Algorithmen & Datenstrukturen 2020

Aufgabe 1: Komplexität

a)

Bestimmen Sie die Komplexität der folgenden Funktion in Big-O Notation und beweisen Sie Ihre Behauptung:

Determine the complexity of the following function in Big-O notation and prove your claim:

$$f(n) = \sin n! + 12n \log_2 n + n + 24$$

b)

Betrachten Sie das folgende Code-Fragment:

The following code fragment is given:

```
for (int i = 0; i < N; i++) {
   arr[i] = i + 2;
}

for (int i = 0; i < N - 15; i++) {
   int j = N - 15;
   while (j > 0) {
      arr[i] = arr[j] + 5;
      j = j / 2;
   }
}
```

Welcher Big-O Komplexitätsklasse gehört die Anzahl an Arrayzugriffen dieses Codefragments bezüglich der Problemgröße N an? Begründen Sie ihre Lösung.

State the Big-O complexity class of the number of array accesses in this code fragment in terms of the problem size N and justify your answer.

Aufgabe 2: Maps / Hash Tables

Fügen Sie die folgenden Werte in der gegebenen Reihenfolge in eine Hashtable mit 7 freien Plätzen ein.
Insert following values in their given order into a hashtable with 7 free spaces:

4, 2, 11, 10, 3

Die Hashfunktion ist:
The hashfunction is:
$h(i) = (2i + 3) \mod 7$

Kollisionsbehandlung findet mit doppeltem Hashing statt und verwendet die sekundäre Hashfunktion:
Collisions are handled by double hashing with the secondary hash function:
$h'(k) = 5 - (k \mod 5)$

Aufgabe 3: Suchbäume

a) Theorie

Geben Sie für jede der folgenden Aussagen an, ob sie wahr oder falsch ist.

State whether each of the following statements is true or false.

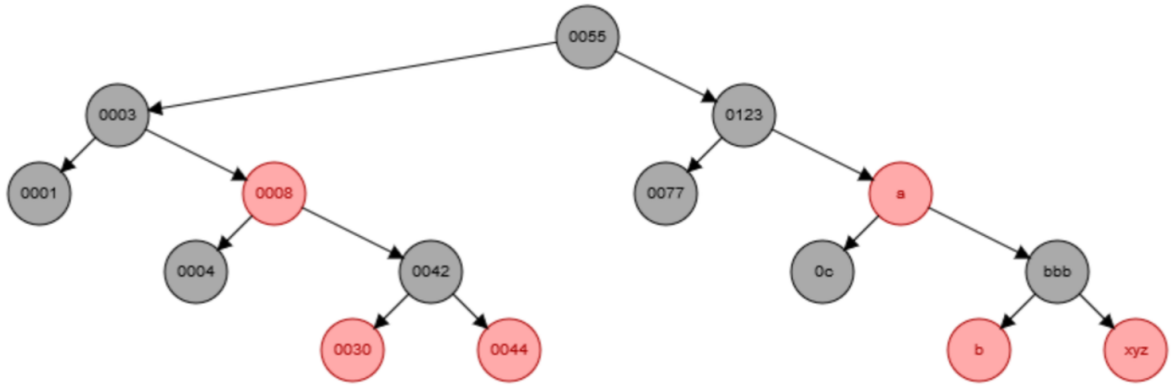
Unanswered	Right	Wrong	
☐	☐	☑	Jedem Rot-Schwarz-Baum ist ein eindeutiger 2-4-Baum zugeordnet, und umgekehrt. (Any red-black tree has a unique 2-4 tree associated with it, and vice versa.)
☐	☐	☑	Das Einfügen in einen Rot-Schwarz-Baum kann bis zu Ω(n) Umfärbungen erfordern. (insertion into a red-black tree may require up to Ω(n) recolourings.)
☐	☑	☐	Es ist möglich, einen AVL-Baum der Höhe 4 mit 15 Knoten zu erstellen. (It is possible to construct an AVL tree of height 4 with 15 nodes.)
☐	☑	☐	AVL-Bäume und Rot-Schwarz-Bäume sind selbstausgleichende binäre Suchbäume. (AVL trees and red-black trees are self-balancing binary search trees.)
☐	☑	☐	Jeder AVL-Suchbaum ist auch ein Mehrweg-Suchbaum. (Each AVL search tree is also a multiway search tree.)
☐	☐	☑	Bei einer Präorder-Traversierung eines binären Suchbaums ist das erste besuchte Element immer das kleinste. (In a preorder traversal of a binary search tree, the first item visited is always the smallest.)

