Image Analysis Lab
Jun.-Prof. Dr.
Olaf Ronneberger
Claudius Korzen

Algorithmen und Datenstrukturen (ESE + IEMS) WS 2014 / 2015

http://lmb.informatik.uni-freiburg.de/lectures/

UNI FREIBURG

Exam with Solutions

Freitag, 27. Februar 2015, 16:00 Uhr - 18:00 Uhr, Gebäude 082, Raum 00-006 (Kinohörsaal)

Friday, February 27, 2015, 4.00 pm - 6.00 pm, building 082, room 00-006 (Kinohörsaal)

Nachname: / Last name:

Vorname: / First name:

Matrikelnummer: / Matriculation number:

Studiengang: / Degree course: ◯ **Bachelor** ◯ **Master**

Durch den Antritt dieser Prüfung erklären Sie sich für prüfungsfähig. Sollten Sie sich während der Prüfung nicht prüfungsfähig fühlen, können Sie aus gesundheitlichen Gründen auch während der Prüfung von dieser zurücktreten. Gemäß der Prüfungsordnungen sind Sie verpflichtet, die für den Rücktritt oder das Versäumnis geltend gemachten Gründe unverzüglich (innerhalb von 3 Tagen) dem Prüfungsamt durch ein Attest mit der Angabe der Symptome schriftlich anzuzeigen und glaubhaft zu machen. Weitere Informationen hierzu können auf den Internetseiten des Prüfungsamtes nachgelesen werden.

By taking part in this exam you agree that there are no health reasons that hinder you taking part in the exam. Should you feel that you have health problems during the exam, you may abort this exam. In such a case you must provide an official medical certificate by a doctor including the symptoms within 3 days. More details can be obtained at the examination office.

Unterschrift des Studenten: / Student signature:

Punktzahl/Score:

A1	A2	A3	A4	A5	A6	A7	$\sum$
/10	/20	/8	/20	/10	/10	/22	/100

Note/Grade:

Einsicht am:

Unterschriften der Prüfer:

Es gibt 7 Aufgaben. Sie haben insgesamt 120 Minuten Zeit. Die erreichbare Punktzahl einer Aufgabe entspricht ihrem geschätzten Zeitaufwand (1 Punkt entspricht 1 Minute). Bei einer Gesamtpunktzahl von 100 Punkten sind das 100 Minuten.

Sie dürfen eine beliebige Menge an Papier, Büchern, etc. verwenden. Sie dürfen keinerlei elektronische Geräte wie Notebook, Mobiltelefon, Smartwatches, etc. verwenden, insbesondere keine Geräte, mit denen Sie mit Dritten kommunizieren oder sich mit dem Internet verbinden können. Bei einem Täuschungsversuch wird die Prüfungsleistung mit *nicht ausreichend* (5,0) bewertet. Bereits der Besitz unzulässiger Hilfsmittel zählt als Täuschungsversuch.

Sie dürfen Ihre Lösungen entweder in Deutsch oder Englisch aufschreiben. Schreiben Sie Ihre Lösungen NICHT in Bleistift und nicht in der Farbe Rot. Schreiben Sie Ihre Lösungen bitte wenn möglich auf die Klausur! Benutzen Sie dabei für jede Aufgabe zuerst die Vorderseite und dann die Rückseite. Wenn Sie zusätzliches Papier benötigen, schreiben Sie auf jedes dieser Blätter bitte oben rechts gut lesbar Ihren Namen (in BLOCKBUCHSTABEN) und Ihre Matrikelnummer.

Tipp: Verbringen Sie nicht zu viel Zeit mit einer einzelnen Teilaufgabe. Wenn Sie nicht weiterkommen, machen Sie erstmal mit einer anderen (Teil-)Aufgabe weiter. Widmen Sie sich dann am Ende, wenn Sie noch Zeit haben, den Teilen, bei denen Sie Schwierigkeiten hatten.

Wir wünschen Ihnen viel Erfolg!

There are 7 tasks. You have 120 minutes of time overall. The score of a task correlates to the estimated time period to solve the task (1 point corresponds to 1 minute). So the estimated time period to complete the exam is 100 minutes (at a total score of 100 points).

