Aufgabensammlung ADS-Repetitorium WS 25/26

Grundlagen – Analyse – Sortieren

Aufgabe 1: 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 2: Schleifeninvariante

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

Algorithmus 2: 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 3: Induktionsbeweis eines Algorithmus

Betrachten sie folgenden Algorithmus zur Berechnung der Fakultät:

Algorithmus 3: int fakultät(int k)

```
1 f = j = k
2 while j > 1 do
3     j = j − 1
4     f = f · j
5 return f
```

(a) Geben Sie einen Algorithmus in Pseudocode an, der die Fakultät rekursiv berechnet.

(b) Beweisen Sie mittels vollständiger Induktion die Korrektheit Ihrer rekursiven Variante.

Aufgabe 4: Laufzeiten von Pseudocodes bestimmen

Analysieren Sie die Algorithmen 5-7 bezüglich der Laufzeit. Geben Sie asymptotisch scharfe Schranken in Θ-Notation an!

Algorithmus 5: 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 6: 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 7: 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 5: 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 6: 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 7: 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 8: 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 9: 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 10: 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 11: 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 8: 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 9: 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 10: SelectionSort
Gegeben sei der folgende Sortieralgorithmus.

Algorithmus 13: 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 11: 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 12: Korrektheitsbeweise und Rekursion
Betrachten Sie den folgenden Algorithmus.

Algorithmus 14: int doSomethingSimple(**int** A[ ], **int** $i = 1$)

```
Data: Feld mit natürlichen Zahlen A, natürliche Zahl i
Result: Ein Wert, der mit A zusammenhängt
1 if i == A.length then
2     return A[i]
3 k = doSomethingSimple(A, i + 1);
4 if k > A[i] then
5     return k
6 else
7     return A[i]
```

(a) Beschreiben Sie in einem Satz, was der Algorithmus macht.

(b) Beweisen Sie die Korrektheit des Algorithmus.

(c) Geben Sie einen Algorithmus an, der äquivalent zu doSomethingSimple ist, ohne Rekursion zu verwenden.

(d) Geben Sie eine Schleifeninvariante für Ihren inkrementellen Algorithmus an.

(e) Beweisen Sie die Korrektheit Ihres Algorithmus mit der von Ihnen aufgestellten Schleifeninvariante.

Aufgabe 13: Algorithmen und Laufzeiten

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

Algorithmus 16: 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 17: int Produkt(int j, int i)

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

Algorithmus 18: 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 14: 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 19:

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

(b)

Algorithmus 20:

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

(c)

Algorithmus 21:

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

(d)

Algorithmus 22:

```
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 15: $\mathcal{O}$-, 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 16: 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?