AUD Klausurblatt SoSe 2020 | Janson/Fischlin

Definitionen/Wissen

Sortier-Algorithmen in einem Satz

- *InsertionSort*: Sortierthalten der linken Teilfolge, neuen Wert an richtige Position einfügen
- *BubbleSort*: Vergleiche Paare von benachbarten Schlüsselwerten
- *SelectionSort*: Wähle kleinstes Element und tausche es nach vorne
- *MergeSort*: Teilen, sortiertes Zurückschreiben in Array
- *QuickSort*: Vergleich der Werte mithilfe PivotElement, rekursiver Aufruf auf Teilarray

Asymptotik

- $\frac{f(n)}{g(n)} = 0 \Rightarrow f(n) = o(g(n))$
- $\frac{f(n)}{g(n)} : konvergent \Rightarrow f(n) = O(g(n))$
- $o(n) \in O(n) \Rightarrow$ schließt alle anderen aus
- $f(n) = g(n) \Rightarrow f(n) = \Theta(g(n))$

Master-Theorem

- *Anwendbar?*
  - $a \geq 1$ und konstant?
  - $b > 1$ und konstant?
  - $f(n)$ positiv?
- *Vorgehen*
  - Berechnung von $log_b(a)$
  - Vergleich mit $f(n)$
    - $f(n)$ **polynomial kleiner** als $n^{log_b(a)}$ $\Rightarrow T(n) = \Theta(n^{log_b(a)})$
      $(f(n) = O(n^{log_b(a-\epsilon)}), \epsilon > 0)$
    - $f(n)$ und $n^{log_b(a)}$ **gleiche Größe** $\Rightarrow T(n) = \Theta(n^{log_b(a)} lg(n))$
      $(f(n) = \Theta(n^{log_b(a)})$
    - $f(n)$ **polynomial größer** als $n^{log_b(a)}$ und
      $af(\frac{n}{b}) \leq c\ f(n)$, $(\epsilon < 1)$ $\Rightarrow T(n) = \Theta(f(n))$
      $(f(n) = \Omega(n^{log_b(a+\epsilon)}), \epsilon > 0)$

String-Matching

Allgemein

- durchsuchender Text: Array `T` der Länge `lenTxt`
- Textmuster: Array `P` der Länge `lenPat` $\leq$ `lenTxt`
- Gesucht: alle gültigen Verschiebungen mit denen `P` in `T` auftaucht
- Rückgabe: alle `sft` $\in \mathbb{N}$, sodass `T[sft,...,sft+lenPat-1] = P` gilt

NaiveStringMatching

Pseudocode:

NaiveStringMatching(T,P)

```
lenTxt = length(T)
lenPat = length(P)
L = empty
FOR sft = 0 TO lenTxt - lenPat DO
    isValid = TRUE
    FOR j = 0 TO lenPat - 1 DO
        IF P[j] ≠ T[sft+j] THEN
            isValid = FALSE
    IF isValid THEN
        L = append(L, sft)
RETURN L
```

Beispiel: `T=[h,e,h,e,h,h,h,e,y,h]`, `P=[h,e,h]`

sft	$T[sft, \dots, sft + lenPat - 1] \stackrel{?}{=} P$	L
0	**true**	$[0]$
1	**false**	$[0]$
2	**true**	$[0, 2]$
3	**false**	$[0, 2]$
4	**false**	$[0, 2]$
5	**false**	$[0, 2]$
6	**false**	$[0, 2]$
7	**false**	$[0, 2]$

Queue mithilfe von zwei Stacks

enqueue pusht auf den ersten Stack.
dequeue wird als $pop(S_2)$ definiert.
Falls der zweite Stack leer ist werden alle Elemente aus S_1 geholt und in S_2 überführt.
Dann wird das erste Element von S_2 ausgegeben.

new(Q)

```
S1 = new(S1)
S2 = new(S2)
Q = [S1, S2]
return Q
```

isEmpty(Q)

```
parse Q = [S1, S2]
b1 = isEmpty(S1)
b2 = isEmpty(S2)
return b1 ∧ b2
```

enqueue(Q, x)

```
parse Q = [S1, S2]
push(S1, x)
```

dequeue(Q)

```
parse Q = [S1, S2]
if isEmpty(Q) then
    return Error
if isEmpty(S2) then
    while ¬isEmpty(S1) do
        push(S2, pop(S1))
return pop(S2)
```

