Dies Aufgaben dieser *Klausur* stammen aus den Altklausuren und Gedächtnisprotokollen der folgenden Daten:

- 12.2.2018
- 5.3.2018
- 7.2.2019
- 24.2.2020

Aufgabe 1: Laufzeitkomplexität Geben Sie für folgende Algorithmen die asymptotische Laufzeitkomplexität in $\mathcal{O}$-Notation an:

(a) Dijkstra

(b) Quicksort

(c) Tiefensuche

(d) Mergesort

(e) Kruskal

Aufgabe 2a: $\mathcal{O}$-Notation Geben Sie für die folgenden Funktionen möglichst einfache und scharfe $\mathcal{O}$, Ω und ggf. Θ-Klassen an!

$f_1(n) = n^{2/12} + \log^{2018} n$

$f_2(n) = n^{2+(-1)^n}$

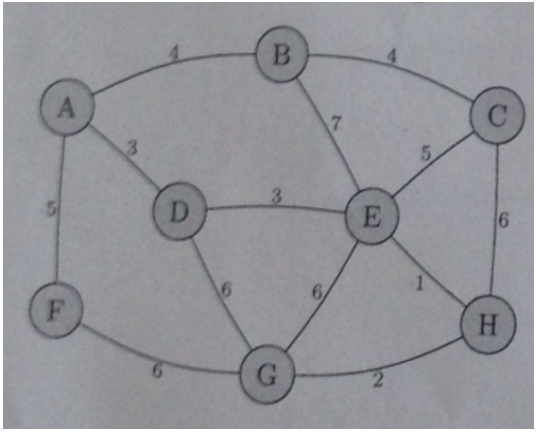
Aufgabe 2b Begründen Sie, ob für Funktionen $f_1, f_2 : \mathbb{N} \to \mathbb{R}^+$ immer $f_1 \neq \mathcal{O}(f_2) \implies f_2 = \mathcal{O}(f_1)$ gilt.

Aufgabe 2c Ordnen Sie n, $\sqrt{n}$, n^2, $n!$, 42, $n \log n$, 2^n, $8^{\log_2 n}$, $\log n$ gemäß asymptotischem Wachstum!

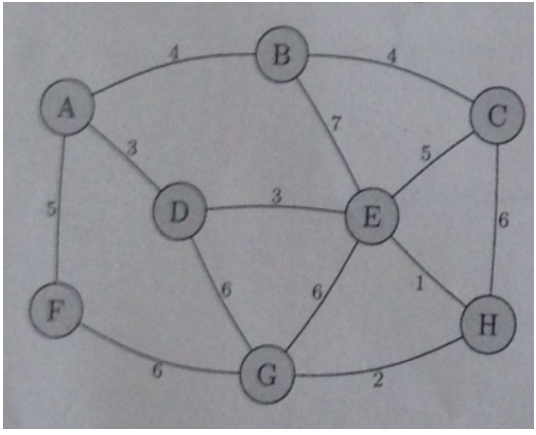
Aufgabe 2d Geben Sie eine Funktion an, die echt langsamer wächst als $n^{\log_4 16}$, aber nicht polynomiell langsamer, d.h. nicht um einen polynomiellen Faktor langsamer.

Aufgabe 2e Geben Sie für $T(n) = 2T(n/2) + n$ mit $(1) = 1$ eine scharfe asymptotische Schranke an.

Aufgabe 3a Führen Sie vom Knoten A ausgehend eine Breitensuche durch und geben Sie die besuchten Knoten entsprechend der Reihenfolge des Auffindens aus. Gehen Sie dabei davon aus, dass jeder Knoten seine Nachbarn in einer alphabetisch sortierten Liste speichert.



Aufgabe 3b Verwenden Sie Prims Algorithmus, um ausgehend vom Knoten A für den Graphen einen minimalen Spannbaum zu ermitteln. Geben Sie jeweils beim Hinzufügen eines Knotens den Inhalt der Prioritätswarteschlange (Knoten und Gewichte) in sortierter Reihenfolge an.



Aufggabe 4a Verwenden Sie Mergesort wie in der Vorlesung vorgestellt, um folgendes Array zu sortieren: | 9 | 1 | 7 | 3 | 8 | 2 | 6 | 4 |. Geben Sie dabei das gesamte Array **jeweils nach dem Merge-Schritt** an!

Aufgabe 4b Modifizieren Sie Quicksort, sodass Ihr Algorithmus für jede Eingabe nur $\mathcal{O}(n \log n)$ Zeit benötigt, und beweisen Sie, dass Ihr Algorithmus immer in $\mathcal{O}(n \log n)$ Zeit sortiert!

Aufgabe 4c Skizzieren Sie den Beweis der unteren Schranke für die Laufzeit vergleichsbasierter Sortierverfahren!

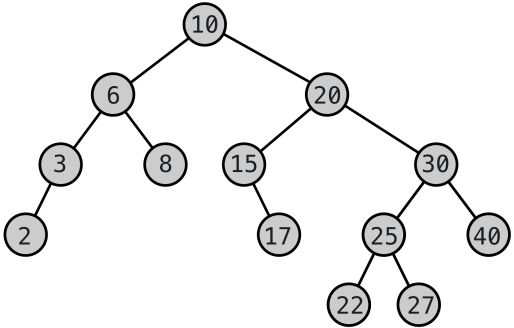
Aufgabe 5a Gegeben ist eine Hashtabelle T der Länge $m = 11$ mit Hashfunktion $h(k, i) = (k + i^2) \bmod m$. Tragen Sie die Schlüssel $56, 46, 31, 45, 42, 65, 29, 44, 23$ in der angegebenen Reihenfolge in die Hashtabele ein. Verwalten Sie Kollisionen i) durch Verkettung ($i = 0$) und ii) durch offene Adressierung.

