Name:	Matrikelnummer:

Prof. für Algorithmen
und Datenstrukturen
Prof. Dr. Hannah Bast
Patrick Brosi

Algorithmen und Datenstrukturen
SS 2021

http://ad-wiki.informatik.uni-freiburg.de/teaching

UNI FREIBURG

Klausur

Dienstag 24. August 2021, 14:00 - 16:30 Uhr, Gebäude 101 und 051 der Technischen Fakultät

Es gibt *fünf* Aufgaben. Für jede davon gibt es maximal 25 Punkte. Wir zählen nur die *vier besten* Aufgaben. Sie können also maximal 100 Punkte erreichen. Zum Bestehen reichen 50 Punkte.

Sie haben insgesamt 150 Minuten Zeit. Wenn Sie vier Aufgaben bearbeiten, haben Sie im Durchschnitt 30 Minuten pro Aufgabe Zeit und dann noch 30 Minuten Puffer.

An Lehrmaterialien dürfen Sie maximal **ein** nach Ihrer Wahl beschriftetes DIN-A4 Blatt Papier verwenden. Sie dürfen keinerlei elektronische Geräte verwenden, insbesondere keine Geräte, mit denen Sie mit Dritten kommunizieren oder sich mit dem Internet verbinden können.

Wir werden die Klausur am selben und am nächsten Tag korrigieren und die Noten dann gleich im HISinOne eintragen. Zum Ablauf und Termin der Klausureinsicht wird es nach der Korrektur noch eine Ankündigung auf dem Forum geben.

Bemerkung zu den Programmieraufgaben: Wann immer das Schreiben von Code gefragt ist, können Sie frei zwischen Python, Java und C++ wählen. Für kleinere, rein syntaktische Fehler, wie z.B. ein fehlendes Semikolon, gibt es keinen Punktabzug. Alle Programmieraufgaben dieser Klausur lassen sich mit wenig Code lösen, bei mehr als 12 Zeilen riskieren Sie Punktabzug.

Was Sie wie abgeben sollen: Schreiben Sie bitte oben in die blaue Box Ihren Namen und Ihre Matrikelnummer. Schreiben Sie Ihre Lösungen bitte auf die Klausur: benutzen Sie dabei für jede Aufgabe zuerst die Vorderseite und dann die Rückseite des entsprechenden Blattes. Wenn Sie zusätzliches Papier benötigen, schreiben Sie ebenfalls auf jedes dieser Blätter Ihren Namen und Ihre Matrikelnummer.

Tipps: Lesen Sie die Aufgaben sorgfältig durch und vermeiden Sie Flüchtigkeitsfehler. Verbringen Sie nicht zu viel Zeit mit einer einzelnen Teilaufgabe. Wenn Sie nicht weiterkommen, machen Sie erstmal mit einer anderen (Teil)aufgabe weiter.

Wir wünschen Ihnen viel Erfolg!

A1	A2	A3	A4	A5	Punkte	Note

Aufgabe 1 (Sortieren und O-Notation, 25 Punkte)

1.1 (10 Punkte) Betrachten Sie die folgende Funktion, die ein gegebenes Feld absteigend sortiert. Geben Sie für jede der Zeilen 2 - 5 an, wie oft die Zeile für ein Feld der Größe n ausgeführt wird. Wenn die Anzahl der Ausführungen von der Eingabe abhängt, geben sie an, wie viele Ausführungen es mindestens gibt und wie viele höchstens (mit Begründung). Eine Zeile mit einer *for* Anweisung wird dabei so oft ausgeführt, wie die Menge groß ist, über die iteriert wird.

```
1.  def max_sort(A):
2.    for i in range(len(A)):
3.      for j in range(i + 1, len(A)):
4.        if A[i] < A[j]:
5.          A[i], A[j] = A[j], A[i]
```

1.2 (5 Punkte) Jeder vergleichsbasierte Algorithmus lässt sich leicht so erweitern, dass er zu Beginn vergleichsbasiert in Zeit $O(n)$ überprüft, ob die Folge schon sortiert ist, und falls ja, nichts weiter tut. Warum widerspricht das nicht der unteren Schranke von $\Omega(n \cdot \log n)$ für vergleichsbasiertes Sortieren?

1.3 (10 Punkte) Seien f_1, f_2, g_1, g_2 Funktionen $\mathbb{N} \to \mathbb{R}_0^+$. Beweisen Sie, dass wenn $f_1 = O(g_1)$ und $f_2 = O(g_2)$, dann auch $f_1 + f_2 = O(g_1 + g_2)$. Argumentieren Sie dabei über die Definition von $O(...)$ und nicht über Grenzwerte.

Aufgabe 2 (Hashing und dynamische Felder, 25 Punkte)

2.1 (5 Punkte) Zeichnen Sie den Zustand der Hashtabelle der Größe 5 nach Einfügen der Elemente $11, 4, 21, 29, 6$ (in dieser Reihenfolge) mit der Hashfunktion $h(x) = 7 \cdot x \bmod 5$. Benutzen Sie Hashing mit offener Adressierung.

2.2 (10 Punkte) Seien H_1 und H_2 zwei c-universelle Klassen für Hashtabellen der Größe m, mit $H_1 \cap H_2 = \emptyset$. Beweisen Sie, dass dann auch $H_1 \cup H_2$ eine c-universelle Klasse für Hashtabellen der Größe m ist.

