Aufgabensammlung ADS-Repetitorium WS 25/26

Rekursive Algorithmen – Randomisierung – Hashing

Aufgabe 1: Rekursive Algorithmen

Geben Sie jeweils einen Algorithmus an, der die gegebene Laufzeit erfüllt. Alle Algorithmen sollen im Basisfall, also für $n = 1$, konstante Laufzeit haben. Als Eingabe bekommen die Algorithmen jeweils ein Feld der Länge n. Die Algorithmen sollten eine Rückgabe haben, doch der Wert der Rückgabe ist egal. Sie dürfen Rundungsfehler ignorieren.

(a) $T(n) = 3T(n-1) + n$

Lösung:

Algorithmus 1: recursiveAlgo(A[])

```
1 if A.size ≤ 1 then
2     return 42
3 for i = 1 to n do
4     A[i] = i
5 recursiveAlgo(A[1..n − 1])
6 recursiveAlgo(A[1..n − 1])
7 recursiveAlgo(A[1..n − 1])
8 return A[1]
```

(b) $T(n) = 4T(n/4) + O(n)$

Lösung:

Algorithmus 2: recursiveAlgo(A[])

```
1 if A.size ≤ 1 then
2     return 42
3 for i = 1 to n do
4     A[i] = i
5 recursiveAlgo(A[1..n/4])
6 recursiveAlgo(A[1..n/4])
7 recursiveAlgo(A[1..n/4])
8 recursiveAlgo(A[1..n/4])
9 return A[1]
```

(c) $T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + T(\sqrt{n})$; bitte beachten Sie hier die Rundungen.

Lösung:

Algorithmus 3: recursiveAlgo(A[])

```
1 if A.size ≤ 1 then
2     return 42
3 for i = 1 to √n do
4     A[i] = i
5 recursiveAlgo(A[1..⌈n/2⌉])
6 recursiveAlgo(A[1..⌊n/2⌋])
7 return A[1]
```

(d) $T(n) = 3T(n-5) + n \log n$

Lösung:

Algorithmus 4: recursiveAlgo(A[])

```
1 if A.size ≤ 1 then
2     return 42
3 MergeSort(A)
4 recursiveAlgo(A[1..n − 5])
5 recursiveAlgo(A[1..n − 5])
6 recursiveAlgo(A[1..n − 5])
7 return A[1]
```

(e) $T(n) = T(n-1) + T(n-2) + 42T(n/2) + n^2$

Lösung:

Algorithmus 5: recursiveAlgo(A[])

```
1 if A.size ≤ 1 then
2     return 42
3 InsertionSort(A)
4 recursiveAlgo(A[1..n − 1])
5 recursiveAlgo(A[1..n − 2])
6 for i = 1 to 42 do
7     recursiveAlgo(A[1..n/2])
8 return A[1]
```

Aufgabe 2: Rekursive Laufzeiten

Finden Sie für die nachstehenden Rekursionsgleichungen jeweils eine Funktion f, für die $T \in \Theta(f)$ gilt. Sie können davon ausgehen, dass die Laufzeit im Basisfall konstant ist.

(a) $T(n) = 4T(\lfloor n/2 \rfloor) + \frac{1}{2}n^2\sqrt{n}$

Lösung: Wir setzen $a = 4$, $b = 2$ und $f(n) = 0,5n^2\sqrt{n}$. Dann ist $\log_b a = \log_2 4 = 2$.
Damit gilt $f(n) \in \Omega(n^{2+\varepsilon})$ für $\varepsilon = 0,1$. Das ist der dritte Fall des Meistertheorems.
Damit ist $T(n) \in \Theta(0,5n^2\sqrt{n})$.

Die Regularitätsbedingung ist erfüllt:

$$\begin{aligned} 4 \cdot 0,5(n/2)^2\sqrt{n/2} = \cdot 0,5n^2\sqrt{n/2} &\leq c \cdot f(n) \\ \sqrt{n/2} &\leq c \cdot \sqrt{n} \\ \sqrt{n}/\sqrt{2} &\leq c \cdot \sqrt{n} \end{aligned}$$

Wähle $1/\sqrt{2} < c < 1$.

(b) $T(n) = 4T(\lfloor n/2 \rfloor) + n^2 \log n + n$

Lösung: Sei $a = 4$, $b = 2$ und $f(n) = n^2 \log n + n$. Dann ist $\log_b a = \log_2 4 = 2$ und $f(n) \notin \Omega(n^{2+\varepsilon})$. Deshalb wenden wir die Rekursionsbaum-Methode an:

Ebene	Knoten	Beitrag
0	1	$n^2 \log n + n$
1	4	$4 \cdot (n/2)^2 \log(n/2) + 4 \cdot n/2 = n^2 \log(n/2) + 2n$
2	16	$16 \cdot (n/4)^2 \log(n/4) + 4^2 \cdot n/4 = n^2 \log(n/4) + 4n$
3	64	$64 \cdot (n/8)^2 \log(n/8) + 4^3 \cdot n/8 = n^2 \log(n/8) + 8n$
i	4^i	$4^i \cdot (n/2^i)^2 \log(n/2^i) + 4^i \cdot n/2^i = n^2 \log(n/2^i) + 2^i \cdot n$
$\log_2 n$	$4^{\log_2 n} = n^2$	1

So ergibt sich die folgende Summe:

$$\begin{aligned} T(n) &= n^2 + \sum_{i=0}^{\log_2 n - 1} n^2 \log(n/2^i) + 2^i n = n^2 + n^2 \sum_{i=0}^{\log_2 n - 1} \log(n/2^i) + \frac{2^i}{n} \\ &= n + n^2 \left(\sum_{i=0}^{\log_2 n - 1} \log(n) - \sum_{i=0}^{\log_2 n - 1} \log(2^i) + \sum_{i=0}^{\log_2 n - 1} \frac{2^i}{n} \right) \\ &= n^2 + n^2 \left(\log_2 n \cdot \log(n) - \frac{(\log_2 n - 1)(\log_2 n)}{2} + \frac{1}{n} \sum_{i=0}^{\log_2 n - 1} 2^i \right) \\ &< n^2 + n^2 \left(\log_2 n \cdot \log(n) - \frac{(\log_2 n)(\log_2 n)}{2} + \frac{1}{n} \sum_{i=0}^{\log_2 n - 1} 2^i \right) \\ &= \Theta(n^2) + \Theta(n^2 \log^2 n) + \underbrace{\Theta(1)}_{\sum_{i=0}^{\log_2 n - 1} 2^i = n-1} \in \Theta(n^2 \log^2 n) \end{aligned}$$

(c) $T(n) = T(n-3) + 2n$

Lösung: Meistermethode nicht anwendbar, aber einfache Lösung möglich:

$$T(n) = 2n + 2(n-3) + 2(n-6) + \cdots = 2\sum_{i=0}^{n/3} n - 3i = \frac{n}{3} \cdot 2n - 6 \cdot \frac{(n/3)(n/3+1)}{2} \in \Theta(n^2)$$

(d) $T(n) = 2T(\lfloor n/4 \rfloor) + 3\sqrt{n}$

Lösung: Wir setzen $a = 2$, $b = 4$ und $f(n) = 3\sqrt{n}$. Dann ist $\log_b a = \log_4 2 = 0.5$.
Damit gilt $f(n) \in \Theta(n^{0.5})$. Das ist der zweite Fall des Meistertheorems.

