Aufgabensammlung ADS-Repetitorium WS 24/25

$\mathcal{O}$-Notation – inkrementelle Algorithmen – Sortieren

Aufgabe 1: Schleifeninvariante

Gegeben sei der folgende Algorithmus, der die Fakultät einer Zahl k berechnet.

Algorithmus 1: int fakultät(int k)

```
1 if k = 0 then
2     return 1
3 f = j = k
4 while j > 1 do
5     j = j − 1
6     f = f · j
7 return f
```

(a) Geben Sie eine geeignete Invariante an, mit der wir zeigen können, dass fakultät für Eingaben ≥ 1 korrekt arbeitet.

(b) Zeigen Sie mit Hilfe der in (a) aufgestellten Invariante die Korrektheit des Algorithmus.

Aufgabe 2: SelectionSort

Gegeben sei der folgende Sortieralgorithmus.

Algorithmus 2: SelectionSort(int[] A)

```
1 n = A.length
2 for i = 1 to n − 1 do
3     ℓ = i
4     for j = i to n do
5         if A[j] < A[ℓ] then
6             ℓ = j
7     Swap(A, i, ℓ)
8 return A
```

(a) Welche Laufzeit hat SelectionSort jeweils im besten und im schlechtesten Fall?

(b) Geben Sie eine geeignete Invariante an, um die Korrektheit von SelectionSort zu beweisen.

(c) Beweisen Sie die Korrektheit von SelectionSort. Für die innere Schleife muss kein Korrektheitsbeweis angegeben werden, es ist ausreichend zu beschreiben, was die Schleife berechnet.

Aufgabe 3: $\mathcal{O}$-, Θ- und Ω-Notation

Beweisen oder widerlegen Sie die Behauptungen. Arbeiten Sie mit der Definition aus der Vorlesung.

(a) $f(n) = \frac{1}{2}n - 2 \in \Omega(\log_2 n)$

(b) $f(n) = n^n + n^2 \in O(n^{n-1})$

(c) $f(n) = \frac{n^4 - 4n^2}{2n+7} \notin O(n^3)$

(d) $f(n) = \log_3(n^5 \cdot 9^{n^2}) \in \Omega(n \log_3 n)$

(e) $f(n) = \log_a n \in \Theta(\log_b n)$ für beliebige $a, b > 1$
(f) $f(n) = \frac{1}{100}n^2 + n \sin n \in \Theta(n^2)$
(g) $f(n) = n^4 - 10n^3 + 2n \in O(n^3)$
(h) $f(n) = \frac{9}{n} \notin \Omega(\frac{1}{\sqrt{n}})$

Aufgabe 4: Asymptotisches Wachstum von Funktionen
Ordnen Sie die Liste von Funktionen nach ihrem asymptotischen Wachstum ($\mathcal{O}(\dots) \subsetneq \mathcal{O}(\dots) \cdots \subsetneq \mathcal{O}(\dots)$). Nutzen Sie $=$ für das gleiche asymptotische Wachstum und $\subsetneq$ für unterschiedliches Wachstum. Beispiel: $f(n) = n$, $g(n) = 2n$, $h(n) = n^2$ dann gilt: $\mathcal{O}(f(n)) = \mathcal{O}(g(n)) \subsetneq \mathcal{O}(h(n))$.

$$\sqrt{n}\log_4(n^2),\ 5\sqrt{n}, 2^n,\ \log_2(n),\ n^2 - 7n,\ n\log_{10}(n/3),\ n^{\log_3(4)},\ \log_4(n^3),\ \sqrt{n}\log_2(n^{\sqrt{n}}),$$
$$4^{\log_3 n},\ 2^{2n},\ (n+1)^2,\ n(\log_2(n))^2,\ n!,\ 2^{3n\log_2(n)}$$

Aufgabe 5: Aufwandsanalyse
Gegeben seien die Funktionen $f \in \Theta(\log n)$ und $g(n) \in \Theta(n)$. Bestimmen Sie die Laufzeit der folgenden Programmfragmente in der Θ-Notation.

