Matrizen mit dem Schulalgorithmus zu multiplizieren.

Bei der Schulmethode rechnet man einfach Zeilen mal Spalten, was bei $n \cdot n$ Einträgen pro Eintrag n Multiplikationen und $n-1$ Additionen sind.

$\Rightarrow n^3$ Multiplikationen und $n^3 - n^2$ Additionen.

(b) Zeigen Sie, warum die in der Übung gegebene Zerlegung von 2×2 Matrizen auf beliebige Matrizen erweiterbar ist.
Die genaue Aufteilung der Matrizen für die Multiplikation kann dem Übungszettel entnommen werden.

Wir betrachten die Koeffizienten der Matrix nun als Matrizen und nicht mehr als Zahlen. Wir zeigen, dass die gewählten Formen die richtige Form erzeugt (bekannt aus der Schulmethode):

$$\begin{aligned} r &= ae + bf \\ t &= ce + df \\ s &= ag + bh \\ u &= cg + dh \end{aligned}$$

Nun setzten wir für die einzelnen Komponenten r, t, s, u die Formeln aus dem Algorithmus ein:

$$\begin{aligned} r &= (a+d)(e+h) + d(f-e) - (a+b)h + (b-d)(f+h) \\ &= ae + ah + de + dh + df - de - ah - bh \\ &\quad + bf + bh - df - dh \\ &= ae + bf \end{aligned}$$

$$\begin{aligned} s &= a(g-h) + (a+b)h \\ &= ag - ah + ah + bh \\ &= ag + bh \end{aligned}$$

$$\begin{aligned} t &= (c+d)e + d(f-e) \\ &= ce + de + df - de \\ &= ce + df \end{aligned}$$

$$\begin{aligned} u &= (a+d)(e+h) + a(g-h) - (c+d)e - (a-c)(e+g) \\ &= ae + ah + de + dh + ag - ah - ce - de - ae \\ &\quad - ag + ce + cg \\ &= dh + cg \end{aligned}$$

Wir sehen, dass in der Matrix nach dem Algorithmus in jedem Schritt wieder die richtige Matrix zusammen gesetzt wird. Im Anker können wir einfach die Schulmethode verwenden um auf die richtigen Werte zu kommen. Nach dieser Überlegung sollte der Algorithmus korrekt funktionieren.

(c) Beschreiben Sie die Laufzeit von Strassens Algorithmus mit einer Rekursionsgleichung und lösen Sie diese.

Für die Analyse nehmen wir das EKM.
Um zwei Matrizen zu Addieren (oder Subtrahieren) müssen wir die Komponeten addieren. Dafür brauchen wir also n^2 Operationen.
Um nun die einzelnen Komponenten zu berechnen braucht man für P_1 bis P_7 10 dieser Additionen/Subtraktionen. Um nach der Rekursion auf r, s, t, u zu kommen benötigen wir nocheinmal 9 Additionen/Subtraktionen. Diese operieren alle auf $\frac{n}{2} \times \frac{n}{2}$ Matrizen.

$$\Rightarrow T(n) = \begin{cases} O(1) & , n = 2 \\ 7T(\frac{n}{2}) + 19n^2 & , \text{sonst} \end{cases}$$

Nun wenden wir darauf das Mastertheorem an:
Mit $a = 7,\ b = 2,\ f(n) = 19n^2$

$$f(n) = 19n^2 \overset{\varepsilon=0.5}{=} O(n^{2.5-0.5}) = O(n^{\log_2 7}), \text{da} \log_2 7 \approx 2.81$$

Damit können wir den ersten Fall des Mastertheorems anwenden:

$$\Rightarrow T(n) = \Theta\left(n^{\log_2 7}\right) \approx \Theta\left(n^{2.81}\right)$$

Aufgabe 2

Geben Sie möglichst gute asymptotische Schranken für die folgenden rekursiv definierte Funktion $T(n)$ an; dabei ist $T(n) \in \Theta(1)$ für $n \leq 2$.