Damit ist $T(n) \in \Theta(\sqrt{n} \log n)$.

(e) $T(n) = 3T(\lfloor n/2 \rfloor) + \frac{n}{6}$

Lösung: Wir setzen $a = 3$, $b = 2$ und $f(n) = n/6$. Dann ist $\log_b a = \log_2 3$.
Damit gilt $f(n) \in \mathcal{O}(n^{\log_2 3 - \varepsilon})$ für $\varepsilon = 0,1$. Das ist der erste Fall des Meistertheorems.
Damit ist $T(n) \in \Theta(n^{\log_2 3})$

(f) $T(n) = 3T(\lfloor n/5 \rfloor) + \frac{1}{2}\sqrt{n}$

Lösung: Wir setzen $a = 3$, $b = 5$ und $f(n) = 0,5\sqrt{n}$. Dann ist $\log_b a = \log_5 3$.
Damit gilt $f(n) \in \mathcal{O}(n^{\log_5 3 - \varepsilon})$ für $\varepsilon = 0,1$. Das ist der erste Fall des Meistertheorems.
Damit ist $T(n) \in \Theta(n^{\log_5 3})$

(g) $T(n) = 12T(\lfloor n/2 \rfloor) + n^4$

Lösung: Wir setzen $a = 12$, $b = 2$ und $f(n) = n^4$. Dann ist $\log_b a = \log_2 12 \approx 3,32$.
Damit gilt $f(n) \in \Omega(n^{3.32+\varepsilon})$ für $\varepsilon = 0,1$. Das ist der dritte Fall des Meistertheorems.
Damit ist $T(n) \in \Theta(n^4)$.
Die Regularitätsbedingung ist erfüllt:

$$\begin{aligned} 12 \cdot \frac{n}{2}^4 &\leq c \cdot n^4 \\ \frac{3n^4}{4} &\leq c \cdot n^4 \\ \frac{3}{4} &\leq c \end{aligned}$$

Wähle $3/4 < c < 1$.

(h) $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + \Theta(n \log n)$

Lösung: Meistermethode wegen des Θ-Terms nicht anwendbar. Deshalb Rekursionsbaum-Methode:

Ebene	Knoten	Beitrag
0	1	$n \log n$
1	2	$n \log n/2$
2	4	$n \log n/4$
i	2^i	$n \log n/2^i$
$\log_2 n$	n	n

Dadurch ergibt sich:

$$\begin{aligned} T(n) &= n + \sum_{i=0}^{\log_2 n - 1} n(\log_2 n - \log_2 2^i) \\ &= n + \left(n \sum_{i=0}^{\log_2 n - 1} \log_2 n \right) - \left(n \sum_{i=0}^{\log_2 n - 1} i \right) \\ &= n + n \log_2 n (\log_2 n - 1) - n \frac{\log_2 n (\log_2 n - 1)}{2} \\ &= n + n \frac{\log_2 n (\log_2 n - 1)}{2} \in \Theta(n \log^2 n) \end{aligned}$$

Aufgabe 3: Von Monte Carlo nach Las Vegas

(a) Geben Sie eine scharfe obere Schranke für die Fehlerwahrscheinlichkeit von Algorithmus **??** an.

Algorithmus 6: `FindLarge`(**int**[] A, **int** k)

input : Array A ist ein Feld ganzzahliger paarweise verschiedener Zahlen
output: Gibt ein Element zurück, dass mindestens so groß ist wie der Median

```
1 m = 0
2 for i = 1 to k do
3 |   randomly choose r ∈ {1, ..., n}
4 |   if A[r] > m then
5 |   |   m = A[r]
6 return m
```

Lösung: Wir geben ein falsches Ergebnis zurück, wenn in allen k Iterationen ein Element gezogen wird, das kleiner ist als der Median. Dies sind insgesamt $\lfloor n/2 \rfloor - 1$ Elemente. Somit haben wir eine Fehlerwahrscheinlichkeit von $\leq 1/2^k$.

(b) Geben Sie die Erfolgswahrscheinlichkeit mit jedem Schritt von Algorithmus **??** an.

Algorithmus 7: `FindRepeated`(**int**[] A)

input : Array A der Länge n mit ganzzahligen Zahlen, sodass $\lceil n/2 \rceil$ Zahlen identisch und $\lfloor n/2 \rfloor$ paarweise unterschiedlich sind.
output: Das identische Element.

```
1 while true do
2 |   randomly choose i ∈ {1, 2, ..., n}
3 |   randomly choose j ∈ {1, 2, ..., n} \ {i}
4 |   if A[i] = A[j] then
5 |   |   return A[i]
```

Lösung: Es gibt mindestens $n/2$ identische Elmente, somit ist die Wahrscheinlichkeit, dass $A[i]$ ein identisches Element ist mindestens $1/2$. Die Wahrscheinlichkeit, dass $A[j]$ ein identisches Element ist, ist dann mindestens $(n/2 - 1)/(n - 1)$. Somit ist die Erfolgswahrscheinlichkeit $\approx 1/4$.

(c) Was sind die Vor- und Nachteile beider Algorithmen?

Lösung: `FindLarge` könnte ein falsches Ergebnis zurückgeben, aber falls wir k sinnvoll wählen, z.B. $k = c \log_2 n$, dann ist die Laufzeit dieses Algorithmus schneller als jeder deterministische Algorithmus, der immer das korrekte Ergebnis liefert und hat eine sehr geringe Fehlerwahrscheinlichkeit. `FindRepeated` gibt auf jeden Fall immer das korrekte Ergebnis zurück aber könnte potentiell lange laufen. Allerdings ist die erwartete Anzahl an Iterationen 4 und somit erwartet konstant.

Aufgabe 4: Algorithmen und Zufall

Bestimmen Sie den Erwartungswert der Ausgabe von Algorithmus **??**.