Stack mithilfe von zwei Queues

Eine der beiden Queues bleibt immer leer (anfangs Q_2)
push fügt Wert immer der leeren Queue hinzu.
pop holt alle Elemente bis auf das letzte aus der Queue zurück und fügt sie in die leere ein.
Das letzte Element wird dann ausgegeben.

new(S)

```
Q1 = new(Q1)
Q2 = new(Q2)
S = [Q1, Q2]
return S
```

isEmpty(S)

```
parse S = [Q1, Q2]
b1 = isEmpty(Q1)
b2 = isEmpty(Q2)
return b1 ∧ b2
```

push(S, x)

```
parse S = [Q1, Q2]
if isEmpty(Q1) then
    enqueue(Q2, x)
else
    enqueue(Q1, x)
```

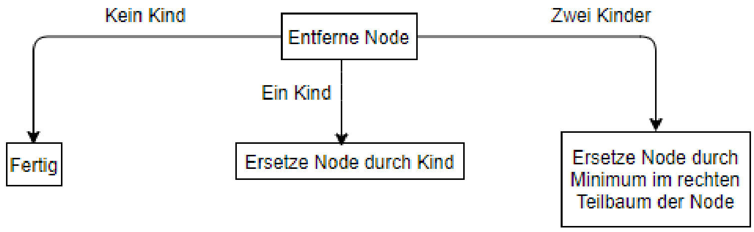
pop(S)

```
parse S = [Q1, Q2]
if isEmpty(S) then
    return Error
if isEmpty(Q2) then
    t = dequeue(Q1)
    while ¬isEmpty(Q1) do
        enqueue(Q2, t)
        t = dequeue(Q1)
else
    t = dequeue(Q2)
    while ¬isEmpty(Q2) do
        enqueue(Q1, t)
        t = dequeue(Q2)
return t
```