b) Rot-Schwarz-Bäume

Betrachten Sie den folgenden Baum, der einen lexikalischen Komparator nutzt, wobei Ziffern vor Buchstaben sortiert werden. Leere Sentinel-Blätter sind hier nicht gezeigt, aber Sie können von ihrer Existenz ausgehen. Fügen Sie in diesen Baum die neuen Elemente 0009, 0a und 0b ein, nacheinander, in dieser Reihenfolge.

Consider the following tree that uses a lexical comparator with digits preceding letters. Empty sentinel nodes are not shown but you can assume their existence.

Insert into this tree the new elements 0009, 0a, and 0b, one after another, in this order.



Geben Sie die Anzahl der Knotenumfärbungen an, die zum Einfügen des Elements 0009 (nicht mitgezählt) erforderlich sind:
State the number of node recolorings required for the insertion of the element 0009 (not counted):

5

Geben Sie die Anzahl der Knotenumfärbungen an, die zum Einfügen des Elements 0a (nicht mitgezählt) erforderlich sind:
State the number of node recolorings required for the insertion of the element 0a (not counted):

0

Geben Sie die Anzahl der Knotenumfärbungen an, die zum Einfügen des Elements 0b (nicht mitgezählt) erforderlich sind:
State the number of node recolorings required for the insertion of the element 0b (not counted):

1

Sei *T* der aus den obigen Operationen resultierende Baum ohne die leeren Sentinel-Knoten. Betrachten Sie die Knoten der untersten Ebene von *T*, d.h. die Knoten *p* mit depth(*T*, *p*) = height(*T*). Schreiben Sie die Elemente dieser Knoten in einer zusammenhängenden Zeichenkette auf, ohne Leerzeichen:

Let *T* be the tree resulting from the above operations without the empty sentinel nodes. Consider the nodes within the bottom level of *T*, i.e., the nodes *p* with depth(*T*, *p*) = height(*T*). Write down the elements of these nodes in one contiguous character sequence, without spaces:

00090a0cbxyz

c) Binärer Suchbaum

Angenommen, Sie fügen die Werte 42, 33, 61, 54 und 19 in einen binären Suchbaum ein.
Dann besucht eine Inorder-Traversierung die resultierenden Knoten in dieser Reihenfolge:

Suppose you insert the values 42, 33, 61, 54, and 19 into a binary search tree.
Then, an in-order traversal will visit the resulting nodes in the following order:

- 42, 33, 19, 61, 54
- 19, 33, 42, 61, 54
- 19, 33, 54, 61, 42
- 19, 33, 42, 54, 61

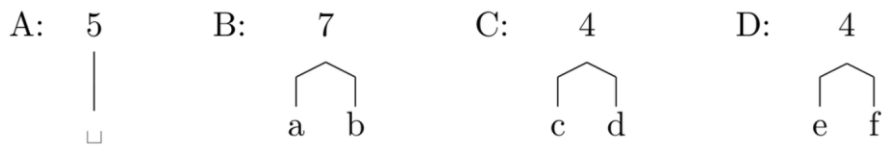
Aufgabe 4: Greedy / Divide & Conquer

a) Greedy 1

Betrachen Sie die folgenden Zeichenhäufigkeiten:
Consider the following character frequencies:

Character	␣	a	b	c	d	e	f
Frequency	5	5	2	2	2	2	2

Zeichnen Sie einen Baum nach dem Algorithmus zum Erstellen von Huffman-Bäumen aus der Vorlesung. Beginnen Sie mit dem folgenden Zwischenergebnis (es wird der Inhalt der Vorrangwarteschlange bei demjenigen Zwischenschritt gezeigt, von dem aus Sie die Prozedur fortsetzen; beginnen Sie nicht am Anfang).
Construct a tree using the algorithm for constructing Huffman coding trees presented in the lecture starting from the following intermediate state (shown are the contents of the priority queue at the intermediate state from where you are to continue the procedure; do not start the algorithm from the beginning).