You are allowed to use any amount of paper, books, etc. You are not allowed to use any computing devices, mobile phones, smartwatches, etc. in particular nothing with which you can communicate with others or connect to the Internet. Any attempt to cheat will result in the grade “failed” (5.0). This already includes the possession of forbidden material.

You may write down your solutions in either English or German. Please use neither a pencil nor a red pencil. Please write down your solutions on this hand-out! You can also use the back side of the pages. If you need additional pages, please write your matriculation number and your name (in PRINTED LETTERS) on each of them.

Hint: Don’t waste your time on a single subtask. If you are unable to solve a task, continue with another (sub-)task first. Come back to such a task at the end, if you have enough time left over.

Good luck!

Aufgabe 1 (10 Punkte)

Berechnen Sie die Editierdistanz zwischen den folgenden Zeichenketten x und y. Geben Sie die vollständige dynamische Tabelle an und zeichnen Sie alle optimalen Pfade ein. Benennen Sie schließlich für einen optimalen Pfad Ihrer Wahl die Folge der entsprechenden Editieroperationen, um x in y zu überführen.

Compute the edit distance between the following strings x and y. Write down the entire dynamic table and mark all optimal paths. For a single optimal path of your choice, write down the sequence of related edit operations to transform x into y.

$x =$ `PFERDE`
$y =$ `APFEL`

	ϵ	P	F	E	R	D	E
ϵ	0	1	2	3	4	5	6
A	1	1	2	3	4	5	6
P	2	1	2	3	4	5	6
F	3	2	1	2	3	4	5
E	4	3	2	1	2	3	4
L	5	4	3	2	2	3	**4**

Edit distance: 4
The sequences of edit operations of the three optimal paths are:

(1)

```
insert A at index 0.
delete R at index 4.
delete D at index 5.
replace E by L at index 6.
```

(2)

```
insert A at index 0.
delete R at index 4.
replace D by L index 5.
delete E at index 6.
```

(3)

```
insert A at index 0.
replace R by L at index 4.
delete D at index 5.
delete E at index 6.
```

Punktevergabe:
Grundgerüst der Tabelle korrekt: 1 Punkt
Übergänge (Pfeile) korrekt: 1 Punkt
Zahlen (Distanzen) korrekt: 1 Punkt
Editierdistanz korrekt: 1 Punkt
Alle Pfade richtig eingezeichnet: 2 Punkte (1 Punkt für ersten Pfad + 0.5 Punkte pro weiteren)
Editieroperationen korrekt: 4 Punkte (1 Punkt pro richtiger Editieroperation)

Aufgabe 2 (20 Punkte)

Schreiben Sie eine Methode (in Java or C++), die die Länge der längsten gemeinsamen Zeichenkette zwischen zwei gegebenen Zeichenketten x und y in Zeit $\mathcal{O}(|x| \cdot |y|)$ berechnet. Begründen Sie die Laufzeit Ihrer Methode.

Hinweis: Für $x =$ `PFERDE` und $y =$ `APFEL` ist die längste gemeinsame Zeichenkette `PFE`. Ihre Methode soll in diesem Fall also die Zahl 3 ausgeben. Sie dürfen annehmen, dass alle Zeichen in x und y Großbuchstaben sind.

Write down a method (in Java or C++) which computes the length of the longest common substring between two given strings x und y in time $\mathcal{O}(|x| \cdot |y|)$. Substantiate the runtime of your method.

Hint: For $x =$ `PFERDE` und $y =$ `APFEL`, the longest common substring is `PFE` and your method should return the number 3 in this case. You may assume that all characters in x and y are in uppercase.