Algorithmus 8:

```
1 s = 0
2 for i = 1 to n do
3 |   /* Random(a, b) gibt zufällig und gleichverteilt einen Wert zwischen a und b aus.
  |      */
4 |   if Random(1, 2 · i) ≤ i then
5 |   |   s = s + i
6 return s
```

Lösung: Sei X die Zufallsvariable, die den Rückgabewert repräsentiert. Sei X_i eine Indikatorzufallsvariable, die angibt, ob in Iteration i die Zeile 5 ausgeführt ($X_i = 1$) wird. Dann können wir X folgendermaßen bestimmen:

$$\mathbf{E}[X] = \mathbf{E}\left[\sum_{i=1}^{n} X_i \cdot i\right] = \sum_{i=1}^{n} \mathbf{E}[X_i] \cdot i$$

Nachdem X_i eine Indikatorzufallsvariable ist, ist $\mathbf{E}[X_i] = \mathbf{Pr}[X_i = 1] = i/(2i) = 1/2$. $\mathbf{E}[X]$ ist also

$$\mathbf{E}[X] = \sum_{i=1}^{n} \mathbf{E}[X_i] \cdot i = \sum_{i=1}^{n} \frac{1}{2} \cdot i = \frac{1}{2} \sum_{i=1}^{n} i = \frac{n(n+1)}{4}.$$

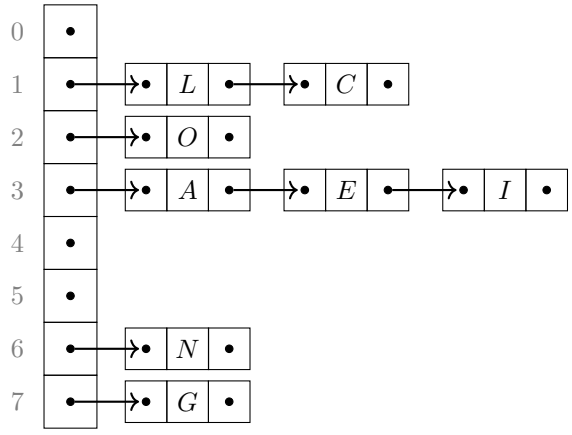
Aufgabe 5: Hashing ausführen

Gegeben seien die folgenden Schlüssel k mit entsprechenden Werten der Hashfunktion $h(k)$:

k	N	I	C	E	A	L	G	O
$h(k)$	6	3	1	3	3	1	7	2

(a) Fügen Sie die Schlüssel in obiger Reihenfolge in eine Hash-Tabelle der Größe 8 ein. Kollisionen sollen durch Verkettung aufgelöst werden.

Lösung: Da die Reihenfolge der Tabelle uns vorgibt, wann welcher Schlüssel eingefügt wurde und ein Eintrag einer Hashttabelle mit Verkettung eine einfach verkettete Liste ist, in der wir neue Elemente immer vorne einfügen, sieht die Hashtabelle folgendermaßen aus:



(b) Fügen Sie nun die Werte in der gleichen Reihenfolge in eine neue Hashtabelle mit Größe 8 ein. Lösen Sie Kollisionen mit linearem Sondieren auf. Geben Sie für jeden Schlüssel an, wie viele Felder der Tabelle betrachtet wurden.

Lösung:

Index	0	1	2	3	4	5	6	7
Schlüssel	O	C	L	I	E	A	N	G
Felder betrachtet	7	1	1	1	2	3	1	1

(c) Fügen Sie wieder die Schlüssel in eine Tabelle der Größe 8 ein. Verwenden Sie dafür folgende Hashfunktion $l(k, i)$, welche Kollisionen mittels doppelten Hashing auflöst. Wobei m die Tabellengröße ist.

$$l(k, i) = (h(k) + i \cdot h_1(k)) \mod m$$

Im Folgenden sind die Werte von $h_1(k)$ für die Schlüssel angegeben. Geben Sie für jeden Schlüssel an, wie viele Felder der Tabelle betrachtet wurden.

k	N	I	C	E	A	L	G	O
$h_1(k)$	3	5	3	7	5	1	7	3

Lösung:

Index	0	1	2	3	4	5	6	7
Schlüssel	A	C	E	I	L	O	N	G
Felder betrachtet	2	1	2	1	4	2	1	1

Aufgabe 6: Kosinus rekursiv

Es seien die folgenden Theoreme für $x, y \in \mathbb{R}$ gegeben:

Additionstheorem: $\cos(x+y) = 2\cos x \cos y - \cos(x-y)$

Periodizität: $\cos(x+2\pi) = \cos x$

Wir betrachten im Folgenden nur positive Winkel. Die Kosinusfunktion lässt sich für kleine Winkel x durch $\cos x \approx 1 - x^2/2$ annähern. Diese Näherung gelte für $x < 0,001$. Die Operatoren $+, -, \cdot, /$ und mod werden in konstanter Zeit ausgewertet.

(a) Geben Sie einen rekursiven Algorithmus zur Berechnung des Kosinus in Pseudocode an, der $\cos x$ für $x > 0$ mithilfe des Additionstheorems berechnet. Die gegebene Näherung soll als Abbruchkriterium der Rekursionsgleichung dienen. Die Laufzeit des Algorithmus soll in $\Theta(\log x)$ liegen.

Lösung: Falls $x = y$, dann vereinfacht sich das Additionstheorem zu $\cos(x+y) = 2\cos x \cos y - 1$. Die Idee des rekursiven Algorithmus besteht darin, die Eingabe zu halbieren, sodass zwei halb so große Kosinuswerte berechnet werden müssen. Sobald der Eingabewert kleiner als $0,001$ ist, verwenden wir die gegebene Abschätzung für kleine Winkel als Abbruchbedingung.

Algorithmus 9: double cos(x)

```
1 if |x| < 0,001 then
2     return 1 − x²/2
3 halfCos = cos(x/2)
4 return 2 · halfCos · halfCos − 1
```

(b) Nun soll die Worst-Case-Laufzeit durch Periodizität verbessert werden. Wie kann dies geschehen? Welche Laufzeit ergibt sich dann?

Lösung: Die Idee ist, den Eingabewert x auf 2π zu beschränken. Dies können wir mit dem mod Operator erreichen, der auf reellen Zahlen ein c mit $x = a \cdot 2\pi + c$ ausgibt. Da eine a ganze Zahl sein muss, ist der Wert c für ein bestimmtes x eindeutig definiert. Zu Beginn des Algorithmus aus a) fügen wir die Zeile $x = x \bmod 2\pi$ ein. Es gilt dann $x < 2\pi$ und die Laufzeit ist nach oben beschränkt durch $\mathcal{O}(\log_2 2\pi)$. Dies ist ein konstanter Wert, sodass unsere Kosinusfunktion jetzt in konstanter Zeit läuft.

(c) Was passiert, wenn Sie für $x < 0,001$ die Näherung $\cos x \approx 1$ verwenden?

Lösung: Die Rückgabe im Basisfall wäre dann immer 1. Die Ebene über dem Basisfall, also wenn $0,001 \leq x < 0,002$ gilt, würde ebenfalls 1 zurückgeben. Folglich würden alle Ebenen 1 berechnen und die Funktion würde konstant 1 zurückgeben. Dies ist offentsichtlich falsch, da der Kosinus keine konstante Funktion ist.

Aufgabe 7: 3-MergeSort

Gegeben sei folgende Algorithmus **??**, das eine Variante vom bekannten MergeSort ist.