Baumoperationen

BST

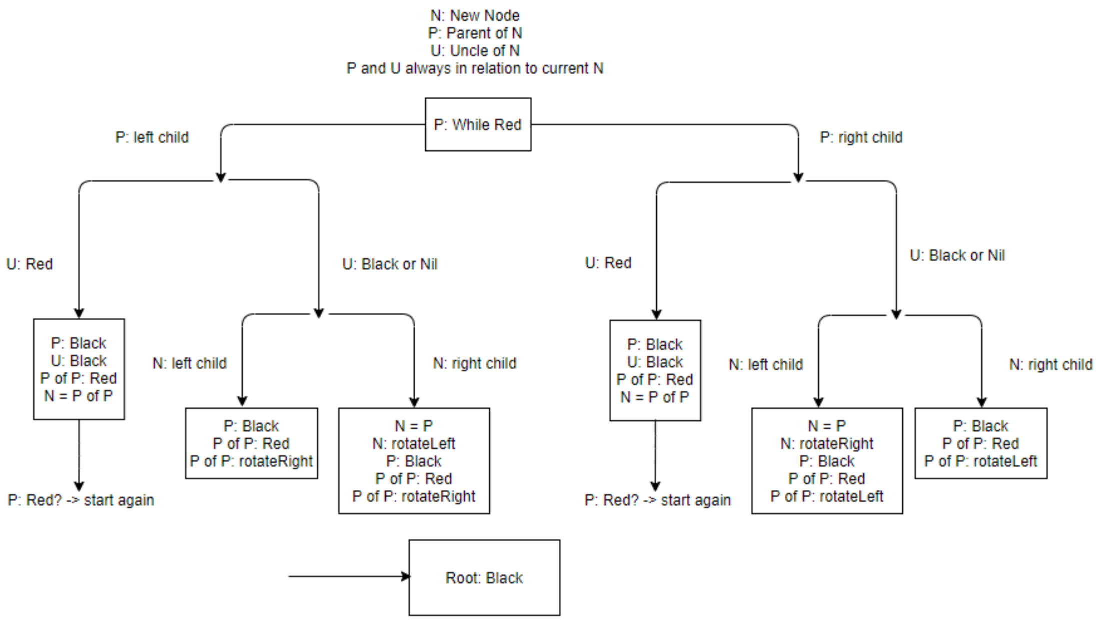
Löschen



RBT

Insert

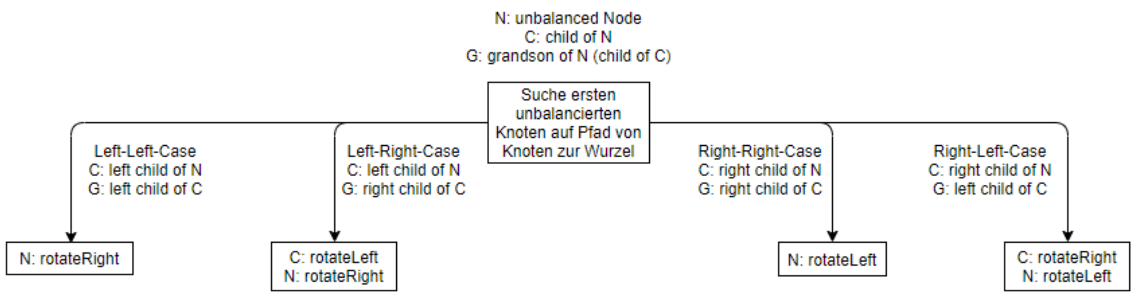
Wie im BST, neuen Knoten rot färben, danach FixUp:



AVL-Bäume

Einfügen/Löschen

Einfügen und Löschen wie beim BST, danach jeweils Rebalancieren:



Beachte beim Löschen:

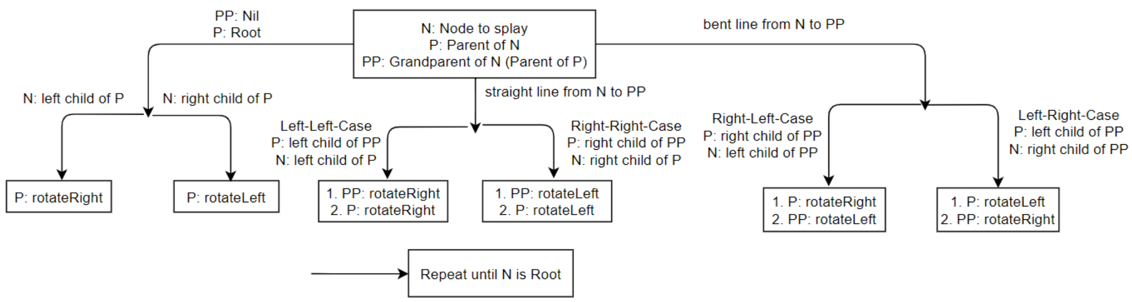
- Wahl von C: größter Teilbaum von N (eindeutig, da N unbalanciert)
- Wahl von G: größter Teilbaum von C (nicht eindeutig, Wahl Rechts-Rechts/Links-Links)
- **Potenziell** müssen mehrere Knoten bearbeitet werden ⇒ alle Knoten bis Wurzel prüfen

Splay-Bäume

- Suchen: Spüle gesuchten Knoten an die Wurzel (alternativ zuletzt besuchten Knoten)

- Einfügen: Einfügen nach BST-Regeln und danach Hochspülen des Knotens
- Löschen:
    1. zu löschenden Knoten hochspülen
    2. Knoten löschen
    - Falls nur ein Kind: Dieses Kind neue Wurzel und fertig
    - Falls zwei Kinder: Spüle größten Knoten im linken Teilbaum hoch
      Hänge danach beide Teilbäume an diesen Knoten