Alternative 1:

```
public int computeLCS(String x, String y) {
  int maxLength = 0;
  int[][] table = new int[x.length() + 1][y.length() + 1];

  for (int i = 0; i <= x.length(); i++) { table[i][0] = 0; }       // O(|x| + 1)
  for (int j = 0; j <= y.length(); j++) { table[0][j] = 0; }       // O(|y| + 1)

  for (int i = 1; i <= x.length(); i++) {                          // O(|x|)
    for (int j = 1; j <= y.length(); j++) {                        // O(|y|)
      if (x.charAt(i - 1) == y.charAt(j - 1)) {                    // O(1)
        table[i][j] = table[i-1][j-1] + 1;                         // O(1)
        if (table[i][j] > maxLength) {                             // O(1)
          maxLength = table[i][j];                                 // O(1)
        }
      }
    }
  }
  return maxLength;
}
```

Runtime:
$\mathcal{O}(|x| + 1) + \mathcal{O}(|y| + 1) + \mathcal{O}(|x|) \cdot \mathcal{O}(|y|) \cdot 4 \cdot \mathcal{O}(1)$

$= \mathcal{O}(|x|) + \mathcal{O}(|y|) + \mathcal{O}(|x|) \cdot \mathcal{O}(|y|)$
$= \mathcal{O}(|x|) \cdot \mathcal{O}(|y|)$
$= \mathcal{O}(|x| \cdot |y|)$

Alternative 2:

```
int computeLCS(std::string x, std::string y) {
  int maxLen = 0;
  for (int offset = -x.size()+1; offset < x.size(); ++offset) {    // O(|x|)
    int curLen = 0;                                                // O(1)
    for (int yi = 0; yi < y.size()); ++yi) {                       // O(|y|)
      xi = yi+offset;                                              // O(1)
      if( xi >= 0 && xi < x.size()) {                              // O(1)
        if( x[xi] == y[yi]) {                                      // O(1)
          curLen++;                                                // O(1)
          maxLen = std::max(curLen, maxLen);                       // O(1)
        } else {
          curLen = 0;                                              // O(1)
        }
      }
    }
  }
  return maxLen;
}
```

Punktevergabe:
Alle Buchstabenkombinationen durchprobiert: 6 Punkte
„Diagonalläufe“ vorhanden: 6 Punkte
Algorithmus berechnet korrekte Ergebnisse: 3 Punkte
Laufzeit in $\mathcal{O}(|x| \cdot |y|)$: 3 Punkte
Laufzeit korrekt begründet: 2 Punkte

Aufgabe 3 (8 Punkte)

Ein Ternärbaum ist definiert als ein Baum, in dem jeder innere Knoten genau drei Nachfolger hat. Zeigen Sie per vollständiger Induktion, dass ein vollständiger Ternärbaum der Höhe d immer $n(d) = \frac{3^d-1}{2}$ Knoten hat.

A ternary tree is defined as a tree, where every inner node has exactly three child nodes. Prove by complete induction that the number of nodes in a complete ternary tree with height d is always $n(d) = \frac{3^d-1}{2}$.

Induction basis: $d = 1$.

$$n(1) = \frac{3^1 - 1}{2} = 1 \quad \text{1 Punkt}$$

Correct, because a tree with height 1 has exactly one node (the root node). 1 Punkt

Induction step: $(d-1) \to d$.

A full ternary tree with height d consists of three full ternary trees with height $d-1$ and an additional node (the root node): 1 Punkt

$$n(d) = 3 \cdot n(d-1) + 1 \quad \text{1 Punkt}$$

Apply the induction claim:

$$\begin{aligned} n(d) &= 3 \cdot \left(\frac{3^{d-1} - 1}{2} \right) + 1 \\ &= \frac{3 \cdot 3^{d-1} - 3}{2} + 1 \\ &= \frac{3^d - 3}{2} + \frac{2}{2} \\ &= \frac{3^d - 1}{2} \quad \text{4 Punkte} \end{aligned}$$

□

Aufgabe 4 (20 Punkte)

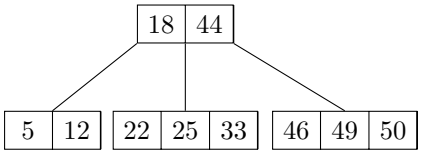
Gegeben seien die unten abgebildeten $(2, 4)$-Bäume.

4.1 (10 Punkte) Fügen Sie nacheinander die Elemente 7, 28 und 48 in Baum (a) ein. Zeichnen Sie jeweils den Zustand des Baumes nach jeder Einfügung und beschreiben Sie jeweils die nötigen Zwischenschritte, um die Bedingungen an den Baum aufrecht zu erhalten.