(a) $T(n) = T(\frac{9}{10}n) + n$
Hier gilt $a = 1,\ b = \frac{10}{9},\ f(n) = n$

$$f(n) = n = \Omega(n^{0+\varepsilon}), \quad \text{mit } \varepsilon = 1,$$

bleibt zu zeigen, dass $a \cdot f(\frac{n}{b}) \leq c \cdot f(n)$:

$$1 \cdot f(\frac{9}{10}n) = \frac{9}{10}n \text{ gilt, mit } c = \frac{9}{10}$$
$$\Rightarrow T(n) = \Theta(n)$$

(b) $T(n) = T(n-a) + T(a) + n,\ a \leq 1$
Wie man schnell sieht, kann man hier das Mastertheorem nicht anwenden ($n - a$ kann nicht auf $\frac{n}{b}$ gebracht werden). Wir verlegen uns deshalb aufs einsetzen:

$$\begin{aligned} T(n) &= T(n-a) + T(a) + n \\ &= T(n-2a) + T(a) + n - a + T(a) + n \\ &= T(n-2a) + 2T(a) + 2n - a \\ &= T(n-3a) + 3T(a) + 3n - 3a \\ &= T(n-4a) + 4T(a) + 4n - a \cdot \sum_{i=1}^{3} i \\ &\quad \text{Nach k Schritten} \\ &= T(n-ka) + kT(a) + kn - a \cdot \sum_{i=1}^{k-1} i \end{aligned}$$

Der Anker wird bei
$n - ka = 2 \Leftrightarrow k = \frac{n-2}{a}$ Damit löst sich die Gleichung auf zu:

$$T(n) = \Theta(1) + \tfrac{n-2}{a}T(a) + \tfrac{n-2}{a} \cdot n - a \cdot \tfrac{(\frac{n-2}{a}-1)(\frac{n-2}{a})}{2}$$

$T(a)$ ist konstant, da $a \leq 1$ den Rekursionsanker trifft. Damit erhält man, wenn man alles ausklammert am Schluss:

$$T(n) = \Theta(n^2)$$

(c) $T(n) = T(\sqrt{n}) + 1$
Wieder sieht man hier schnell das es nicht auf die Form des Mastertheorem kommen kann.

$$\begin{aligned} T(n) &= T(\sqrt{n}) + 1 \\ &= T(n^{\frac{1}{2}}) + 1 \\ &= T(n^{\frac{1}{4}}) + 2 \\ &\quad \text{Nach k Schritten} \\ &= T(n^{\frac{1}{2^k}}) + k \end{aligned}$$

Den Anker erreichn wir dabei mit: $n^{\frac{1}{2^k}} = 2 \Leftrightarrow 2^{2^k} = n \Leftrightarrow \log_2(\log_2 n) = k$
Daraus ergibt sich eine Lauzeit von:

$$T(n) = \Theta(\log\log n)$$

(d) $T(n) = 2T(\frac{n}{4}) + \sqrt{n}$
$a = 2,\ b = 4,\ f(n) = \sqrt{n}$ Nun gilt:

$$\sqrt{n} = n^{\frac{1}{2}} = n^{\log_4 2} = \Theta(n^{\log_4 2})$$

Damit können wir das Mastertheorem anwenden.

$$\Rightarrow T(n) = \sqrt{n} \cdot \log n$$

(e) $T(n) = 7T(\frac{n}{3}) + n^2$
Hier kann man das Mastertheorem verwenden, was wir beim ersten mal leider nicht konnten, da wir uns verrechnet haben. Da wir aber auf das selbe Ergebnis kommen, wollten wir das ganze nicht noch einmal ändern.

$$\begin{aligned} T(n) &= 7T\left(\tfrac{n}{3}\right) + n^2 \\ &= 7^2T\left(\tfrac{n}{3^2}\right) + \tfrac{n^2}{(3^1)^2} + n^2 \\ &\quad \text{Nach k Schritten} \\ &= 7^kT\left(\tfrac{n}{3^k}\right) + n^2 \cdot \sum_{i=0}^{k-1} \tfrac{1}{3^{2\cdot i}} \end{aligned}$$

Den Anker erreichen wir bei:

$$\frac{n}{3^k} = 2 \Leftrightarrow n = 2 \cdot 3^k \Leftrightarrow k = \log_3 n - \log_3 2$$