Spülen

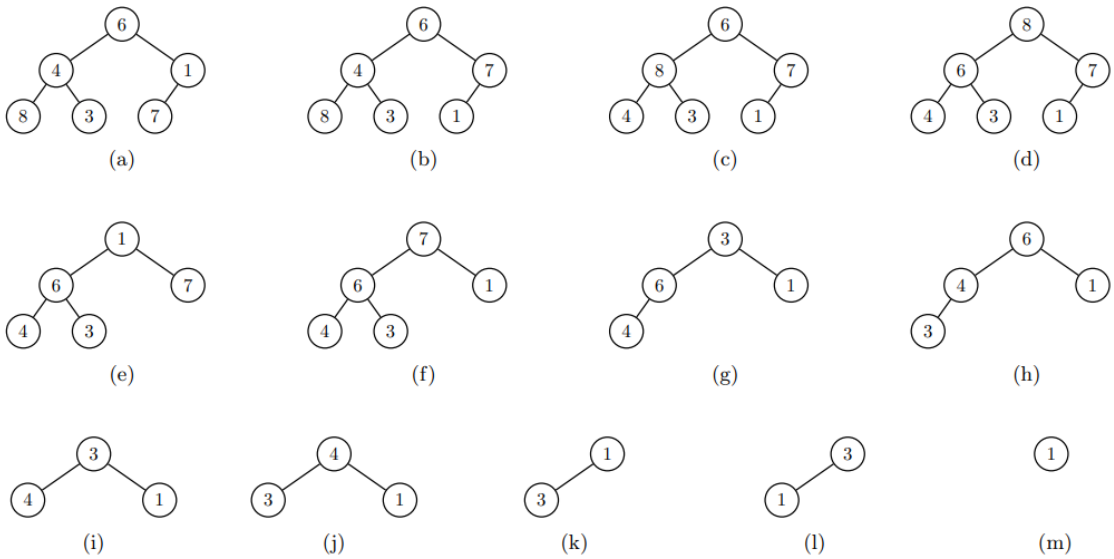


Heaps

Heap-Sort

1. Array wird als Heap aufgefasst
2. Heapeigenschaft wird wiederhergestellt (Heapify)
3. Extrahieren der Wurzel (Maximum) und Ersetzen durch "letztes" Blatt
4. Wieder Heapify um Wert an die richtige Stelle zu rücken
5. Falls der Baum noch nicht leer ist, gehe zu Schritt 3

Heapify: beginnend bei `ceil((H.length-1)/2) - 1` bis 0: vertausche nach unten, falls Parent kleiner als Child



B-Bäume

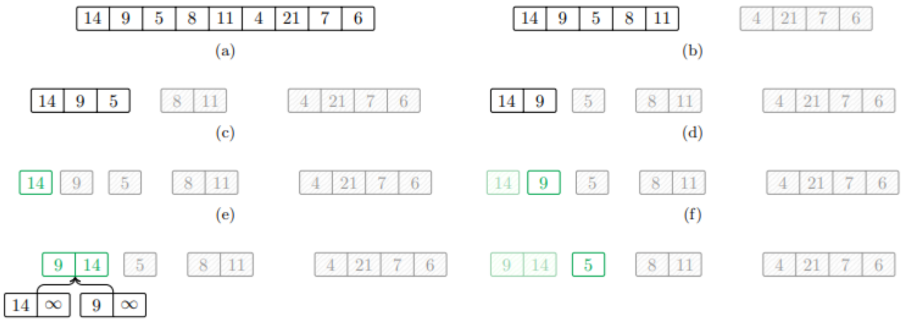
Anwendungsbeispiel

Sorting

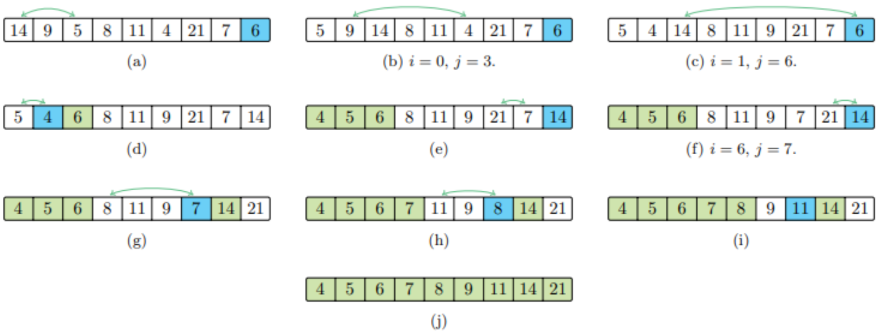
Insertion Sort

$j = 1$:	auf	Baum	Daten	Landesbibliothek	Haus	sortieren
$j = 2$:	Baum	auf	Daten	Landesbibliothek	Haus	sortieren
$j = 3$:	Daten	Baum	auf	Landesbibliothek	Haus	sortieren
$j = 4$:	Landesbibliothek	Daten	Baum	auf	Haus	sortieren
$j = 5$:	Landesbibliothek	Daten	Baum	Haus	auf	sortieren
$j = 6$:	Landesbibliothek	sortieren	Daten	Baum	Haus	auf