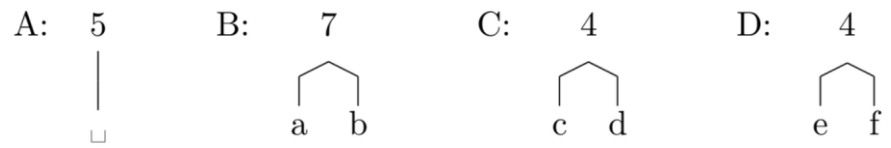
Umkreisen Sie den Unterbaum, der nach dem ersten Schritt produziert wurde, in blau und den Unterbaum, der im zweiten Schritt produziert wurde, in rot. Kreisen Sie keine weiteren Unterbäume ein.
Falls in der Lösung einer der obigen Unterbäume vorkommt, können Sie stattdessen das entsprechende Label (A, B, C oder D) verwenden.
In your solution, encircle the subtree that was produced in the first step in blue and the subtree that was produced in the second step in red. Do not encircle any further subtrees.
You can refer to the given subtrees by their label (A, B, C or D) in your solution and do not have to reproduce them if they match exactly.

b) Greedy 2

Der Huffman-Baum, den Sie in der vorigen Aufgabe erzeugt haben, ist nicht optimal. In der Vorlesung wurde aber gesagt, dass der gegebene gierige Algorithmus am Ende eine global optimale Lösung erzeugt. Was ist falsch gelaufen?
The Huffman coding tree produced in the previous task is not optimal. However, in the lecture it was stated that the greedy approach of Huffman coding tree construction yields an optimal tree. What went wrong?

Zur Erinnerung werden hier die Zeichenhäufigkeiten und der gegebene Startzustand erneut gezeigt:
As a reminder, the frequencies and starting configuration are reproduced here:

Character	␣	a	b	c	d	e	f
Frequency	5	5	2	2	2	2	2



c) Divide & Conquer

Betrachten Sie das Problem, dasjenige Element einer gegebenen Liste zu finden, das einem weiteren gegebenen Wert am nächsten liegt. Zum Beispiel, bei der gegebenen Liste L = [30, 40, 50] und dem gesuchten Wert x = 42 wäre die Lösung das Listenelement 40.
Consider the problem of finding the element in a list that is closest to a requested value. For example, given the list L = [30, 40, 50] and the requested value x = 42, we want to identify the list element 40.

Gegeben ist ein unvollständer Algorithmus, der dieses Problem mit einem Teile-und-Herrsche-Ansatz lösen soll. Füllen Sie die Leerstellen aus.
Below is an incomplete algorithm to solve this problem using divide-and-conquer in pseudocode. Fill in the gaps.

```
Algorithm closest(L, x)
    Input: a list L (with at least one element) and the requested value x
    Output: the element in L that is closest to x
    if L.length > 1 then
        first:=




        second:=




        if abs(first - x) < abs(second - x) then
            return first
        else
            return second
    else
        return L[0]
```

Was könnte ein Vorteil dieser Lösung gegenüber einer anderen Lösung sein, die einfach linear durch die Liste sucht?
What could be an advantage of this solution compared to one that just linearly scans through the list?

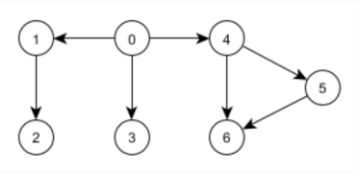
Gibt es auch Nachteile (wenn ja, welche)?
Are there any drawbacks (if so, state them)?