2.3 (5 Punkte) Seien bei einem dynamischen Feld s_i und c_i die Anzahl Elemente und die Kapazität nach der i-ten Operation. Zu Beginn ist $s_0 = 0$ und $c_0 = 1$. Wenn vor einer *push* Operation $s_{i-1} = c_{i-1}$, wird das Feld vergrößert, so dass nach der Operation $c_i = 2 \cdot s_i$. Wenn nach einer *pop* Operation $s_i < c_i/2$, wird das Feld verkleinert, so dass $c_i = s_i$. Geben Sie die Werte von s_i und c_i nach jeder der folgenden Operationen an: *push*, *push*, *push*, *pop*, *pop*, *push*. Die Argumente von *push* wurden weggelassen, weil sie für die Aufgabe unerheblich sind.

2.4 (5 Punkte) Sind bei der Vergrößerungs- und Verkleinerungsstrategie von Aufgabe 2.3 die Gesamtkosten für eine Folge von n beliebigen Operationen auf einem zu Beginn leeren Feld $O(n)$? Geben Sie eine kurze aber stichhaltige Begründung für Ihre Antwort.

Aufgabe 3 (Blockoperationen, Suchbäume und Heaps, 25 Punkte)

3.1 (10 Punkte) Sei σ eine Permutation der Zahlen $0, \ldots, 11$ und nehmen wir an, ein Programm greift nacheinander auf die Elemente $A[\sigma(0)]$, $A[\sigma(1)]$, ..., $A[\sigma(11)]$ eines Feldes A der Größe 12 zu. Bestimmen Sie ein σ, so dass das Programm genau *neun* Blockoperationen benötigt, mit Blockgröße $B = 4$. Mit Begründung! Nehmen Sie dabei an, dass das Feld genau an einer Blockgrenze beginnt und dass $M = B$ (d.h., in den schnellen Speicher passt genau ein Block).

3.2 (8 Punkte) In einem $(2, 4)$-Baum wurde ein Element eingefügt und es kam dabei zu einer Folge von $m \geq 1$ Aufspaltungen. Sei k die Anzahl der Knoten vom Grad 3 vor der letzten Aufspaltung und k' die Anzahl der Knoten vom Grad 3 nach der letzten Aufspaltung. Welche Möglichkeiten gibt es für den Wert von $k' - k$ und warum?

3.3 (7 Punkte) Schreiben Sie eine Funktion *check_heap_property(heap_array)*, die für einen in dem Feld *heap_array* abgespeicherten binären Heap *True* zurückgibt wenn die Heapeigenschaft erfüllt ist und *False* wenn nicht. Sie können dabei annehmen, dass das Feld für einen Heap mit n Elementen die Größe $n + 1$ hat. An Position 0 in dem Feld steht der Wert *None*.

Aufgabe 4 (Graphen und Editierdistanz, 25 Punkte)

4.1 (10 Punkte) Schreiben Sie eine Funktion *explore(adjacency_lists, s)*, die auf einem durch seine Adjazenzlisten gegebenen ungerichteten Graphen alle Knoten berechnet, die von einem Startknoten s aus erreichbar sind. Das Ergebnis sollte als Menge zurückgegeben werden (Sie können dafür ein Python *dictionary* oder *set* verwenden). Sie können annehmen, dass in *adjacency_lists* alle Kanten in beide Richtungen abgespeichert sind, ohne Kantenkosten (die für diese Aufgabe irrelevant sind).

4.2 (10 Punkte) Geben Sie einen gerichteten Graphen an mit folgenden Eigenschaften. Der Graph hat mindestens ein negatives Kantengewicht, aber keinen Zyklus. Es gibt einen Knoten s, so dass Dijkstras Algorithmus, angefangen bei s und ausgeführt bis alle Knoten gelöst sind, zu einem inkorrekten Ergebnis führt. Das heißt, am Ende entspricht mindestens ein *dist* Wert an einem Knoten nicht der Distanz des kürzesten Weges von s aus zu diesem Knoten. Mit Begründung!

4.3 (5 Punkte) Seien x und y die Zeichenketten *ALGO* und *ANGIE*. Bestimmen Sie $\mathrm{ED}(x, y)$, indem Sie die zugehörige 5×6 Tabelle ausfüllen.

Aufgabe 5 (Vermischtes, 25 Punkte)

5.1 (10 Punkte) Schreiben Sie eine Funktion *second_largest(A)*, die die zweitgrößte Zahl in dem gegebenen Feld A findet. Die Laufzeit sollte $O(n)$ sein, wobei n die Anzahl der Elemente in dem Feld ist. Sie können annehmen, dass $n \geq 2$ und dass die Zahlen in dem Feld alle verschieden sind.

5.2 (5 Punkte) Sei G ein ungerichteter Graph mit Knotenmenge V und Kantenmenge E. Beweisen Sie: wenn G ein Baum ist, dann ist $|E| = |V| - 1$.

5.3 (10 Punkte) Betrachten Sie den Aufruf *ed_recursive("aa", "bb")* der folgenden rekursiven Funktion. Genau welche Aufrufe von *ed_recursive* gibt es dabei insgesamt und in welcher Reihenfolge? Stellen Sie dabei jeden Aufruf als Paar der beiden Argumente dar, ohne die Anführungszeichen. Zum Beispiel ist der erste Aufruf (aa, bb) und der zweite Aufruf (aa, b).

```
1.  def ed_recursive(x, y):
2.    n, m = len(x), len(y)
3.    if n == 0: return m
4.    if m == 0: return n
5.    ed1 = ed_recursive(x, y[0:m-1]) + 1
6.    ed2 = ed_recursive(x[0:n-1], y) + 1
7.    ed3 = ed_recursive(x[0:n-1], y[0:m-1]) + (0 if x[n-1] == y[m-1] else 1)
8.    return min(ed1, ed2, ed3)
```