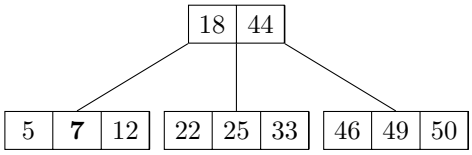
Given the $(2, 4)$-trees below.

4.1 (10 points) Insert the elements 7, 28 and 48 into tree (a), one after another. Draw the state of the tree after each single insertion and describe the required steps to maintain the tree conditions.

(a)

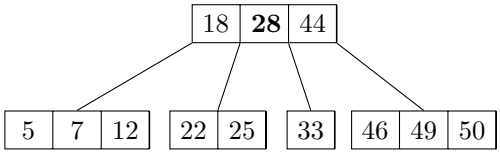


1. Insert element 7 into the leftmost leaf; this leaf has enough space:



2 Punkte

2. Insert element **28** into the middle leaf; this leaf now consists of (22,25,**28**,33), which are too many elements; split this leaf into (22,25) and (33), the element (**28**) climbs up into its parent node, where it has enough space: Unterstrichene Aussagen wurden (sinngemäß) genannt: jeweils 1 Punkt (ins. 2 Punkte)

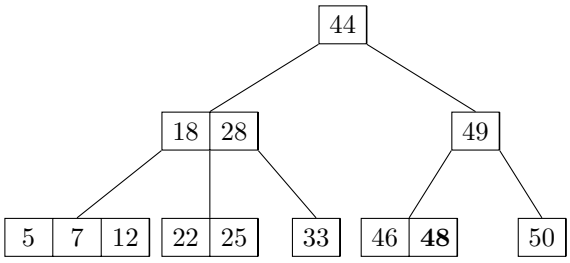


1 Punkt

KEIN Punktabzug, wenn Blatt in (22) und (28,33) aufgespalten wurde.

KEIN Punktabzug, wenn die notwendigen Schritte eindeutig aus Zeichnung hervorgegangen sind.

3. Insert element **48** into the rightmost leaf; this leaf now consists of (46,**48**,49,50), which are too many elements; split this leaf into (46,**48**) and (50), the element 49 climbs up into the root node; the root node now consists of (18, 28, 44, 49), which are again too many elements; split the root node into (18,28) and (49). The node (44) becomes the new root, the height of the tree increases by 1. jeweils 1 Punkt (ins. 4 Punkte)



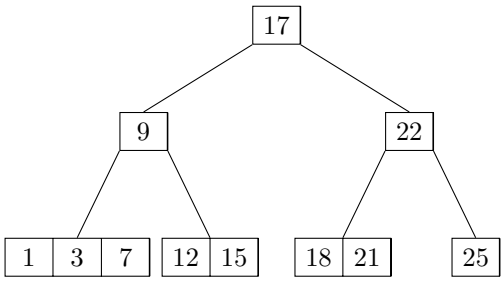
1 Punkt

KEIN Punktabzug, wenn Blatt in (46) und (49,50) aufgespalten wurde.

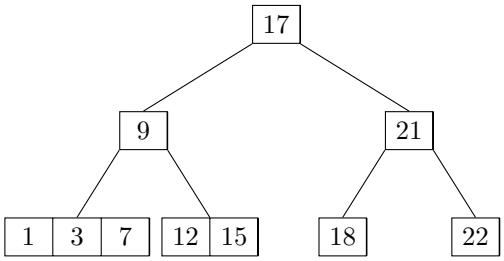
4.2 (10 Punkte) Löschen Sie nacheinander die Elemente 25, 9 und 18 aus Baum (b). Zeichnen Sie wiederum jeweils den Zustand des Baumes nach jeder Löschung und beschreiben Sie die nötigen Zwischenschritte, um die Bedingungen an den Baum aufrecht zu erhalten.

4.2 (10 points) Delete the elements 25, 9 and 18 from tree (b), one after another. Again, draw the state of the tree after each single deletion and describe the required steps to maintain the tree conditions.