Algorithmus 10: `3MergeSort`(**int**[] A, **int** $l = 1$, **int** $r = A.length$)

```
1 if l < r then
2     m1 = ⌊(r+l)/3⌋
3     m2 = ⌊2(r+l)/3⌋
4     3MergeSort(A, l, m1)
5     3MergeSort(A, m1 + 1, m2)
6     3MergeSort(A, m2 + 1, r)
7     Merge(A, l, m1, m2)
8     Merge(A, l, m2, r)
```

(a) Formulieren Sie die Laufzeit $T(n)$ von `3MergeSort` als Rekursionsgleichung in Abhängigkeit von der Länge des Feldes A.

Lösung:

$$T(n) = \begin{cases} \Theta(1) & \text{falls } n = 1 \\ 3T\left(\frac{n}{3}\right) + \Theta(n) \end{cases}$$

(b) Bestimmen Sie die asymptotische Laufzeit von `3MergeSort` mithilfe der Meistermethode. Geben Sie dabei den Fall an.

Lösung: Mit $a = 3$, $b = 3$ und $f(n) = \Theta(n)$ passt Fall 2 der Meistermethode, da $f \in \Theta(n^{\log_b(a)}) = \Theta(n)$ ist. Somit ist die asymptotische Laufzeit von `3MergeSort` $\Theta(n \log n)$.

Aufgabe 8: InsertionSort ∪ MergeSort

Obwohl die asymptotische worst-case Laufzeit von MergeSort $\Theta(n \log n)$ ist und die asymptotische worst-case Laufzeit von InsertionSort $\Theta(n^2)$ ist, läuft InsertionSort wegen der versteckten Konstanten in der O-Notation für kleine Eingaben oft schneller. Betrachte folgende Modifikation von MergeSort: Wir benutzen MergeSort bis wir n/k Teilfelder der Größe k haben. Diese sortieren wir mit InsertionSort und mergen sie anschließend mit dem bekannten Merge-Mechanismus von MergeSort.

(a) Zeigen Sie, dass InsertionSort die n/k Teilfelder insgesamt in $\Theta(nk)$ worst-case sortieren kann!

Lösung: Für jede der n/k Teilfelder der Länge k wird InsertionSort aufgerufen, die Gesamtlaufzeit ergibt sich demnach durch

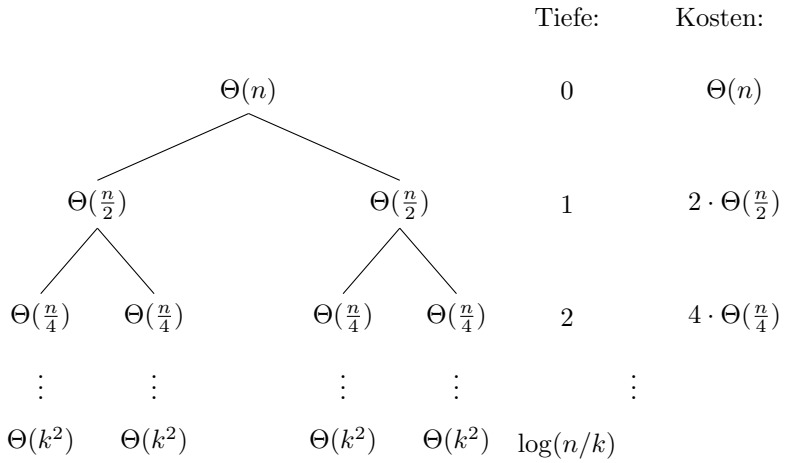
$$T(n) = \frac{n}{k} \cdot \Theta(k^2) = \Theta\left(\frac{nk^2}{k}\right) = \Theta(nk)$$

(b) Zeigen Sie, dass die oben beschriebene Variante von MergeSort in $\Theta(nk + n\log(n/k))$ worst-case läuft!

Lösung: Zur Berechnung der Laufzeit stellen wir zunächst die Rekursionsgleichung der modifizierten Variante auf:

$$T(n) = \begin{cases} \Theta(k^2) & \text{falls } n \leq k \\ 2T\left(\frac{n}{2}\right) + \Theta(n) & \text{sonst} \end{cases}$$

Diese Rekursionsgleichung lösen wir mit der Rekursionsbaummethode und mit Hilfe von der Teilaufgabe (**??**).



Sobald die Tiefe $\log(n/k)$ erreicht wurde, greift der Basisfall. Aus der Teilaufgabe (**??**) wissen wir bereits, dass die Gesamtkosten der Tiefe $\log(n/k)$ $\Theta(nk)$ ist. Das heißt die Gesamtkosten sind

$$T(n) = \Theta(nk) + \sum_{i=0}^{\log(n/k)-1} 2^i \Theta(n/2^i) = \Theta(nk) + \log(n/k)\Theta(n) = \Theta(nk + n\log(n/k))$$

(c) Was ist der größte Wert von k als Funktion von n für die der modifizierte MergeSort Algorithmus die gleiche asymptotische worst-case Laufzeit hat wie die ursprüngliche Variante? (mit O-Notation)

Lösung: Der ausschlaggebende Teil der Laufzeit ist der $\Theta(nk)$ Teil, der nicht $\Theta(n \log n)$ übersteigen darf. $\Rightarrow k \in \mathcal{O}(\log n)$.

(d) Wie sollte k in der Praxis gewählt werden?

Lösung: Wir wollen die Laufzeit so gut es geht reduzieren, das heißt wir wollen $nk + n\log(n/k)$ minimieren.

$$\frac{\mathrm{d}}{\mathrm{d}k}(c_1 \cdot nk + c_2 \cdot n\log(n/k)) = c_1 \cdot n - \frac{c_2' n}{k} = 0 \Leftrightarrow k = \frac{c_2'}{c_1}$$

Das ist auch in der Tat das Minimum. Der Parameter k ist also am besten eine Konstante. Diese muss empirisch ermittelt werden (sogar für jede Maschine), da dieser von den Konstanten c_1 und c_2 abhängt.

Aufgabe 9: Paranoia

Gegeben sei folgender Algorithmus **??**. Dabei ist das Set A eine endliche Menge von ganzzahligen Elementen der Länge n. Sie dürfen davon ausgehen, dass Einfüge-, Lösch- und Suchoperationen in $\mathcal{O}(1)$ möglich sind.

Algorithmus 11:

```
1  ParanoidMaximum(int[] A, ℓ = 1)
2      if A.length = ℓ then
3          return A[ℓ]
4      swap(A, ℓ, Random(ℓ, n))
5      a = A[ℓ]
6      b = ParanoidMaximum(A, ℓ + 1)
7      if b > a then
8          return b
9      else
10         for i = ℓ + 1 to A.length do
11             if A[i] > a then
12                 a = A[i]
13         return a
```

(a) Was berechnet Algorithmus **??** und ist die for-Schleife in der else Bedingung in den Zeilen 8-11 notwendig?

