Nachname:	Vorname:	Matrikelnummer:

↑ Bitte **sehr deutlich** schreiben und BLOCKBUCHSTABEN (groß geschriebene Druckbuchstaben) verwenden ↑

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

Algorithmen und Datenstrukturen
SS 2023

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

Klausur

Dienstag 8. August 2023, 14:00 - 16:30 Uhr, in den Räumen von Gebäude 101 der Technischen Fakultät

Es gibt *fünf* Aufgaben. Für jede davon gibt es maximal 20 Punkte. Sie können also maximal 100 Punkte erreichen. Zum Bestehen reichen 50 Punkte. Sie haben insgesamt zweieinhalb Stunden Zeit. Damit haben Sie im Durchschnitt 30 Minuten Zeit pro Aufgabe. Jede Aufgabe sollte in 20 Minuten machbar sein, die restlichen 10 Minuten sind als Puffer gedacht.

An Lehrmaterialien dürfen Sie maximal **ein** DIN-A4 Blatt Papier verwenden, das beidseitig mit von Ihnen selber zusammengestellten Inhalten beschriftet oder bedruckt sein kann. 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 9. und 10. August korrigieren und die Noten direkt im Anschluss im HISinOne eintragen. Die Klausureinsicht wird am Freitag, den 11. August stattfinden. Genauere Informationen dazu werden noch auf dem Forum bekannt gegeben.

Bemerkung zu den Programmieraufgaben: Wenn Code gefragt ist, können Sie frei zwischen Python, Java und C++ wählen. Wir empfehlen Python. Für kleinere, rein syntaktische Fehler gibt es keinen Punktabzug. Alle Programmieraufgaben dieser Klausur lassen sich mit wenig Code lösen. Pro Aufgabe ist eine Obergrenze für die Anzahl der Zeilen vorgegeben.

Bemerkung zu den Multiple-Choice-Aufgaben: Bei einigen der Aufgaben wird lediglich nach dem Wahrheitswert von Aussagen gefragt. Antworten Sie dann zu jeder Aussage mit den Worten *wahr* oder *falsch* oder nichts. Ohne Antwort gibt es für die betreffende Aussage 0 Punkte. Für eine richtige Antwort gibt es 2 Punkte, für eine falsche Antwort gibt es zwei Punkte Abzug. Insgesamt können Sie für eine solche Aufgabe nicht weniger als 0 Punkte bekommen. Achten Sie darauf, dass in Ihrer Abgabe klar ist, zu welcher Aussage eine Antwort gehört.

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 des Blattes, auf dem die Aufgabe steht und dann die Rückseite des *vorherigen* Blattes. Wenn Sie zusätzliches Papier abgeben wollen (das sollte die Ausnahme sein), schreiben Sie auf jedes zusätzliche Blatt Ihren Namen und Ihre Matrikelnummer.

Wir wünschen Ihnen viel Erfolg!

A1	A2	A3	A4	A5	Punkte	Note

Aufgabe 1 (Sortieren, 20 Punkte)

1.1 (6 Punkte) Schreiben Sie eine Funktion *sort_one(lst, i)*, die eine Liste *lst*, die mit Ausnahme des Elements an Position i bereits aufsteigend sortiert ist, vollständig aufsteigend sortiert. Sei n die Länge von *lst*. Sie können annehmen, dass $n \geq 2$ und $0 \leq i < n$. Die Funktion soll in Zeit $O(n)$ laufen und maximal 10 Zeilen lang sein.

1.2 (6 Punkte) Wir betrachten Eingaben, die aus Key-Value Paaren bestehen, deren Keys alle gleich sind. Ein Sortieralgorithmus wird für solche Eingaben *stabil* genannt, wenn er die Elemente in der gleichen Reihenfolge zurückgibt, in der sie in der Eingabe stehen. Beweisen oder widerlegen Sie: *HeapSort* (mittels eines binären Heaps) ist für solche Eingaben stabil.

1.3 (8 Punkte) Geben Sie für jede der folgenden Aussagen an, ob sie *wahr* oder *falsch* ist. Eine richtige Antwort gibt 2 Punkte, eine falsche Antwort gibt 2 Punkte Abzug. Wenn sie die Antwort nicht wissen, schreiben Sie *keine Aussage* oder nichts. Sie können nicht weniger als 0 Punkte für diese Aufgabe bekommen.