(b)

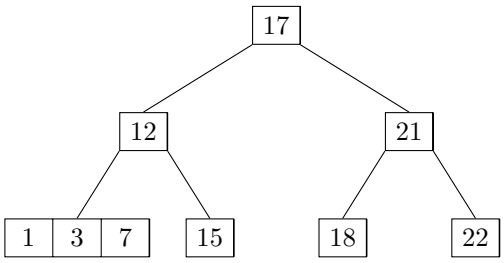


1. Delete element **25**; which is located in a leaf node. This leaf has now too few elements; steal element 21 from left sibling node. 1 Punkt



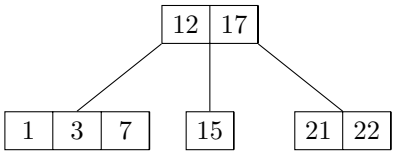
1 Punkt

2. Delete element **9**, which is located in an inner node; search for the successor in the subtree to the right (12); replace element **9** by element 12 and remove element 12 from leaf. je 1 Punkt (ins. 2 Punkte)



1 Punkt
KEIN Punktabzug, wenn (9) durch (7) ersetzt wurde.

3. Delete element **18**, which is located in a leaf; we cannot steal an element from the sibling node, because it has too few elements; merge the leaf with its sibling to (21,22); its parent node now has zero elements; merge it with its sibling to (12,17); The root node has now no more elements; the height of the tree decreases by 1. je 1 Punkt (ins. 4 Punkte)



1 Punkt

Aufgabe 5 (10 Punkte)

Gegeben sei die unten angegebene Klasse `Node` (in Pseudocode), die einen Knoten in einem binären Suchbaum repräsentiert. Seien `leftChild` und `rightChild` Pointer (in C++), bzw. Referenzen (in Java) auf den linken bzw. rechten Kindknoten. Wenn der entsprechende Kindknoten nicht existiert, ist der Pointer (bzw. die Referenz) `null`.

Schreiben Sie (in Java oder C++) eine rekursive Methode `printAllKeys(Node p)`, die für einen binären Suchbaum mit dem Wurzelknoten `p` alle Schlüssel (keys) in absteigender Reihenfolge ausgibt.

Given the class `Node` below (in pseudocode), that represents a node in a binary search tree. Let `leftChild` and `rightChild` be pointers (in C++), respectively references (in Java) to the left respectively to the right child node. Such a pointer (reference) is `null`, if the related child node doesn't exist.

Write down (in Java or C++) a recursive method `printAllKeys(Node p)`, that returns all keys of the binary search tree with the root node `p` in descending order.

```
public class Node {
  int key;
  Node leftChild;
  Node rightChild;
}
```

C++:

```
void printAllKeys(Node* p) {
  if (p == NULL) return;
  printAllKeys(p->rightChild);
  std::cout << p->key << std::endl;
  printAllKeys(p->leftChild);
}
```

Java:

```
void printAllKeys(Node p) {
  if (p == null) return;
  printAllKeys(p.rightChild);
  System.out.println(p.key);
  printAllKeys(p.leftChild);
}
```

Abbruchkriterium korrekt: 2 Punkte
Rekursiver Aufruf für linken Teilbaum korrekt: 2 Punkte
Print-Anweisung korrekt: 2 Punkte
Rekursiver Aufruf für rechten Teilbaum korrekt: 2 Punkte
Ausgabe in korrekter Reihenfolge: 2 Punkte

Aufgabe 6 (10 Punkte)

Gegeben sei ein beliebiger gerichteter ungewichteter Graph mit Startknoten S. Alle Knoten sind von S erreichbar. Beantworten Sie folgende Fragen knapp aber präzise:

6.1 (3 Punkte) Welche Eigenschaften muss eine Kante jeweils aufweisen, um bei einer **Breitensuche** ausgehend von S eine Vorwärts-, Quer-, bzw. Rückwärtskante zu werden?

6.2 (2 Punkte) Hängt bei einer **Breitensuche** der Typ einer Kante von der Reihenfolge ab, in der die ausgehenden Kanten eines Knotens abgearbeitet werden?

6.3 (2 Punkte) Welche Eigenschaften muss eine Kante aufweisen, um bei einer **Tiefensuche** ausgehend von S eine Vorwärts- bzw. Rückwärtskante zu werden?

6.4 (3 Punkte) Hängt bei einer **Tiefensuche** der Typ einer Kante von der Reihenfolge ab, in der die ausgehenden Kanten eines Knotens abgearbeitet werden?