Nach Einsetzen ergibt sich:

$$\begin{aligned} T(n) &= c \cdot 7^{\log_3 n - \log_3 2} + n^2 \cdot \sum_{i=0}^{\log_3 n - \log_3 5} \left(\tfrac{1}{3^2}\right)^i \\ &\overset{\text{summe konvergiert, } |q|<1}{=} \tfrac{c}{7^{\log_3 2}} 7^{\log_3 n} + dn^2 \\ &= c'7^{\log_3 n} + dn^2 \\ &= c'7^{\frac{\log_7 n}{\log_7 3}} + dn^2 \\ &= c'n^{\frac{1}{\log_7 3}} + dn^2 = c'n^{\log_3 7} + dn^2 \end{aligned}$$

Nun ist $\log_3 7 < 2$. Deshalb können wir das ganze abschätzen.

$$\Rightarrow T(n) = \Theta(n^2)$$

(f) $T(n) = 2T(\frac{n}{2}) + n \log n$
Hier können wir das Mastertheorem leider nicht andwenden.
Deshlab setzen wir ein:

$$\begin{aligned} T(n) &= 2T(\tfrac{n}{2}) + n \log n \\ &= 2^2 T(\tfrac{n}{2^2}) + \tfrac{n}{2^1} \log \tfrac{n}{2^1} + \tfrac{n}{2^0} \log \tfrac{n}{2^0} \\ &\quad \text{Nach k Schritten} \\ &= 2^k T(\tfrac{n}{2^k}) + \sum_{i=0}^{k-1} \tfrac{n}{2^i} \log \tfrac{n}{2^i} \\ &= 2^k T(\tfrac{n}{2^k}) + n \cdot \sum_{i=0}^{k-1} \tfrac{1}{2^i} (\log n - i) \\ &= 2^k T(\tfrac{n}{2^k}) + \left(n \log n \cdot \sum_{i=0}^{k-1} \tfrac{1}{2^i} \right) - n \cdot \sum_{i=0}^{k-1} \tfrac{i}{2^i} \end{aligned}$$

Per geeignetem Test kann man sehen, dass die beiden hinteren Reihen konvergieren.
(Die hinterste von beiden kann man gegen $\sum_{i=0}^{\infty} \frac{2^{0.5i}}{2^i} = \sum_{i=0}^{\infty} \frac{1}{2^{0.5i}}$ abschätzen.)
Der Anker liegt bei:

$$2 = \frac{n}{2^k} \Leftrightarrow k = \log n - 1$$

Es ergibt sich:

$$\begin{aligned} T(n) &= \tfrac{1}{2} n \cdot c + n \log n \cdot c_2 - n \cdot c_3 \\ &= \Theta(n \log n) \end{aligned}$$

(g) $T(n) = T(n-1) + \frac{1}{n}$
Wie man sieht, kann man diese Formel nicht auf Mastertheoremform bringen.
Deshalb setzen wir ein:

$$\begin{aligned} T(n) &= T(n-1) + \tfrac{1}{n} \\ &= T(n-2) + \tfrac{1}{n-1} + \tfrac{1}{n} \\ &\quad \text{Nach k Schritten} \\ &= T(n-k) + \sum_{i=0}^{k-1} \tfrac{1}{n-i} \end{aligned}$$

Den Anker erreichen wir bei

$$n - k = 2 \Leftrightarrow k = n - 2$$

Also gilt:

$$\begin{aligned} T(n) &= \Theta(1) + \sum_{i=0}^{n-3} \tfrac{1}{n-i} \\ &= \Theta(1) - \tfrac{1}{1} - \tfrac{1}{2} + \sum_{i=0}^{n-1} \tfrac{1}{n-i} \\ &= \Theta(1) + \sum_{i=1}^{n} \tfrac{1}{i} \\ &= \Theta(1) + O(\log n) = O(\log n) \end{aligned}$$

Den letzten Umformungsschritt haben wir auf dem ersten Übungszettel gesehen. Das ganze würde sich noch genauer durch die Euler-Mascheroni-Konstante abschätzen, aber diese Näherung war uns nah genug.