Lösung: Der Algorithmus berechnet das Maximum von $A[\ell..n]$. Nein, die else Bedingung in den Zeilen 8-11 ist nicht notwendig, denn falls $a \geq b$ ist, so ist a das Maximum in $A[\ell..n]$, weil b bereits das Maximum in $A[\ell + 1..n]$ ist.

(b) Berechnen Sie die worst-case Laufzeit des Algorithmus, indem Sie eine Rekursionsgleichung aufstellen und diese mittels Substitionsmethode beweisen!
Hinweis: Hier eignet es sich in der Rekursionsgleichung die Anzahl der Vergleiche zu zählen.

Lösung: Der worst-case tritt ein, wenn wir in jedem Rekursionsaufrauf in die else Bedingung in den Zeilen 8-11 kommen. Wir stellen eine Rekursionsgleichung für die Anzahl der Vergleiche auf,

nachdem diese proportional zu der gesamten Laufzeit des Algorithmus sind:

$$T(n) = \begin{cases} 1 & n = 1 \\ n + 1 + T(n-1) & \text{else.} \end{cases}$$

Wir behaupten, dass $T \in \mathcal{O}(n^2)$. Wir beweisen dies mittels Induktion über n.

Induktionsanfang $(n = 1)$**:** Hier sind wir auch im Basisfall der Rekursionsgleichung und es gilt $1 \leq c \cdot 1^2$ für jede Konstante $c \geq 1$.

Induktionshypothese: Für alle $k < n$ gelte $T(k) \leq c \cdot k^2$ für eine Konstante $c > 0$ und $n \geq 1$.

Induktionsschritt $(n \to n+1)$**:** Es gilt

$$\begin{aligned} T(n) &= (n+1) + T(n-1) \\ &\overset{\text{I.H.}}{\leq} (n+1) + c \cdot (n-1)^2 \\ &= n + 1 + cn^2 - 2c \cdot n + c \\ &= c \cdot n^2 + (1-2c) \cdot n + 1 + c \\ &\leq c \cdot n^2 + (1-2c) \cdot n + (1+c) \cdot n \quad \text{denn } n \geq 1 \text{ und } c > 0 \\ &= c \cdot n^2 + (2-c) \cdot n \\ &\leq c \cdot n^2 \quad \text{falls } c = 2 \end{aligned}$$

Somit gilt die Behauptung, dass $T(n) \leq c \cdot n^2$ für eine Konstante c und damit ist $T \in \mathcal{O}(n^2)$.

Wir nehmen nun im Folgenden an, dass A paarweise verschiedene Zahlen enthält. Weiterhin ist die Anordnung der Zahlen in A zufällig gleichverteilt.

(c) Definieren Sie eine geeignete Zufallsvariable, die beide Fälle in den Zeilen 6-11 in Abhängigkeit der Rekursionstiefe i angibt!

Lösung: Sei X_i eine Indikatorzufallsvariable, die in der i-ten Rekursion angibt, ob wir Zeile 7 oder die Zeilen 8-11 ausführen:

$$X_i = \begin{cases} 1, & a \geq b \\ 0 & \text{else.} \end{cases}$$

(d) Geben Sie eine Formel für die erwartete Laufzeit mit Hilfe der definierten Zufallsvariable aus Teilaufgabe (**??**) an!

Lösung: In jeder Rekursionsstufe wird die Funktion mit einem Element weniger aufgerufen. Nachdem $X_i \in \{0, 1\}$ angibt, ob wir Zeile 7 ausführen $(X_i = 0)$ oder Zeilen 8-11 ausführen $(X_i = 1)$,

können wir die erwartete Laufzeit mit der Linearität des Erwartungswerts ausdrücken:

$$\begin{aligned}\mathbf{E}[T(n)] &= \mathbf{E}\left[\Theta(1) + \sum_{i=2}^{n}(1 - X_i) \cdot \Theta(1) + X_i \cdot \Theta(n - i + 1)\right] \\ &= \Theta(1) + \sum_{i=2}^{n} \mathbf{E}[(1 - X_i)] \cdot \Theta(1) + \mathbf{E}[X_i] \cdot \Theta(n - i + 1) \\ &= \mathcal{O}(n) + \sum_{i=2}^{n} \mathbf{E}[X_i] \cdot \Theta(n - i + 1)\end{aligned}$$

Hierbei entspricht $\Theta(1)$ dem Basisfall und die Summe dem rekursiven Fall. Nachdem der Fall für Zeile 7 insgesamt höchstens Linearzeit in Anspruch nimmt, schätzen wir dies mit $\mathcal{O}(n)$ ab und konzentrieren uns auf den aufwändigen Fall.

(e) Mit welcher Wahrscheinlichkeit tritt der Fall in den Zeilen 8-11 auf? Was ist der Erwartungswert Ihrer Zufallsvariable aus Teilaufgabe (**??**)?

Lösung: Weil A keine Duplikate enthält, hängt die Wahrscheinlichkeit, dass a das größte Element ist, einzig von der Länge von A ab. Somit ist $\mathbf{Pr}[X_i = 1] = 1/(n - i + 1)$. Nachdem X_i eine Indikatorzufallsvariable ist, ist $\mathbf{E}[X_i] = \mathbf{Pr}[X_i = 1]$.

(f) Berechnen Sie nun die erwartete Laufzeit mit Hilfe der Formel in Teilaufgabe (**??**) und der Wahrscheinlichkeit aus Teilaufgabe (**??**).

Lösung: Wir müssen nur noch die Formeln zusammenführen und erhalten:

$$\begin{aligned}\mathbf{E}[T(n)] &= \mathcal{O}(n) + \sum_{i=2}^{n} \mathbf{E}[X_i] \cdot \Theta(n - i + 1) \\ &= \mathcal{O}(n) + \sum_{i=2}^{n} \mathbf{Pr}[X_i = 1] \cdot \Theta(n - i + 1) \\ &= \mathcal{O}(n) + \sum_{i=2}^{n} \frac{1}{n - i + 1} \cdot \Theta(n - i + 1) \\ &= \mathcal{O}(n) + \sum_{i=2}^{n} \mathcal{O}(1) = \mathcal{O}(n)\end{aligned}$$

Die erwartete Laufzeit ist also $\mathcal{O}(n)$.

Aufgabe 10: Wahrscheinlichkeit und Erwartungswert

Auf dem Frühjahrs-Volksfest gibt es ein faires Glücksrad, welches mit den fünf Farben rot, grün, blau, gelb und glitzer angemalt ist. Der rote und der grüne Sektor nehmen jeweils 25 % der Fläche des Glücksrads ein, die anderen drei Farben nehmen jeweils ein Sechstel des Glücksrads ein.

(a) Hängt die Wahrscheinlichkeit, dass der Zeiger am Ende einer Drehung auf eine bestimmte Farbe zeigt, davon ab, ob die Farben zusammenhängen oder das Glücksrad in nach außen gehenden Streifen bemalt ist?