1. Die Laufzeit von *MergeSort* ist für alle Eingaben der Größe n in $\Theta(n \cdot \log n)$.
2. Die Eingabegröße für *MergeSort* muss eine Zweierpotenz sein.
3. Die Anzahl der Vergleiche von *MinSort* ist abhängig von der Reihenfolge der Eingabeelemente.
4. *BogoSort* ist für jede Eingabe langsamer als *HeapSort*.

Aufgabe 2 (I/E-Sequenzen und O-Notation, 20 Punkte)

2.1 (6 Punkte) Die folgende Funktion sortiert die gegebene Liste absteigend. Geben Sie die `I/E`-Sequenz für die Eingabe $[1, 5, 3]$ an und schreiben Sie dabei *über* jedes `I` bzw. `E` die zugehörige Zeilennummer des Programms. Die *for*-Schleifen sollen Sie dabei *nicht* umschreiben, sondern wie folgt berücksichtigen: wenn eine Iteration einer *for*-Schleife ausgeführt wird, entspricht dies einem `E`, sonst einem `I`, jeweils mit der Zeilennummer der *for*-Schleife darüber.

```
1. def bubble_sort(nums: list[int]):
2.   upper = len(nums) - 1
3.   for i in range(upper):
4.     for k in range(i, upper):
5.       if nums[k] < nums[k + 1]:
6.         nums[k+1], nums[k] = nums[k], nums[k+1]
```

2.2 (6 Punkte) Geben Sie für jede der folgenden Aussagen an, ob sie *wahr* oder *falsch* ist. Eine richtige Antwort gibt 2 Punkte, eine falsche Antwort gibt 2 Punkte Abzug. Wenn Sie die Antwort nicht wissen, schreiben Sie *keine Aussage* oder nichts. Sie können insgesamt nicht weniger als 0 Punkte für diese Aufgabe bekommen.

1. $n^3 + 6n^2 + n \cdot \log n \in \Theta(0.5n^3)$
2. $n! \in O(n^n)$
3. $\log(n!) \in o(n \log n)$

2.3 (8 Punkte) Seien f_1, f_2, f_3 Funktionen mit $f_1 \in \mathrm{O}(f_2)$ und $f_2 \in \mathrm{o}(f_3)$. Beweisen Sie, dass $f_1 \in o(f_3)$. Argumentieren Sie über die Definitionen von O und o und **nicht** über Grenzwerte.

Aufgabe 3 (Assoziative und dynamische Felder, 20 Punkte)

3.1 (5 Punkte) Sei $h(x) = (26 \cdot x - 137) \bmod 5$ eine Hashfunktion. Berechnen Sie den Hashwert $h(x)$ für jeden Schlüssel x der Folge $1907, 1934, 1953, 1981,\ 1999$. Die Schlüssel werden in dieser Reihenfolge in eine Hashtabelle der Größe 5 eingefügt, mittels der Hashfunktion h und offener Adressierung. Zeichnen Sie den Zustand der Hashtabelle nach dem letzten Einfügen.

3.2 (7 Punkte) Für ein dynamisches Feld sei s_i die Anzahl der Elemente und c_i die Kapazität, jeweils nach Operation O_i. Sei die Vergrößerungsstrategie so, dass wenn vor einem *push* $c_{i-1} = s_{i-1}$, dann wird die Kapazität verdoppelt ($c_i = 2 \cdot c_{i-1}$). Sei die Verkleinerungsstrategie so, dass wenn vor einem *pop* $s_{i-1} \leq c_{i-1}/2$, dann wird die Kapazität halbiert ($c_i = \lceil c_{i-1}/2 \rceil$). Geben Sie für $n \in \mathbb{N}$ eine Folge von $4n$ Operationen an, deren Kosten $\Omega(n^2)$ sind, wenn zu Beginn $s_0 = c_0 = n$ (das n darf dabei nicht als konstant angenommen werden). Geben Sie die Werte von s_i und c_i nach jeder Operation an.

3.3 (8 Punkte) Wir hatten in der Vorlesung ein Mastertheorem formuliert für eine Folge von n Operationen $O_1, \ldots, O_n$, Laufzeit T_i für Operation O_i, Potenzial Φ_i nach O_i, und Potential Φ_0 am Anfang. Die Bedingungen waren, dass $\Phi_i \geq 0$ und $T_i \leq A + B \cdot (\Phi_{i-1} - \Phi_i)$ für $i = 1, \ldots, n$ und Konstanten $A, B > 0$.