(a)

Algorithmus 3:

```
1 for i = 1 to n do
2     g(n)
```

(b)

Algorithmus 4:

```
1 i = 1
2 while i < n do
3     f(n)
4     g(n)
5     i = i · 2
```

(c)

Algorithmus 5:

```
1 i = n
2 while i > 0 do
3     f(n)
4     i = i/2
```

(d)

Algorithmus 6:

```
1 for i = 1 to n do
2     j = n
3     while j ≥ 1 do
4         for k = 1 to n do
5             g(n)
6         j = j/2
```

Aufgabe 6: Laufzeiten von Pseudocodes bestimmen
Analysieren Sie die Algorithmen 7-9 bezüglich der Laufzeit. Geben Sie asymptotisch scharfe Schranken in Θ-Notation an!

Algorithmus 7: Algo1(**int**[] A)

```
1 k = 5
2 i = 0
3 while i ≤ A.length − k do
4 |   MergeSort(A, i + 1, i + k)
5 |   i = i + k
6 MergeSort(A, i + 1, A.length)
```

Algorithmus 8: Algo2(**int**[] A)

```
1 x = 0
2 i = 1
3 while i ≤ A.length do
4 |   x = x + A[i]
5 |   i = 2 · i
6 return x
```

Algorithmus 9: Algo1(**int** n)

```
1 total = 0
2 for i = 1 to n do
3 |   x = 0
4 |   for j = 1 to i do
5 |   |   x = x + 1
6 |   total = total + x
7 return total
```

Aufgabe 7: Sortieralgorithmen

(a) Sortieren Sie das Feld A = [4, 3, 7, 2, 0, 9 ,8, 1, 5, 6] mit InsertionSort. Geben Sie nach jeder Iteration der äußeren Schleife das Feld an.

(b) Sortieren Sie das Feld B = [3, 7, 2, 9, 1, 4, 6, 5, 8, 0] mit HeapSort. Geben Sie bei jedem Schritt den entstandenen Heap an.

(c) MergeSort arbeitet rekursiv. Geben Sie für das Feld C = [9, 4, 1, 3, 5, 2, 6, 0, 8, 7] den Rekursionsbaum von MergeSort an. In jedem Knoten soll der jeweilige Aufruf von MergeSort und die zu sortierende Teilliste stehen, jeweils *vor* der Sortierung.

(d) Sortieren Sie das Feld D = [6, 4, 7, 9, 2, 3, 1, 5, 0, 8] mit einem vereinfachten QuickSort. Dieser schreibt alle Elemente, die kleiner als das Pivotelement sind, links und alle größeren rechts neben das Pivotelement. Zeichnen Sie den Rekursionsbaum. Schreiben Sie in jeden Knoten das zu sortierende Teilfeld nach dem Aufruf von Partition und markieren Sie das Pivotelement, die Wurzel sieht also so aus: [6, 4, 7, 0, 2, 3, 1, 5, 8, 9]. *Achtung: Das ist nicht der VL-Algorithmus, aber die Idee ist die gleiche!*

Aufgabe 8: Sortieren mit Quicksort

(a) Gegeben ist die Ausgabe der Methode Partition des Quicksort Algorithmus. Rekonstruieren Sie die Eingabe. Konkret sollen Sie das Array $A = \langle \_, \_, 1, \_, \_ \rangle$ so vervollständigen, dass der Aufruf Partition(A, 1, 5) die Zahl 3 zurückgibt und nach dem Aufruf gilt, dass $A = \langle 1, 2, 3, 4, 5 \rangle$ ist.

(b) Beweisen Sie die Korrektheit von Partition mittels Schleifeninvariante.

(c) Geben Sie für jede natürliche Zahl n eine Instanz der Länge n an, sodass QuickSort $\Omega(n^2)$ Zeit benötigt. Begründen Sie ihre Behauptung.

(d) Was müsste Partition (in Linearzeit) leisten, damit QuickSort Instanzen der Länge n in $\mathcal{O}(n \log n)$ Zeit sortiert? Zeigen Sie, dass Partition mit der von Ihnen geforderten Eigenschaft zur gewünschten Laufzeit von QuickSort führt.