Aufgabe 3

(a) Implementieren Sie Karatsubas Algorithmus, sowie die Schulmethode zur Multiplikation ganzer Zahlen. Vergleichen Sie die Laufzeit beider Algorithmen experiementell und schätzen Sie ab, ab welcher Eingabe Karatsuba besser ist.

Die Schulmethode haben wir wie folgt implementiert:

```
private static BigInteger schulMultiplikationStat(BigInteger
    a, BigInteger b){
  BigInteger erg = new BigInteger("0");
  for(int i = 0; i< a.bitLength(); i++){
    //Wir iterieren einmal ueber die erste Zahl
    if(a.testBit(i)){
      //Und wenn das Bit gesetzt ist, addieren wir die n
          Stellen auf die Zahl drauf
      erg = erg.add(b.shiftLeft(i));
      counter += b.bitLength(); // Wir muessen die Stellen
          von b oft addieren
    }
  }
  return erg;
}
```

Wir iterieren einmal über die erste Zahl rüber. Wenn wir an der i-ten Stelle eine 1 haben, so addieren wir die Zahl auf unser Ergebnis. Dafür benötigen wir so viele Additionen, wie die zweite Zahl Bits hat. Wir haben uns an dieser Stelle das händische Addieren der einzelnen Bits gesparrt, aber mit in die Laufzeit eingerechnet. Über diese an sich zweifache Schleife kommen wir wieder auf eine quadratische Laufzeit.

Der Karatsumaalgorithmus sieht wie folgt aus:

```
private static BigInteger karatsubaStat(BigInteger a,
   BigInteger b){
  int n = Math.max(a.bitLength(), b.bitLength());
  if(n <= 2){
    counter += 4;
    return a.multiply(b);
  }
  int half = n / 2;

  BigInteger a_H = a.shiftRight(half);
  BigInteger a_L = a.subtract(a_H.shiftLeft(half));
  BigInteger b_H = b.shiftRight(half);
  BigInteger b_L = b.subtract(b_H.shiftLeft(half));

  /*
   * Formel aus der Vorlesung, mit 3 Rekursionsschritten
   */
  BigInteger high = karatsubaStat(a_H, b_H);
  BigInteger middle = karatsubaStat(a_H.add(a_L), b_H.add(
      b_L));
  BigInteger low = karatsubaStat(a_L, b_L);
  counter +=2; //Fuer die beiden Additionen in middle

  BigInteger highShifted = high.shiftLeft(2*half);
  BigInteger middleShifted = middle.subtract(high.add(low)).
      shiftLeft(half);

  // 2 Additionen in middleShifted und 2 im Ergebnis
  counter += 4;
  return highShifted.add(middleShifted).add(low);
}
```

Der Counter den wir mitführen zählt die Anzahl der Additionen. Er wird bei jedem Aufruf reseted und kann mit getCount() den letzten Wert ausgeben.
Wir nehmen hier einen sehr niedrigen Anker, von 2 Bits. Bei diesen 2 Bits wissen wir, das maximal 2 weitere Additionen ausgeführt werden, die wir auch mitzählen.
Wenn wir die Zahlen in unsere Nibble aufteilen, erhalten wir die obere Hälfte, indem wir die untere Hälfte einfach aus dem Bereich shiften. Da von oben 0en reingeshifted werden, steht am Ende nur noch der obere Teil da.
Den unteren Teil können wir gewinnen, indem wir den oberen Teil wieder zurück shiften und von der ursprünglichen Zahl subtrahieren. Danach steht folglich nur noch der untere Teil da.