Was ist die Zeitkomplexität des gegebenen Ansatzes? Begründen Sie Ihre Antwort präzise, und argumentieren Sie über die Anzahl der rekursiven Aufrufe. Sie müssen nicht alle Leerstellen ausfüllen, aber können sie verwenden, um Ihre Lösung zu strukturieren.
What is the time complexity of this approach? Give precise justification including reasoning about the number of recursive calls. You do not have to use all the fields but can use them to structure your solution.

Aufgabe 5: Graphs

a) Struktur

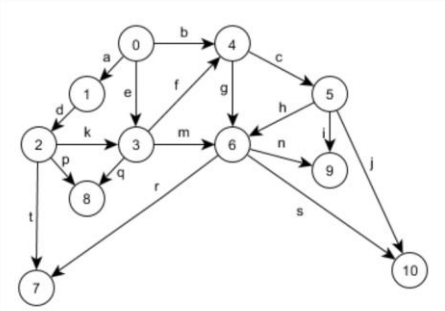
Füllen Sie die Adjazenzmatrix des folgenden Graphen aus.
Give the adjacency matrix representation of this graph.



	0	1	2	3	4	5	6
0	☐	☑	☐	☑	☑	☐	☐
1	☐	☐	☑	☐	☐	☐	☐
2	☐	☐	☐	☐	☐	☐	☐
3	☐	☐	☐	☐	☐	☐	☐
4	☐	☐	☐	☐	☐	☑	☑
5	☐	☐	☐	☐	☐	☐	☑
6	☐	☐	☐	☐	☐	☐	☐

b) Tiefensuche

Betrachten Sie den folgenden Graphen:
Consider the following graph:



Führen Sie eine Tiefensuche durch. Starten Sie beim Knoten 0 und besuchen Sie Knoten in absteigender Reihenfolge. Geben Sie die Reihenfolge an, in der die Kanten besucht werden:
For a depth-first search starting at node 0 and visiting the nodes in descending order, give the order in which the edges are visited:

Aufgabe 6: Application Problem

Jemand hat Ihren Schreibtisch durcheinandergebracht, alle Ihre Kugelschreiber zerlegt und einen Haufen Kappen, Minen und Stiftgehäuse hinterlassen. Sie wissen, dass jeder Stift aus Gehäuse, Mine und Kappe zusammengesetzt ist, die jeweils genau zueinander und nur zueinander passen. Unglücklicherweise sehen alle Gehäuse genau gleich aus, und ebenso jeweils die Minen und Kappen. Der einzige Weg, herauszufinden, ob eine Kappe auf ein Gehäuse passt, ist, es auszuprobieren. Hierdurch bekommen Sie die Information, dass die Kappe für das Gehäuse entweder (a) zu groß oder (b) zu klein ist, oder dass sie (c) genau passt. Auf dieselbe Weise können Sie probieren, ob eine Mine zu einem Gehäuse gehört, aber es gibt keine Möglichkeit, Kappen und Minen direkt miteinander zu vergleichen. Beschreiben Sie einen Algorithmus, der alle Stifte in $O(n \log n)$ erwarteter Zeit zusammenbaut.

Beschreiben Sie Ihren Algorithmus in päziser Sprache. Beziehen Sie sich auf ADT, Datenstrukturen, oder Algorithmen, die in der LV besprochen wurden, und nutzen Sie Pseudocode, wo es angemessen ist. Schreiben Sie kein Java oder andere reale Programmiersprachen.

Somebody made a mess of your desk and took all your n ballpoint pens apart, leaving behind an unsorted pile of caps, refills and barrels. You know that each pen is assembled from a barrel, a refill and a cap that exactly fit each other and no other. Unfortunately, all the barrels look the same, and so do all the caps and all the refills, respectively. The only way to see if a cap fits a given barrel is to try it out. In doing so, the only information you receive is that (a) the cap is too small to fit the barrel, (b) the cap is too large, or (c) the cap fits. In the same way you can check whether a refill fits a barrel, but there is no way to check directly whether a given cap and a given refill belong together. Describe an algorithm that reassembles all pens in $O(n \log n)$ expected time.

Describe your algorithm in precise language, referring to ADT, data structures, or algorithms discussed in class and using pseudo-code where appropriate. Do not write Java or any other real programming language.