Lösung: Die Wahrscheinlichkeiten sind unabhängig davon, wie das Glücksrad bemalt ist, solange der insgesamte Flächenanteil einer Farbe dem insgesamten Anteil am Umfang des Glücksrads gleicht. Damit wird sichergestellt, dass keine Farbe zur Zierde im Inneren der Glücksrads angebracht ist und der Zeiger, der ja auf den Umfang zeigt, diese Farbe nicht erreichen kann.

(b) Sie bekommen fünf Freifahrten, falls Sie „glitzer“ gedreht haben (Hauptpreis), drei Freifahrten, wenn Sie „gelb“ gedreht haben. Zwei Freifahrten gibt es, wenn der Zeiger „blau“ zeigt und eine Freifahrt ist der Gewinn bei „grün“. Die Farbe „rot“ geht leer aus. Wie viele Freifahrten gibt der Standbesitzer im Durchschnitt pro Besucher aus, wenn jeder Besucher einmal drehen darf?

Lösung: Wir definieren zunächst die Zufallsvariable F, die dem Ergebnis ω eines Versuchs die Anzahl der Freifahrten zuweist:

$$F(\omega) = \begin{cases} 0 & \text{falls } \omega = \text{„rot“} \\ 1 & \text{falls } \omega = \text{„grün“} \\ 2 & \text{falls } \omega = \text{„blau“} \\ 3 & \text{falls } \omega = \text{„gelb“} \\ 5 & \text{falls } \omega = \text{„glitzer“} \end{cases}$$

Der Erwartungswert $E[F]$ ist dann die Summe aus den numerischen Werten multipliziert mit den Wahrscheinlichkeiten, dass diese Werte auftreten:

$$E(F) = 0,25 \cdot 0 + 0,25 \cdot 1 + 0,1\overline{6} \cdot 2 + 0,1\overline{6} \cdot 3 + 0,1\overline{6} \cdot 5 = 1,91\overline{6}$$

Pro Drehung werden im Schnitt $1,91\overline{6}$ Freifahrten verschenkt.

(c) Die letzten neun Drehungen hatten alle das Ergebnis „glitzer“. Wie hoch ist die Wahrscheinlichkeit, dass die nächste, zehnte Drehung, erneut das Ergebnis „glitzer“ hat?

Lösung: Alle Drehungen sind voneinander unabhängig, daher ändert sich die Wahrscheinlichkeit nie. Die Antwort ist daher $P(\text{„glitzer“}) = 0,1\overline{6}$.

(d) Dem Standbetreiber ist es wichtig, dass die Kunden in der Warteschlange sehen, dass der Hauptpreis gewinnbar ist. Es sieht jeweils der nächste Kunde der Warteschlange und der vorherige Kunde, der gerade sein Gewinn abholt, das Ergebnis der aktuellen Drehung. Wenn an einem Tag n Kunden das Glücksrad drehen, wie viele sind dann im Durchschnitt Zeuge eines Hauptgewinns eines anderen Kunden?

Lösung: Sei Z die Anzahl der Kunden, die einen Hauptgewinn sehen. Gesucht ist dann $E[Z]$. Wir lösen die Aufgabe mit der Indikatorzufallsvariable Z_i für $0 \leq i \leq n-1$:

$$Z_i = \begin{cases} 1 & \text{falls } w_{i-1} \text{ oder } w_{i+1} \text{ die Farbe „glitzer“ gezeigt hat} \\ 0 & \text{sonst} \end{cases}$$

Die Wahrscheinlichkeit, dass Z_i gleich 1 ist, müssen wir durch eine Fallunterscheidung berechnen, da der erste und letzte Kunde jeweils nur einmal die Möglichkeit hat, einen Hauptgewinn zu sehen:

$$P(Z_i = 1) = E[Z_i] = \begin{cases} 0,1\overline{6}^2 + 2(1 - 0,1\overline{6}) \cdot 0,1\overline{6} & \text{falls } 0 < i < n-1 \\ 0,1\overline{6} & \text{sonst} \end{cases}$$

Nun gilt

$$\begin{aligned} E[Z] = E[\sum_{i=0}^{n-1} Z_i] = \sum_{i=0}^{n-1} E[Z_i] &= 0,1\overline{6} + \sum_{i=1}^{n-2} 0,1\overline{6}(2 - 0,1\overline{6}) + 0,1\overline{6} \\ &= 0,1\overline{6} + (n-2)0,1\overline{6}(2 - 0,1\overline{6}) + 0,1\overline{6} \\ &= 0,\overline{3} + (n-2)0,1\overline{6}(2 - 0,1\overline{6}) \\ &= 0,\overline{3} + (n-2)\frac{184}{625} \\ &= 0,\overline{3} + \frac{184}{625}n - \frac{368}{625} \\ &= \frac{184}{625}n - \frac{479}{1875} = \frac{552}{1875}n \approx 0,2944n - 0,25 \end{aligned}$$

Das heißt, etwa ein Drittel der Kunden sind Zeugen eines fremden Hauptgewinns.

Aufgabe 11: Zufall und Zufallsvariablen mit Würfeln

Gegeben seien zwei 6-seitige faire Würfel.

(a) Geben Sie den Ereignisraum Ω und dessen Kardinalität an, wenn beide Würfel nacheinander einmal geworfen werden.

Lösung: Jeder Würfel zeigt eine Zahl zwischen 1 und 6 an. Somit ist $\Omega = \{(1,1),(1,2),\ldots,(6,5),(6,6)\} = \{1,2,3,4,5,6\}^2$ und dadurch ist die Kardinalität $|\Omega| = 6^2 = 36$.

(b) Mit welcher Wahrscheinlichkeit würfelt der erste Würfel eine bestimmte Augenzahl i und der zweite eine bestimmte Augenzahl j?

Lösung: Die Wahrscheinlichkeit, dass $(i,j) \in \Omega$ ist $\mathbf{Pr}[(i,j)] = 1/|\Omega| = 1/36$.

(c) Mit welcher Wahrscheinlichkeit haben beide Würfel die gleiche Augenzahl?

Lösung: Sei A das Ereignis, das beschreibt, dass beide Würfel die gleiche Augenzahlen haben. Dann gilt: $A = \{(1,1),(2,2),(3,3),(4,4),(5,5),(6,6)\}$ und somit ist $\mathbf{Pr}[(i,j) \in A] = |A|/|\Omega| = 6/36 = 1/6$.

(d) Was ist der Erwartungswert des Maximums der beiden Augenzahlen eines Wurfes mit diesen zwei Würfeln?