Eine einfache Testmethode, die pro Durchgang größere Zahlen addiert, sollte uns liefern, ab welcher Länge Karatsuma besser als die Schulmethode ist:

```
...

189-Stellen - Schulmethode brauchte: 19467, Karatsuba
   brauchte: 18946, Hybrid brauchte: 5275
192-Stellen - Schulmethode : 18432, Karatsuba : 19478
195-Stellen - Schulmethode : 18330, Karatsuba : 19940
199-Stellen - Schulmethode : 20497, Karatsuba : 21046
202-Stellen - Schulmethode : 20604, Karatsuba : 20654
205-Stellen - Schulmethode : 20295, Karatsuba : 21718
209-Stellen - Schulmethode : 21945, Karatsuba : 21746
212-Stellen - Schulmethode : 23532, Karatsuba : 22838
215-Stellen - Schulmethode : 23865, Karatsuba : 23538
219-Stellen - Schulmethode : 25623, Karatsuba : 23650
222-Stellen - Schulmethode : 24864, Karatsuba : 24392
225-Stellen - Schulmethode : 27000, Karatsuba : 24392
229-Stellen - Schulmethode : 23358, Karatsuba : 24854
232-Stellen - Schulmethode : 28072, Karatsuba : 25582
235-Stellen - Schulmethode : 30315, Karatsuba : 26786
239-Stellen - Schulmethode : 27246, Karatsuba : 27038
242-Stellen - Schulmethode : 27346, Karatsuba : 27192
245-Stellen - Schulmethode : 29645, Karatsuba : 27962
249-Stellen - Schulmethode : 30876, Karatsuba : 29824
252-Stellen - Schulmethode : 31248, Karatsuba : 30258
255-Stellen - Schulmethode : 32640, Karatsuba : 31140
258-Stellen - Schulmethode : 34056, Karatsuba : 33128
262-Stellen - Schulmethode : 30654, Karatsuba : 32652
265-Stellen - Schulmethode : 35775, Karatsuba : 32764
...
```

Wir haben hier den Abschnitt gewählt, indem die Grenze lag. Wie man sieht wird Karatsuma bei uns um die 230 Bits herum langsam schneller als die Schulmethode. Aus Platzgründen haben wir hier die Hybride Laufzeit nicht mit beschrieben, in den Testläufen kann man sich aber überzeugen, das die Hybride Implementierung etwa 3-4 mal so schnell ist.

(b) Entwicklen Sie eine hybride Implementierung.

Die Hybride Methode ist nach unserer Implementierung nicht mehr schwer zu bestimmen. Wenn wir unseren Anker erreicht haben, wechseln wir einfach zur Schulmethode. Dazu sind wir über eine static Variable geganen um den Anker von aussen umsetzen zu können.

```
if(n <= hybridSwitch){
  return schulMultiplikationStat(a, b);
}
```

Nun sind wir über eine Zahl iteriert und haben in jedem Schritt einen anderen *hybridSwitch* gewählt:

```
1-Umschaltpunkt - Hybrid brauchte: 93429
2-Umschaltpunkt - Hybrid brauchte: 39588
3-Umschaltpunkt - Hybrid brauchte: 25712
4-Umschaltpunkt - Hybrid brauchte: 21148
5-Umschaltpunkt - Hybrid brauchte: 19549
6-Umschaltpunkt - Hybrid brauchte: 18877
7-Umschaltpunkt - Hybrid brauchte: 18050
8-Umschaltpunkt - Hybrid brauchte: 17935
9-Umschaltpunkt - Hybrid brauchte: 17941
10-Umschaltpunkt - Hybrid brauchte: 17978
11-Umschaltpunkt - Hybrid brauchte: 17995
12-Umschaltpunkt - Hybrid brauchte: 18321
13-Umschaltpunkt - Hybrid brauchte: 19700
14-Umschaltpunkt - Hybrid brauchte: 20692
15-Umschaltpunkt - Hybrid brauchte: 20734
```

An diesem Beispiel sehen wir, dass wir bei ungefähr 9 die beste Laufzeit erreichen. Im Testlauf zu a) kann man sich davon überzeugen, das es wirklich schneller läuft.Die Umschaltpunkte beschreiben jeweils die Anzahl der Maximalen Bits der beiden Zahlen.

Zur Benutzung des Programms:
Es kann mit

```
java MultiplikationBin -a
java MultiplikationBin -b
java MultiplikationBin
```

aufgerufen werden. Gibt man keinen Parameter an, so wird Aufgabenteil a) ausgeführt. Wählt man eine der Ergenzungen so wird man in den entsprechenden Teil ausgeführt.

Das Programm wurde mit java6 geschrieben und kann einfach Toplevel kompiliert werden.