Merge Sort



Quick Sort



Basic Data Structures

Stacks

1:	4					
2:	4	1				
3:	4	1	3			
4:	4	1				
5:	4	1	8			
6:	4	1				

Queues

1:	3			
2:	3	4		
3:		4		
4:		4	6	
5:		4	6	7
6:	8	4	6	7
7:	8		6	7
8:	8			7

Pseudocode

Sorting

Insertion Sort(A)

```
FOR j = 1 TO A.length - 1
  key = A[j]
  // Füge A[j] in die sortierte Sequenz A[0...j-1] ein
  i = j - 1
  WHILE i >= 0 and A[i] > key
      A[i + 1] = A[i]
      i = i - 1
  A[i + 1] = key
```

BubbleSort(A)

```
FOR i = 0 TO A.length - 2
    FOR j = A.length - 1 DOWNTO i + 1
        IF A[j] < A[j-1]
            SWAP(A[j], A[j-1])
```

SelectionSort(A)

```
FOR i = 0 TO A.length - 2
    k = i
    FOR j = i + 1 TO A.length - 1
        IF A[j] < A[k]
            k = j
    SWAP(A[i], A[k])
```

Merge Sort

MergeSort(A, p, r)

```
IF p < r
    q = ⌊(p+r)/2⌋ // Teilen in 2 Teilfolgen
    MERGE-SORT(A,p,q) // Sortieren der beiden Teilfolgen
    MERGE-SORT(A,q+1,r)
    MERGE(A,p,q,r) // Vereinigung der beiden sortierten Teilfolgen
```

MERGE(A,p,q,r)

```
// Geteiltes Array an Stelle q
n_1 = q - p + 1
n_2 = r - q
Let L[0...n_1] and R[0...n_2] be new arrays
FOR i = 0 TO n_1 - 1 // Auffüllen der neu erstellten Arrays
    L[i] = A[p + i]
FOR j = 0 TO n_2 - 1
    R[j] = A[q + j + 1]
L[n_1] = ∞ // Einfügen des Sentinel-Wertes
R[n_2] = ∞
i = 0
j = 0
FOR k = p TO r  // Eintragweiser Vergleich der Elemente
    IF L[i] ≤ R[j]
        A[k] = L[i] // Sortiertes Zurückschreiben in Original-Array
        i = i + 1
    ELSE
        A[k] = R[j]
        j = j + 1
```

Quicksort

QUICKSORT(A,p,r)

```
IF p < r    // Überprüfung, ob Teilarray leer ist
    q = PARTITION(A,p,r)
    QUICKSORT(A,p,q-1)
    QUICKSORT(A,q+1,r)
```

PARTITION(A,p,r)

```
x = A[r]    // Wahl des Pivotelements
i = p - 1   // Index i setzen
FOR j = p TO r - 1 // Auffüllen des Teilarrays mit Elementen
    IF A[j] ≤ x
        i = i + 1
        SWAP(A[i], A[j]) /
SWAP(A[i+1], A[r]) // Tausch des Pivotelements
RETURN i + 1 // Neuer Index des Pivotelements
```

MasterTheorem - Beispiele

Begründen Sie für jede der folgenden Rekursionsgleichungen $T(n)$, ob Sie das Mastertheorem anwenden können oder nicht. Benutzen Sie gegebenenfalls das Mastertheorem, um eine asymptotische Schranke für $T(n)$ zu bestimmen. Die entsprechenden Anfangsbedingungenen (also die Werte $T(1)$ in allen Beispielen und, in (f), (h) und (i), zusätzlich $T(2)$) sind dabei vorgegebene Konstanten.