Aufgabe 9: Suppentöpfe

Sie kennen das. Man will sich eine Nudelsuppe kochen, findet aber nicht den passenden Deckel für den Topf, da alle Deckel und Töpfe durcheinandergekommen sind. Da Sie immer auf Ihre Töpfe geachtet haben wissen Sie, dass zu jedem Topf ein Deckel vorhanden ist.

(a) Sie möchten ein *beliebiges* passendes Deckel-Topf-Paar finden. Wie viele Vergleiche sind dafür im besten Fall nötig?

(b) Geben Sie einen Algorithmus in Pseudocode an, der ein Feld T mit Topfgrößen und ein Feld mit Deckelgrößen D entgegennimmt. Die Ausgabe soll aus zwei Indizes i und j bestehen, sodass $\mathsf{D}[i] = \mathsf{T}[j]$. Wie viele Vergleiche braucht Ihr Algorithmus am schlechtesten Fall, um ein solches Paar zu finden? Können Sie Ihren Algorithmus verbessern, sodass er im schlechtesten Fall weniger Vergleiche braucht?

(c) Nun haben Sie genug von der Unordnung und möchten zu jedem Topf den passenden Deckel finden. Wie gehen Sie vor, um jedem Topf einen passenden Deckel zuzuordnen? Sie dürfen dabei nur Topf mit Topf und Deckel mit Deckel vergleichen. Verwenden Sie $\Theta(n \log n)$ Vergleiche.

(d) Lösen Sie nun Teilaufgabe c), aber diesmal sollen nur Vergleiche zwischen je einem Topf und einem Deckel verwendet werden. Die Anzahl der Vergleiche soll wieder in (erwartet) $\Theta(n \log n)$ liegen. Welchem Verfahren aus der Vorlesung ähnelt Ihre Vorgehensweise?

Aufgabe 10: Min-Heaps

Gegen sei folgender Min-Heap in Feld-Darstellung:

$$[-5, 6, 3, 11, 18, 10, 8, 13, 17, 19]$$

(a) Wandeln Sie den Min-Heap in Baumdarstellung um und begründen Sie, dass es sich um einen Min-Heap handelt.

(b) Fügen Sie in den Min-Heap die Zahl 15 ein. Geben Sie das Ergebnis als Feld an.

(c) Führen Sie auf dem originalem (nicht auf dem aus Teilaufgabe b) enstandenem) Min-Heap die Methode ExtractMin aus. Geben Sie das Ergebnis als Feld an.

(d) Geben Sie an, ob folgende Aussage korrekt ist. Begründen Sie ihre Antwort:
In einem Min-Heap hat der Knoten mit dem größten Element immer maximale Tiefe.

(e) Geben Sie an, ob folgende Aussage korrekt ist. Begründen Sie ihre Antwort:
Das drittkleinste Element in einem Min-Heap ist nicht notwendigerweise ein Kind der Wurzel.

Aufgabe 11: Pseudocode – Spot the Error

Die folgenden Algorithmen berechnen nicht das, was sie sollen. Erklären Sie, was der Fehler ist und schreiben Sie den richtigen Algorithmus auf. Geben Sie auch die asymptotische Worst-Case-Laufzeit in Θ-Notation an.

(a) Der Algorithmus soll $f(n) = n$ berechnen.

Algorithmus 11: int Algo1(int n)

```
1 zähler = 0
2 for i = 1 to n do
3 |  return zähler +1
```

(b) Der Algorithmus soll $f(n) = \sum_{i=0}^{n} i$ berechnen

Algorithmus 12: int Algo2(int n)

```
1 zähler = 0
2 for i = 1 to n do
3 |  zähler+1
4 return zähler
```

(c) Der Algorithmus soll $f(n) = n!$ berechnen.

Algorithmus 13: int Algo3(int n)

```
1 return Algo3(n − 1) · n
```

(d) Der Algorithmus soll **true** zurückgeben, wenn i im Array A enthalten ist, sonst **false**.

Algorithmus 14: boolean Algo4(int i, int[] A, int $l = 0$)

```
1 if A.length == l then
2     return false
3 else
4     return (i == A[l]) or Algo4(i, l + 1)
```

Aufgabe 12: Algorithmen und Laufzeiten

(a) Was berechnet der Algorithmus?
Wie viele Vergleiche, Additionen und Multiplikationen werden in Abhängigkeit von n ausgeführt?