Lösung: Sei $X = \max\{i,j\} \in \{1,2,3,4,5,6\}$ die Zufallsvariable, die das Maximum eines Wurfes mit Augenzahlen i und j angibt. Durch Zählen erhalten wir die Wahrscheinlichkeiten $\mathbf{Pr}[X = 1] = 1/36$, $\mathbf{Pr}[X = 2] = 3/36$, $\mathbf{Pr}[X = 3] = 5/36$, …, $\mathbf{Pr}[X = 6] = 11/36$ und somit erhalten wir den

Erwartungswert von X durch:

$$\mathbf{E}[X] = 1 \cdot \mathbf{Pr}[X = 1] + 2 \cdot \mathbf{Pr}[X = 2] + \cdots + 6 \cdot \mathbf{Pr}[X = 6] \approx 4.5$$

Aufgabe 12: Leichte Klausuren

Jede ADS-Klausur wird von den Studierenden nach Ihrer Schwierigkeit bewertet. Die erste Klausur wurde im Semster 0 geschrieben, die letzte Klausur im Semester $n+1$. Jede Klausur hat einen eindeutigen Rang erhalten, sodass sich Bewertungen von 0 bis $n+1$ ergeben. Wir betrachten eine Klausur im Semester i als *leicht*, falls für deren Bewertung b_i gilt, dass $b_i < b_{i-1}$ und $b_i < b_{i+1}$ ist.

Wir nehmen an, dass jede mögliche Folge der Bewertungen gleich wahrscheinlich ist, also alle Permutationen von $(b_0, b_1, \ldots, b_n, b_{n+1})$ mit gleicher Wahrscheinlichkeit auftreten können. Als Studierender möchten Sie berechnen, wie viele leichte Klausuren Sie in n Semestern erwarten können.

(a) Sei K eine Zufallsvariable, die die Anzahl der leichten Klausuren in den Semestern 1 bis n beschreibt. Angenommen, Sie hätten eine Formel, die die Wahrscheinlichkeit $P(K = k)$ für $0 \leq k \leq n$ berechnet. Wie könnten Sie dann $E[K]$ für ein festes n ausrechnen?

Lösung: In diesem Fall könnten Sie die allgemeine Definition des Erwartungswerts verwenden: $E[K] = \sum_{k=0}^{n} k \cdot P(K = k)$.

(b) Im Allgemeinen ist es schwierig, eine Formel für $P(K = k)$ zu finden. Die Fragestellung können Sie aber trotzdem lösen, indem Sie mit Indikatorzufallsvariablen arbeiten. Definieren Sie eine Indikatorzufallsvariable K_i die einem Semester $0 < i < n+1$ den Wert 0 oder 1 zuweist, je nachdem, ob die Klausur einfach war oder nicht. Wir können Sie K in Abhängigkeit von K_i für $0 < i < n+1$ berechnen?

Lösung:

$$K_i = \begin{cases} 1 \text{ falls } b_i < b_{i-1} \text{ und } b_i < b_{i+1} \\ 0 \text{ sonst} \end{cases}$$

$$K = \sum_{i=1}^{n} K_i$$

(c) Geben Sie außerdem eine Formel an, mit der Sie $E[K]$ in Abhängigkeit von $E[K_i]$ für $0 < i < n+1$ berechnen können. Welche Gesetzmäßigkeit des Erwartungswerts machen Sie sich zu Nutze?

Lösung: Um $E[K]$ zu berechnen, bilden wir den Erwartungswert der Summe $E[K] = E[\sum_{i=1}^{n} K_i]$. Dank der *Linearität* des Erwartungswerts können wir auch erst alle Erwartungswerte berechnen und dann die Summe bilden $E[K] = \sum_{i=1}^{n} E[K_i]$.

(d) Nehmen Sie an, Sie kennen eine Formel, die $P(K_i = 1)$ für ein festes i berechnet. Geben Sie eine Formel an, mit der Sie $E[K_i]$ berechnen können. Formulieren Sie in Worten, was die Wahrscheinlichkeit $P(K_i = 1)$ ausdrückt.

Lösung: Auch hier können wir die allgemeine Definition des Erwartungswerts verwenden. Da K_i nur 0 oder 1 annehmen kann, besteht die Formel nur aus einem Term $E[K_i] = 1 \cdot P(K_i = 1) + 0 \cdot P(K_i = 0) = P(K_i = 1)$, da der Faktor 0 wegfällt. Die gegebene Wahrscheinlichkeit $P(K_i = 1)$ ist die Wahrscheinlichkeit, dass die Klausur im Semester i eine leichte Klausur war.

(e) Wie groß ist die Wahrscheinlichkeit, dass die Bewertung b_i des Semesters i einen bestimmten Wert j aufweist *und* diese Klausur in Semester i eine leichte ist?

Lösung: Die gesuchte Wahrscheinlichkeit besteht aus zwei Teilen, die mit einem „und“ verknüpft sind. Wir betrachten beide Teile einzeln und verknüpfen Sie unter Beachtung der Pfadregel mit einer

Multiplikation. Die Wahrscheinlichkeit, dass die Bewertung b_i den Wert j aufweist, ist $1/(n+2)$, da es $n+2$ verschiedene Bewertungen gibt.

Es gibt offenbar j Zahlen, die kleiner als j sind, da unsere Zählung bei 0 beginnt. Also bleiben für die Bewertung b_{i-1} noch j Möglichkeiten und für die Bewertung b_{i+1} noch $j-1$ Möglichkeiten. Es bleiben $(n-1)!$ Möglichkeiten der anderen Werte $b_0, \ldots, b_{i-2}$ und $b_{i+2}, \ldots, b_{n+1}$. Da die Bewertung b_i den festen Wert j hat, gibt es insgesamt noch $(n+1)!$ Möglichkeiten, um die restlichen Bewertungen zu verteilen.

Die gesuchte Wahrscheinlichkeit ist also:

$$P(b_i = j \wedge (b_i < b_{i-1} \wedge b_i < b_{i+1}) = \frac{1}{n+2} \cdot \frac{j \cdot (j-1) \cdot (n-1)!}{(n+1)!} = \frac{j(j-1)}{(n+2)(n+1)n}$$

(f) Wie groß ist die Wahrscheinlichkeit $P(K_i = 1)$ für ein gegebenes i?

Lösung: Die Bewertung b_i kann $n+2$ verschiedene Werte j annehmen und für jeden dieser Werte j kennen wir die Wahrscheinlichkeit, dass $b_i = j$ und die Klausur ist eine leichte Klausur. Gemäß der Baumregel müssen wir die Wahrscheinlichkeiten für alle möglichen Werten von j aufaddieren. Dies können wir dann vereinfachen:

$$\begin{aligned}
P(X_i = 1) &= \sum_{j=0}^{n+1} \frac{1}{n+2} \cdot \frac{j \cdot (j-1) \cdot (n-1)!}{(n+1)!} = \frac{(n-1)!}{(n+2)(n+1)!} \sum_{j=2}^{n+1} j \cdot (j-1) \\
&= \frac{1}{(n+2)(n+1)n} \sum_{j=1}^{n} (j+1)j = \frac{1}{(n+2)(n+1)n} \left(\sum_{j=1}^{n} j^2 + \sum_{j=1}^{n} j \right) \\
&= \frac{1}{(n+2)(n+1)n} \Big(\frac{n(n+1)(2n+1)}{6} + \frac{n(n+1)}{2} \Big) = \frac{2n+1}{6(n+2)} + \frac{1}{2(n+1)} \\
&= \frac{2n+1+3}{6(n+2)} = \frac{2(n+2)}{6(n+2)} = \frac{1}{3}
\end{aligned}$$

(g) Was ist die erwartete Anzahl von Klausuren in den Semestern 1 bis n, die als leicht empfunden werden?