Aufgabe 5b Beweisen Sie, dass ein Rot-Schwarz-Baum mit Schwarz-Höhe h maximal $2/3(4^h - 1)$ rote Knoten hat.

Aufgabe 5c Definieren Sie einen einfachen Algorithmus, der alle Elemente eines binären Suchbaums in sortierter Reihenfolge ausgibt. Beweisen oder widerlegen Sie: Es gibt einen vergleichsbasierten Algorithmus, der aus n Elementen in $\mathcal{O}(n)$ Zeit einenn binären Suchbaum erstellt.

Aufgabe 5d Zeigen oder widerlegen Sie: Es gibt einen Algorithmus, der in $\mathcal{O}(n + m)$ Zeit aus zwei (sortierten) binären Suchbäumen der Größen m und n einen kombinierten, balancierten Suchbaum erstellt.

Aufgabe 5e Fügen Sie in den folgenden AVL-Baum der Reihe nach folgende Werte ein: $7, 1, 31, 15, 18, 28$. Zeichnen Sie einen neuen Baum nach jeder Rotation.



Aufgabe 6a Ein *Cut-Vertex* ist ein Knoten, bei dessen Entfernung sich die Anzahl der Zusammenhangskomponenten eines ungerichteten Graphen erhöht. Spezifizieren Sie einen Algorithmus, der für einen eingegebenen Graphen in Adjazenzlistendarstellung alle Cut-Vertices in Zeit $\mathcal{O}(|V|^2 + |E||V|)$ ermittelt.

Aufgabe 6b Ein bipartiter Graph ist ein ungerichteter Graph, dessen Knoten V derart in zwei Mengen V_1, V_2 geteilt werden können, sodass es keine Kanten jeweils innerhalb dieser Mengen gibt. Geben Sie einen möglichst effizienten Algorithmus an, der entscheidet, ob ein gegebener Graph in Adjazenzlistendarstellung bipartit ist.

Aufgabe 6c Wählen Sie **entweder** Aufgabe **6a)** oder Aufgabe **6b)** und beweisen Sie die Korrektheit Ihrer Lösung.

Aufgabe 6d Ein gerichteter Graph $G = (V, E)$ heißt Turniergraph, wenn für für jedes Knotenpaar $u, v \in V$ mit $u \neq v$ entweder $(u, v) \in E$ oder $(v, u) \in E$ ist, aber nicht beides. Es gibt außerdem keine Schlaufen. Zeigen Sie, dass ein Turniergraph maximal eine Quelle und maximal eine Senke haben kann.

Aufgabe 6e Definieren Sie einen Algorithmus, der in einem Turniergraphen in Adjazenzmatrixdarstellung in $\mathcal{O}(|E|)$ Zeit die Quelle findet. Beweisen Sie die Korrektheit Ihres Algorithmus'.

Aufgabe 7a Es ist eine Folge $A = (a_1, a_2, \ldots, a_n)$ von n Elementen gegeben. Die Indizes $1 \leq i_1 < i_2 < \cdots < i_k \leq n$ definieren eine aufsteigende Teilfolge der Länge k, falls $a_{i_1} < a_{i_2} < \cdots < a_{i_k}$.
Definieren Sie rekursiv die Länge $\ell(i)$ einer längsten aufsteigenden Teilfolge, die an Position i endet.

Aufgabe 7b Spezifizieren Sie einen effizienten Algorithmus, um die Länge einer längsten aufsteigenden Teilfolge zu ermitteln und geben Sie Laufzeit und Speicherbedarf an.

Aufgabe 7c Beschreiben Sie, wie die längste aufsteigende Teilfolge ermittelt werden kann.

Aufgabe 7d Gegeben ist der folgende PERRIN-Algorithmus. Spezifizieren Sie einen effizienten Algorithmus zur Berechnung der Funktion, indem Sie das das Prinzip der dynamischen Programmierung anwenden. Bestimmen Sie die Zeit- und Platzkomplexität Ihrer Lösung. Wenn Sie es schaffen, eine Platzkomplexität von $\mathcal{O}(1)$ zu zeigen, gibt es 2 Zusatzpunkte.

PERRIN(n):

```
1: if n = 0 then
2:     return 3
3: else if n = 1 then
4:     return 0
5: else if n = 2 then
6:     return 2
7: else
8:     return PERRIN(n − 2) + PERRIN(n − 3)
9: end if
```

Aufgabe 8a Geben Sie eine formale Definition von P, NP und NP-Vollständigkeit an.

Aufgabe 8b Zeigen Sie, dass SUBGRAPH NP-vollständig ist.

$$\text{SUBGRAPH} = \{\langle G, H\rangle \mid G \text{ enthält einen zu } H \text{ isomorphen Teilgraphen.}\}$$

Aufgabe 8c Betrachten Sie die folgenden Probleme auf einem Graphen G:

$$\textsc{Vertex-Cover} = \{\langle G, n\rangle \mid \exists U \subseteq V : |U| \leq n \wedge \forall (u,v) \in E : u \in U \vee v \in U\}$$
$$\textsc{Independent-Set} = \{\langle G, n\rangle \mid \exists I \subseteq V : |I| \geq n \wedge \forall u, v \in I : u \neq v \implies (u,v) \notin E\}$$

Zeigen Sie, dass $\textsc{Vertex-Cover}$ NP-vollständig ist.