Algorithmus 15: SomeAlgo(n)

```
1 int j = 0; int s = 1; int S = 0
2 while j < n do
3     S = S + s
4     j = j + 1
5     s = s · 2
6 return S
```

(b) Sei folgender Algorithmus zur Berechnung des Produkts $i \cdot (i+1) \cdot \ldots \cdot (j-1) \cdot j$ für natürliche Zahlen i und j mit $i < j$ gegeben:

Algorithmus 16: int Produkt(int j, int i)

```
1 return Fakultaet(j)/Fakultaet(i − 1)
```

Algorithmus 17: int Fakulaet(int x)

```
1 if x == 0 then
2     return 1
3 return x · Fakultaet(x − 1)
```

Begründen Sie kurz, warum der Algorithmus Produkt korrekt ist. Geben Sie die Worst-Case-Laufzeit von Produkt in Abhängigkeit von i und j an.

Aufgabe 13: Polynome evaluieren

Die Regel von Horner ist eine Möglichkeit Polynome zu evaluieren:

$$P(x) = \sum_{k=0}^{n} a_k x^k = a_0 + x(a_1 + x(a_2 + \cdots + x(a_{n-1} + xa_n)\ldots))$$

Folgender Pseudocode implementiert diese Regel:

Algorithmus 18: evaluatePolynomial($a_0, \ldots, a_n$, x)

```
1 y = 0
2 for k = n downto 0 do
3     y = a_k + x · y
4 return y
```

(a) Welche asymptotische Laufzeit hat evaluatePolynomial? Begründen Sie Ihre Antwort kurz!

(b) Schreiben Sie einen Pseudocode, der naiv das Polynom evauliert, indem jedes x^k komplett neu berechnet wird. Schätzen Sie die Laufzeit Ihres Algorithmus asymptotisch scharf in Θ-Notation ab!

(c) Zeigen Sie die Korrektheit von evaluatePolynomial mittels Schleifeninvariante!

Aufgabe 14: Vollständige Induktion

Zeigen Sie die folgenden Aussagen mittels vollständiger Induktion.

(a) Für jede natürliche Zahl n ist 3 ein Teiler von $n^3 - n$.

(b) Zeigen Sie, dass für alle $n \in \mathbb{N}$ gilt: $1 + 3 + \cdots + (2n - 1) = n^2$

(c) Die Fibonacci-Folge ist eine rekursiv definierte Zahlenfolge. Dabei ist $F(0) = 0$ und $F(1) = 1$. Die n-te Fibonacci-Zahl für ein $n > 1$ ist dann $F(n-1) + F(n-2)$. Die Berechnungsvorschrift dauert für große n jedoch sehr lange. Mit der Formel von Moivre-Binet kann die n-te Fibonacci-Zahl direkt ausgerechnet werden. Beweisen Sie die Richtigkeit der Formel:

$$F(n) = \frac{(\frac{1+\sqrt{5}}{2})^n - (\frac{1-\sqrt{5}}{2})^n}{\sqrt{5}}$$

(d) Auf einem quadratischen Schachbrett mit einer Seitenlänge von mehr als drei Feldern kann der Springer jedes Feld von jedem anderen Feld erreichen. Dafür hat er beliebig viele Züge zur Verfügung.

Aufgabe 15: Ähnliche Zahlen

Sei A ein Feld der Länge $n > 1$ von zufälligen Zahlen, wobei Zahlen mehrfach vorkommen dürfen.

(a) Geben Sie einen Algorithmus in Pseudocode an, der zwei Zahlen A[i] und A[j] mit $i \neq j$ sucht, sodass $|A[i]-A[j]|$ minimal ist. Der Algorithmus soll die Indizes beider Zahlen ausgeben und $\Theta(n^2)$ Zeit benötigen.

(b) Begründen Sie die Korrektheit Ihres Algorithmus, indem Sie die Korrektheit der inneren Schleife mit einer Invariante zeigen.