Lösung: Nun können wir alle Ergebnisse zusammenfassen. Wir wissen, dass $E[K] = \sum_{i=1}^{n} E[K_i]$. Da wir $E[K_i]$ kennen, setzen wir das in die Formel ein und lösen auf:

$$E[K] = \sum_{i=1}^{n} E[K_i] = \sum_{i=1}^{n} \frac{1}{3} = \frac{n}{3}$$

Wir können erwarten, dass ein Drittel der Klausuren als leicht empfunden werden.

Aufgabe 13: Indikatorzufallsvariablen und Maximum

Sei A ein Array paarweise unterschiedlicher ganzzahliger Zahlen der Länge n. Algorithmus **??** ermittelt die größte Zahl in A.

Algorithmus 12:

```
1 m = A[1]
2 for i = 2 to n do
3     if A[i] > m then
4         m = A[i]
5     return m
```

(a) Wie häufig wird im worst-case Zeile 4 ausgeführt?

Lösung: Falls A aufsteigend sortiert ist, wird Zeile 4 in jeder Iteration ausgeführt. Zeile 4 wird also $n-1$ mal ausgeführt.

(b) Definieren Sie eine Zufallsvariable X, die die Anzahl der Aktualisierungen von m angibt und definieren Sie zusätzlich Indikatorzufallsvariablen, mit der X gezählt werden kann.

Lösung: Sei $X \in \{1, \ldots, n\}$ die Zufallsvariable, die angibt, wie häufig m aktualisiert wird und seien X_i Indikatorzufallsvariablen, die folgendermaßen definiert sind:

$$X_i = \begin{cases} 1 & m \text{ wird in der } i\text{-ten Iteration aktualisiert} \\ 0 & \text{sonst.} \end{cases}$$

Weil m immer in Zeile 1 aktualisiert wird, ist $X = 1 + X_2 + X_3 + \cdots + X_n$

(c) Berechnen Sie den Erwartungswert von X!

Lösung: A enthält paarweise unterschiedliche Zahlen, somit ist die Wahrscheinlichkeit, dass $A[i]$ im Teilfeld $A[1..i]$ das Maximum ist $\mathbf{Pr}[X_i = 1] = 1/i$. Somit ist der Erwartungswert von X:

$$\begin{aligned} \mathbf{E}[X] &= \mathbf{E}[1 + X_2 + X_3 + \cdots + X_n] \\ &= 1 + \sum_{i=2}^{n} \mathbf{E}[X_i] = 1 + \sum_{i=2}^{n} \mathbf{Pr}[X_i = 1] = 1 + \sum_{i=2}^{n} \frac{1}{i} = \mathcal{H}_n \in \mathcal{O}(\log n) \end{aligned}$$

Aufgabe 14: Doppeltes Hashing

Welche der folgenden Funktionen eignen sich für eine Hashtabelle der Länge 25, wenn doppeltes Hashing verwendet wird und die Hashfunktion $h(k,i) = (h_0(k) + ih_1(k)) \bmod 25$ mit $h_0(k) = (4k+2) \bmod 25$ ist? Begründen Sie Ihre Entscheidungen.

(a) $h_1(k) = 1$

Lösung: Diese Hashfunktion eignet sich. Die Sondierfolge durchläuft alle Tabelleneinträge, da 1 und 25 teilerfremd sind. Diese Methode entspricht dem linearen Sondieren.

(b) $h_1(k) = 9 - (k \bmod 4)$

Lösung: Diese Hashfunktion eignet sich. Die Sondierfolge durchläuft alle Tabelleneinträge, da die Ergebnisse der Hashfunktion $\{6,7,8,9\}$ alle teilerfremd zu 25 sind.

(c) $h_1(k) = k \bmod 17$

Lösung: Diese Hashfunktion eignet sich *nicht*. Die Sondierfolge durchläuft alle nicht Tabelleneinträge, da $h_1(5) = 5$ nicht teilerfremd zu 25 ist.

(d) $h_1(k) = (3 + 5k) \bmod 25$

Lösung: Diese Hashfunktion eignet sich. Die Sondierfolge durchläuft alle Tabelleneinträge, da die Ergebnisse der Hashfunktion $\{3,8,13,18,23\}$ alle teilerfremd zu 25 sind.

(e) $h_1(k) = (4k - 1) \bmod 13$

Lösung: Diese Hashfunktion eignet sich *nicht*. Die Sondierfolge durchläuft alle nicht Tabelleneinträge, da $h_1(6) = 10$ nicht teilerfremd zu 25 ist.

Aufgabe 15: Gute Sondierfolgen erkennen

Gegeben sind folgende Hashfunktionen $h(k,i)$. Geben Sie an, um welche Methode der Kollisionsauflösung es sich handelt und gegebenfalls Probleme der Sondierfolgen, bei den gegebenen Tabellengrößen, an.

(a) $h(k,i) = (h_0(k) + (i+1) \cdot i) \mod m$, $m = 512$

Lösung:

$$h(k,i) = (h_0(k) + (i+1) \cdot i) \mod m = (h_0(k) + i^2 \cdot i) \mod m$$

Damit benutzt die Hashfunktion quadratisches Sondieren zur Kollisionsauflösung. Dabei tritt das Problem des sekundären Clusterings auf, also falls $h_0(k) = h_0(l)$ gilt, haben k und l die gleiche Sondierfolge. Ein weiteres Problem ist, dass $(i+1) \cdot i$ immer gerade ist und m ebenfalls gerade ist. Wenn $h_0(k)$ gerade ist, werden nur Felder mit geradem Index besucht. Wenn $h_0(k)$ ungerade ist, werden nur Felder mit ungeraden Index besucht.

(b) $h(k,i) = (h_0(k) + 2i) \mod m$, $m = 511$

Lösung: Die Hashfunktion benutzt lineares Sondieren. Da m ungerade ist, werden alle Felder besucht. Es tritt aber trotzdem das Problem des primären Clusterings auf.

(c) $h(k,i) = (h_0(k) + i \cdot h_1(k)) \mod m$, $h_1(k) = m - (2k \mod m) - 1$, $m = 16$.

Lösung: Die Hashfunktion benutzt doppeltes Hashing zur Kollisionsauflösung. Es treten keine Probleme bei dieser Hashfunktion auf, da $h_1(k)$ für alle k ungerade ist. Da m eine Zweierpotenz ist, werden alle Felder in der Sondierfolge besucht.