Given an arbitrary directed unweighted graph with start node S. All nodes are reachable from S. For each of the following tasks, give a short but precise answer:

6.1 (3 points) On **breadth first search** starting at node S, which characteristics must an edge have in order to be a forward edge, a cross edge resp. a back edge?

6.2 (2 points) On **breadth first search**, does the type of an edge depend on the order in which the outcoming edges of a node are processed?

6.3 (2 points) On **depth first search** starting at node S, which characteristics must an edge have in order to be a forward edge resp. a back edge?

6.4 (3 points) On **depth first search**, does the type of an edge depend on the order in which the outcoming edges of a node are processed?

6.1 Bei der Breitensuche bekommt jeder Knoten einen eindeutigen Level (kürzester Abstand zum Startknoten) zugewiesen:

- Vorwärtskante: Startknotenlevel $<$ Zielknotenlevel 1 Punkt
- Querkante: Startknotenlevel $==$ Zielknotenlevel 1 Punkt
- Rückwärtskante: Startknotenlevel $>$ Zielknotenlevel 1 Punkt

6.2 Die Zuordnung ist eindeutig, da der Level jedes Knotens eindeutig definiert ist. Alle von Level i erreichbaren Knoten, die noch nicht gelabelt sind, erhalten Level $i + 1$. Die Reihenfolge, in der

die ausgehenden Kanten abgearbeitet werden, spielt dabei keine Rolle.

Aussage korrekt: 2 Punkte
Begründung korrekt: 1 Zusatzpunkt

6.3

- Vorwärtskante: Der Startknoten muss auf einem Pfad erreichbar sein, der nicht über den Zielknoten läuft. 1 Punkt
- Rückwärtskante: Der Zielknoten muss auf einem Pfad erreichbar sein, der nicht über den Startknoten läuft. 1 Punkt

6.4 Die Zuordnung ist nicht eindeutig, sobald es einen Weg zum Zielknoten gibt, der nicht über den Startknoten führt. Je nachdem, welche Pfade zuerst abgearbeitet werden, wird dann der Zielknoten vor oder nach dem Startknoten erreicht. Bei einem Baum wäre die Zuordnung eindeutig (nur Vorwärtskanten).

Aussage korrekt: 3 Punkte
Begründung korrekt: 1 Zusatzpunkt

Aufgabe 7 (22 Punkte)

Prüfen Sie, ob auf folgenden Rekursionsgleichungen das Mastertheorem angewandt werden kann und bestimmen Sie jeweils bei Anwendbarkeit die Laufzeit von $T(n)$.

Check, if the master theorem can be applied to the following equations. If so, determine the runtime of $T(n)$.

7.1

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

7.2

$$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ T\left(n-1\right) + n & \text{otherwise} \end{cases}$$

7.3

$$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ 2T\left(\frac{n}{2}\right) + \frac{n}{\log n} & \text{otherwise} \end{cases}$$

7.1 4 Punkte

$$a = 3, b = 3, f(n) = \sqrt{n} \quad \text{1 Punkt}$$

Bedingung für Fall 1:

$$f(n) \in O(n^{\log_b a - \epsilon}), \epsilon > 0$$

Prüfe, ob Fall 1 anwendbar:

$$\sqrt{n} \in O(n^{\log_3 3 - \epsilon}), \epsilon > 0$$
$$n^{\frac{1}{2}} \in O(n^{1-\epsilon}), \epsilon > 0 \quad \text{1 Punkt}$$

Gilt für $\epsilon = \frac{1}{2}$, d.h. Fall 1 ist anwendbar. 1 Punkt

Laufzeitkomplexität berechnen:

$$T(n) \in \Theta(n^{log_b a})$$
$$T(n) \in \Theta(n^{log_3 3})$$
$$T(n) \in \Theta(n) \quad \text{1 Punkt}$$

7.2 2 Punkte

Mastertheorem nicht anwendbar, da $T(n-1) + n$ nicht der Form $a \cdot T(\frac{n}{b}) + f(n)$ entspricht.