(a) $T(n) = 3T\left(\frac{n}{2}\right) + n^2$ (für $n > 1$);
(b) $T(n) = 4T\left(\frac{n}{2}\right) + n^2$ (für $n > 1$);
(c) $T(n) = 2^n T\left(\frac{n}{2}\right) + n^n$ (für $n > 1$);
(d) $T(n) = \frac{1}{2}T\left(\frac{n}{2}\right) + 1/n$ (für $n > 1$);
(e) $T(n) = \sqrt{2}T\left(\frac{n}{2}\right) + \log(n)$ (für $n > 1$);
(f) $T(n) = 2T\left(\frac{n}{\log(n)}\right) + n^2$ (für $n > 2$);
(g) $T(n) = 64T\left(\frac{n}{8}\right) - n^2\log(n)$ (für $n > 1$);
(h) $T(n) = 4T\left(\frac{n}{2}\right) + \frac{n}{\log(n)}$ (für $n > 2$);
(i) $T(n) = 2T\left(\frac{n}{2}\right) + \frac{n}{\log(n)}$ (für $n > 2$);
(j) $T(n) = 6T\left(\frac{n}{3}\right) + n^2\log(n)$ (für $n > 1$);
(k) $T(n) = 2T\left(\frac{4n}{3}\right) + n$ (für $n > 1$);
(l) $T(n) = T\left(\frac{n}{2}\right) + 2T\left(\frac{n}{4}\right) + n$ (für $n > 1$).

Lösung. (a) Hier können wir das Mastertheorem anwenden: Es sind $a = 3 \geq 1$ und $b = 2 > 1$ konstant, und es gilt $f(n) = n^2 \geq 0$ für alle $n \in \mathbb{N}$. Daraus folgt $\log_b(a) < 1{,}59 < 2$. Wir sind also im Fall 3 des Mastertheorems, denn $f(n) \in \Omega(n^{1,59+\varepsilon})$ für ein $\varepsilon > 0$ (z. B. mit $\varepsilon = 1/10$), vorausgesetzt, dass die Regularitätsbedingung erfüllt ist. Diese gilt aber für $c = \frac{3}{4} < 1$ und alle $n \in \mathbb{N}$ (da $3\left(\frac{n}{2}\right)^2 = \frac{3}{4}n^2$), und wir erhalten $T(n) \in \Theta(n^2)$.

(b) Hier können wir das Mastertheorem anwenden: Es sind $a = 4 \geq 1$ und $b = 2 > 1$ konstant, und es gilt $f(n) = n^2 \geq 0$ für alle $n \in \mathbb{N}$. Daraus folgt $\log_b(a) = 2$. Wir sind also im Fall 2 des Mastertheorems, denn natürlich ist $f(n) \in \Theta(n^2)$, und wir erhalten $T(n) \in \Theta(n^2\log(n))$.

(c) Hier können wir das Mastertheorem nicht anwenden, weil $a = 2^n$ nicht konstant ist.

(d) Auch hier können wir das Mastertheorem nicht anwenden, weil $a = \frac{1}{2}$ zwar konstant ist, aber $a < 1$.

(e) Hier können wir das Mastertheorem wieder anwenden: Es sind $a = \sqrt{2} \geq 1$ und $b = 2 > 1$ konstant, und es gilt $f(n) = \log(n) \geq 0$ für alle $n \in \mathbb{N}$. Daraus folgt $\log_b(a) = \frac{1}{2}$. Wir sind also im Fall 1 des Mastertheorems, denn $f(n) \in O(n^{1/2-\varepsilon})$ für ein $\varepsilon > 0$ (z. B. mit $\varepsilon = 1/10$), und wir erhalten $T(n) \in \Theta(\sqrt{n})$.

(f) Hier können wir das Mastertheorem wieder nicht anwenden, da $b = \log(n)$ nicht konstant ist.

(g) Hier können wir das Mastertheorem auch nicht anwenden, weil $f(n) = -n^2\log(n)$ nicht positiv ist.

(h) Hier können wir das Mastertheorem wieder anwenden: Es sind $a = 4 \geq 1$ und $b = 2 > 1$ konstant, und es gilt $f(n) = \frac{n}{\log(n)} \geq 0$ für alle $n \in \mathbb{N}_{\geq 2}$. Daraus folgt $\log_b(a) = 2$. Wir sind also im Fall 1 des Mastertheorems, denn $f(n) \in O(n^{2-\varepsilon})$ für ein $\varepsilon > 0$ (z. B. mit $\varepsilon = 1/2$), und wir erhalten $T(n) \in \Theta(n^2)$.