Aufgabe 16: Vereinigung

Geben Sie in gut kommentiertem Pseudocode einen Algorithmus an, der als Eingabe zwei aufsteigend sortierte Felder A und B erhält. die Ausgabe soll ein Feld C sein, das jede Zahl aus A und B genau einmal enthält. Die Laufzeit soll $O(n)$ sein, wobei n = A.length + B.length.

Aufgabe 17: Zusammenhängende Mengen

Gegeben sei ein Feld A von positiven, natürlichen Zahlen. Das Feld habe n Elemente. Das Feld A heißt *zusammenhängend*, wenn es zwei Zahlen $m, l \in \mathbb{N}$ gibt, sodass $\{A[1], \ldots, A[n]\} = \{m, m+1, \ldots, m+l\}$. Zum Beispiel ist das Feld $\langle 10, 5, 6, 10, 8, 7, 8, 9\rangle$ zusammenhängend, wobei $\langle 10, 5, 6, 10, 8, 9\rangle$ nicht zusammenhängend ist, da die Zahl 7 fehlt. Geben Sie einen Algorithmus an, der in $\mathcal{O}(n)$ Zeit entscheidet, ob das gegebene Feld A zusammenhängend ist.

Aufgabe 18: Elemente ausgeben

Gegeben seien zwei unsortierte Arrays mit ganzen Zahlen A und B mit jeweils n Elementen. Die Elemente eines Arrays sind dabei paarweise verschieden. Entwickeln Sie einen Algorithmus, dessen worst-case Laufzeit $\mathcal{O}(n \log n)$ ist, der alle Elemente von A ausgibt, die **nicht** in B vorkommen. Finden Sie auch einen Algorithmus, der das Problem in (erwartet) $\mathcal{O}(n)$ löst?

Aufgabe 19: Vereinigung von Intervallen

Gegeben sei eine Liste $R = \langle [x_1, y_1], \ldots, [x_n, y_n]\rangle$ von gegebenenfalls überlappenden Intervallen. Geben Sie einen Algorithmus an, der die Gesamtlänge der Vereinigung von den Intervallen in R angibt. Der Algorithmus soll eine asymptotische worst-case Laufzeit von $O(n \log n)$ haben!
Hinweis: Abbildung 1 zeigt ein Beispiel. Könnte eine gewisse Sortierung der Intervalle helfen?

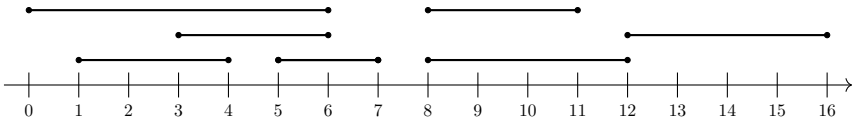

Abbildung 1: Mögliche Intervalle in R. Die Gesamtlänge der Vereinigung dieser Intervalle ist 15.

Aufgabe 20: Flugsicherheit

Im Flugverkehr müssen die Flugzeuge gewisse Abstände einhalten. Gegeben ist eine unsortierte Liste von Flugzeugen. Jedes Flugzeug a hat drei Attribute, nämlich $a.x$, $a.y$ und $a.z$. Diese Attribute geben die Koordinaten im Luftraum an. Sie sollen einen Algorithmus angeben, der true ausgibt, falls sich zwei Flugzeuge näher als den Abstand d kommen. Ihr Kommilitone hat einen Algorithmus entwickelt (siehe Algo 24), der dieses Problem lösen soll.

Algorithmus 24: planesTooClose(Plane[ ] planes, d)

```
1  mergeSort(planes) // Sortiere nach y-Koordinate
2  if planes.length < 2 then
3      return false
4  for i = 2 to planes.length do
5      f_1 = planes[i]
6      f_2 = planes[i − 1]
7      yDistance = |f_1.y − f_2.y|
8      if yDistance < d then
9          e = sqrt((f_1.x − f_2.x)^2 + (f_1.y − f_2.y)^2 + (f_1.z − f_2.z)^2)
10         if e < d then
11             return true
12 return false
```

(a) Welche Laufzeit hat dieser Algorithmus, wenn man davon ausgeht, dass eine Wurzeloperation $\mathcal{O}(1)$ Zeit benötigt?

(b) Ist der Algorithmus korrekt? Begründen Sie Ihre Antwort.