Aussage korrekt: 1 Punkt
Begründung korrekt: 1 Punkte
Wenn Begründung fehlt: insgesamt 0 Punkte.

7.3 16 Punkte

$$a = 2, b = 2, f(n) = \frac{n}{\log n} \quad \text{1 Punkt}$$

Bedingung für Fall 1:

$$f(n) \in O(n^{\log_b a - \epsilon}), \epsilon > 0$$

Prüfe, ob Fall 1 anwendbar:

$$\frac{n}{\log n} \in O(n^{\log_2 2 - \epsilon}), \epsilon > 0$$
$$\frac{n}{\log n} \in O(n^{1-\epsilon}), \epsilon > 0$$
$$\frac{n}{\log n} \in O(\frac{n^1}{n^\epsilon}), \epsilon > 0$$
$$\frac{1}{\log n} \in O(\frac{1}{n^\epsilon}), \epsilon > 0$$

Prüfe mithilfe des Grenzwertsatzes, ob $\frac{1}{\log n} \in O(\frac{1}{n^\epsilon})$:

$$\begin{aligned}
\frac{1}{\log n} \in O(\frac{1}{n^\epsilon}) &\Leftrightarrow \lim_{n\to\infty} \frac{\frac{1}{\log n}}{\frac{1}{n^\epsilon}} < \infty \\
&\Leftrightarrow \lim_{n\to\infty} \frac{n^\epsilon}{\log n} < \infty \\
&\Leftrightarrow \lim_{n\to\infty} \frac{\epsilon \cdot n^{\epsilon-1}}{\frac{1}{n \cdot \ln 2}} < \infty \quad \text{(mit L'Hopital)} \\
&\Leftrightarrow \lim_{n\to\infty} \underbrace{\epsilon \cdot n^{\epsilon-1} \cdot n \cdot \ln 2}_{\to\infty} < \infty \; \lightning \quad \text{4 Punkte}
\end{aligned}$$

Das heißt, dass $\frac{1}{\log n} \notin O(\frac{1}{n^\epsilon})$ und somit ist Fall 1 nicht anwendbar. 1 Punkt

Bedingung für Fall 2:

$$f(n) \in \Theta(n^{\log_b a})$$

Prüfe, ob Fall 2 anwendbar:

$$\frac{n}{\log n} \in \Theta(n^{\log_2 2})$$
$$\frac{n}{\log n} \in \Theta(n)$$

Gilt nicht, da $\frac{n}{\log n} \notin \Omega(n)$. 1 Punkt

Beweis:

$$\begin{aligned}
\frac{n}{\log n} \in \Omega(n) &\Leftrightarrow \lim_{n\to\infty} \frac{\frac{n}{\log n}}{n} > 0 \\
&\Leftrightarrow \lim_{n\to\infty} \frac{n}{n \log n} > 0 \\
&\Leftrightarrow \lim_{n\to\infty} \underbrace{\frac{1}{\log n}}_{\to 0} > 0 \; \lightning
\end{aligned}$$

4 Punkte

Bedingung für Fall 3:

$$f(n) \in \Omega(n^{\log_b a+\epsilon}), \epsilon > 0 \text{ und } a \cdot f(\frac{n}{b}) \leq c \cdot f(n), \quad 0 < c < 1, n > n_0$$

Prüfe, ob Fall 3 anwendbar. Prüfe zunächst die Nebenbedingung $a \cdot f(\frac{n}{b}) \leq c \cdot f(n)$:

$$\begin{aligned}
2 \cdot \frac{\frac{n}{2}}{\log \frac{n}{2}} &\leq c \cdot \frac{n}{\log n} \\
2 \cdot \frac{\frac{1}{2}}{\log \frac{n}{2}} &\leq c \cdot \frac{1}{\log n} \\
\frac{1}{\log n - \log 2} &\leq c \cdot \frac{1}{\log n} \\
\log n - \log 2 &\geq \frac{\log n}{c} \\
\underbrace{1 - \frac{\log 2}{\log n}}_{<1} &\geq \underbrace{\frac{1}{c}}_{>1} \; \lightning
\end{aligned}$$

4 Punkte

Also ist auch Fall 3 nicht anwendbar und somit das Mastertheorem nicht anwendbar. 1 Punkt