(i) Hier können wir das Mastertheorem wieder nicht anwenden, weil keine der drei Bedingungen erfüllt ist. Die Anfangsbedingungen passen zwar, aber keiner der drei Fälle trifft zu. In der Tat, es sind $a = 2 \geq 1$ und $b = 2 > 1$ konstant, und es gilt $f(n) = \frac{n}{\log(n)} \geq 0$ für alle $n \in \mathbb{N}_{\geq 2}$. Daraus folgt $\log_b(a) = 1$. Wir argumentieren jetzt, dass keiner der drei Fälle des Mastertheorems zutrifft:

- Wäre $f(n) \in O(n^{1-\varepsilon})$ für ein $\varepsilon > 0$, dann gäbe es $C > 0$ und $N_0 \in \mathbb{N}$ sodass, für alle $n \geq N_0$, $\frac{n}{\log(n)} \leq Cn^{1-\varepsilon}$. Daraus würde $\frac{n^\varepsilon}{\log(n)} \leq C$ für alle $n \geq N_0$ folgen, was unmöglich ist, da $\lim_{n\to+\infty} \frac{n^\varepsilon}{\log(n)} = +\infty$. Somit gilt, für jedes $\varepsilon > 0$, $f(n) \notin O(n^{1-\varepsilon})$, und Fall 1 des Mastertheorems trifft nicht zu.
- Wäre $f(n) \in \Theta(n)$, dann gäbe es $C > 0$ und $N_0 \in \mathbb{N}$ sodass, für alle $n \geq N_0$, $Cn \leq \frac{n}{\log(n)}$, also $C\log(n) \leq 1$ für alle $n \geq N_0$. Das ist auch unmöglich, denn $\lim_{n\to+\infty} C\log(n) = +\infty$. Somit gilt $f(n) \notin \Theta(n)$, und Fall 2 des Mastertheorem trifft auch nicht zu.
- Wäre $f(n) \in \Omega(n^{1+\varepsilon})$ für ein $\varepsilon > 0$, dann gäbe es $C > 0$ und $N_0 \in \mathbb{N}$ sodass, für alle $n \geq N_0$, $Cn^{1+\varepsilon} \leq \frac{n}{\log(n)}$, also $Cn^\varepsilon\log(n) \leq 1$ für alle $n \geq N_0$. Das kann auch nicht sein, da wieder $\lim_{n\to+\infty} Cn^\varepsilon\log(n) = +\infty$. Somit gilt, für jedes $\varepsilon > 0$, $f(n) \notin \Omega(n^{1+\varepsilon})$, und Fall 3 des Mastertheorems trifft ebenfalls nicht zu.

(j) Hier können wir das Mastertheorem wieder anwenden: Es sind $a = 6 \geq 1$ und $b = 3 > 1$ konstant, und es gilt $f(n) = n^2\log(n) \geq 0$ für alle $n \in \mathbb{N}$. Daraus folgt $\log_b(a) < 1{,}64 < 2$. Wir sind also im Fall 3 des Mastertheorems, denn $f(n) \in \Omega(n^{1,64+\varepsilon})$ für ein $\varepsilon > 0$ (z. B. mit $\varepsilon = 1/10$), vorausgesetzt, dass die Regularitätsbedingung erfüllt ist. Diese gilt aber für $c = \frac{2}{3} < 1$ und alle $n \in \mathbb{N}$ (da $6\left(\frac{n}{3}\right)^2\log\left(\frac{n}{3}\right) \leq \frac{2}{3}n^2\log(n)$), und wir erhalten $T(n) \in \Theta(n^2\log(n))$.

(k) Hier können wir das Mastertheorem wieder nicht anwenden, da $b = \frac{3}{4} < 1$.

(l) Auch hier können wir das Mastertheorem nicht anwenden, da die Rekursionsgleichung nicht die Form hat, die vom Mastertheorem abgedeckt wird. □