Geben Sie für jede der folgenden Aussagen an, ob sie *wahr* oder *falsch* ist. Eine richtige Antwort gibt 2 Punkte, eine falsche Antwort gibt 2 Punkte Abzug. Wenn sie die Antwort nicht wissen, schreiben Sie *keine Aussage* oder nichts. Sie können nicht weniger als 0 Punkte für diese Aufgabe bekommen.

1. Wenn die Bedingungen erfüllt sind, dann gilt $\sum_{i=1}^{n} T_i = O(n + \Phi_0)$.
2. Wenn die Bedingungen erfüllt sind, aber statt $\Phi_i \geq 0$ ist $\Phi_n \geq \Phi_0$, dann gilt $\sum_{i=1}^{n} T_i = O(n)$.
3. Wenn die Φ_i alle negativ sind, kann die Bedingung $T_i \leq A + B \cdot (\Phi_{i-1} - \Phi_i)$ nicht erfüllt sein.
4. Wenn alle $T_i = O(1)$ für alle i, dann erfüllt $\Phi_i = i$ die Bedingungen für geeignete $A, B > 0$.

Aufgabe 4 (Blockoperationen und binäre Suchbäume, 20 Punkte)

4.1 (6 Punkte) Sei A ein Feld mit n Elementen, das an einer Blockgrenze anfängt. Über die n Elemente des Feldes wird in zufälliger Reihenfolge iteriert. Sei M die Größe des schnellen Speichers und B die Blockgröße. Wie viele Blockoperationen werden im besten Fall benötigt und wie viele im schlechtesten Fall, wenn $M = 2 \cdot B$ und $n \gg M$? Mit Begründung!

4.2 (6 Punkte) Ein binärer Suchbaum mit Tiefe d heißt *fast vollständig*, wenn er auf jeder Ebene die maximal mögliche Anzahl an Knoten hat, außer auf der tiefsten Ebene (ein Baum, der nur aus der Wurzel besteht, hat Tiefe 0). Geben Sie die minimale und maximale Anzahl an Knoten eines fast vollständiger Baumes der Tiefe d in Abhängigkeit von d an. Mit Begründung!

4.3 (8 Punkte) Schreiben Sie eine rekursive Funktion *is_valid_bst(node: BinarySearchTreeNode)*, die den Wurzelknoten eines binären Baumes als Argument nimmt und prüft, ob der zugehörige Baum ein valider binärer Suchbaum ist. Sie können annehmen, dass ein *BinarySearchTreeNode* die folgenden Attribute hat: *key* (der Schlüssel, ein Integer), *value* (der Wert, ein beliebiges Objekt), *left* (linkes Kind, ein *BinarySearchTreeNode* oder *None*) und *right* (rechtes Kind, ein *BinarySearchTreeNode* oder *None*). Die Funktion soll maximal 10 Zeilen lang sein.

Aufgabe 5 (Heaps und Graphen, 20 Punkte)

5.1 (5 Punkte) Sei $A = [\times, 3, 5, 8, 12, 7]$ die Speicherung eines binären Heaps in einem Feld. Malen Sie den initialen Zustand des Heaps als Baum. Führen Sie dann nacheinander die folgenden Operationen auf dem Heap aus und geben Sie den Zustand des Feldes A nach *jeder* Operation an: *insert(6)*, *delete_min()*, *delete_min()*, *insert(1)*.

5.2 (10 Punkte) Schreiben Sie eine Funktion *merge(lists)*, die eine Liste von k jeweils aufsteigend sortierten, nicht-leeren Listen von Zahlen als Eingabe akzeptiert und eine einzige aufsteigend sortierte Liste zurückgibt, die genau die Elemente aus allen Listen enthält. Ihre Funktion soll maximal 15 Zeilen umfassen und in Zeit $O(n \cdot \log k)$ laufen, wobei n die Gesamtzahl der Elemente ist. Sie dürfen eine Klasse *PriorityQueue* annehmen, mit einer Operation *get_min*, deren Laufzeit konstant ist, und Operationen *insert* und *delete_min*, deren Laufzeit jeweils $O(\log m)$ ist, wenn m Elemente in der PriorityQueue gespeichert sind.

5.3 (5 Punkte) Sei $P = u_1, \ldots, u_k$ ein kürzester Pfad von u_1 nach u_k durch einen Graph $G = (V, E)$, wobei $k \geq 2$ und $u_1, \ldots, u_k \in V$. Beweisen oder widerlegen Sie: der Pfad $Q = u_1, \ldots, u_{k-1}$ ist ebenfalls ein kürzester Pfad von u_1 nach u_{k-1}.