# Algorithmen und Datenstrukturen 1

ALGO1 · SoSe-2021 · tcs.uni-frankfurt.de/algo1

## Wochenplan: Prioritätswarteschlangen, Heaps *Revision 6f7541f (2021-05-25)*

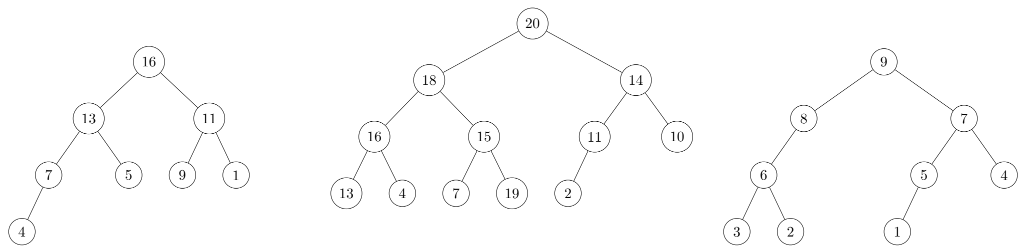
## Vorbereitung

Lies CLRS Kapitel 6, sowie Appendix B.5 und schau das Video der Woche.

## Dienstag

**Aufgabe 1 (Heap-Eigenschaften).** Löse die folgenden Teilaufgaben.

a) (einfach) Welche der folgenden Bäume erfüllen die Heap-Eigenschaft?




(a) (b) (c)

b) (einfach) Welche der durch folgende Felder repräsentierten Bäume erfüllen die Heap-Eigenschaft? Index 0 wird nicht benutzt und ist deshalb mit – markiert.

$$A = [-, 9, 7, 8, 3, 4] \qquad B = [-, 12, 4, 7, 1, 2, 10] \qquad C = [-, 5, 7, 8, 3]$$

c) (einfach) Sei $S = 4, 8, 11, 5, 21, \star, 2, \star$ eine Sequenz von Operationen, wobei eine Zahl für das Einfügen dieser Zahl in den Heap steht und $\star$ für eine `ExtractMax` Operation. Wie sieht der Heap $H$ nach jeder einzelnen Operation aus, wenn $H$ anfangs leer ist?

d) Erfüllt ein sortiertes Feld die Heap-Eigenschaft?

e) Wo befindet sich in einem (Max-)Heap das kleinste Element?

f) Zeige, dass `Insert`, `ExtractMax` und `IncreaseKey` die Heap-Eigenschaft aufrechterhalten.

g) (schwer) Angenommen wir erhalten $k$ sortierte Felder mit **insgesamt** $n$ Elementen als Eingabe. Zeige, wie sich alle Felder in Zeit $O(n \log k)$ zu einem einzelnen sortierten Feld der Länge $n$ verflechten lassen.

**Aufgabe 2 (Priogruppen-Politik, einfach).** Die Kakistokratische Partei will deine Hilfe, um ihre neue „Frischluft“-Politik umzusetzen. Entwirf ein Bürgerregister, das alle Bürger:innen und ihre Gehälter so speichert, dass man die Person mit dem geringsten Einkommen möglichst schnell finden und ausbürgern kann.

Das System soll die folgenden Operationen unterstützen:

- `Insert(c,i)` fügt eine Person mit der Sozialversicherungsnummer $c$ und dem jährlichen Gehalt $i$ ein.
- `DeportLowestIncome()` Gibt die Person mit dem niedrigsten Einkommen aus und entfernt sie aus dem System.

Entwirf eine möglichst effiziente Datenstruktur, die das System implementiert.

**Aufgabe 3 (Operationen für Prioritätswarteschlangen).** Wir wollen nun die Menge an zur Verfügung stehenden Operationen für Prioritätswarteschlangen vergrößern. Wir interessieren uns hierbei für die folgenden Operationen:

- `RemoveLargest(`$m$`)` entfernt das $m$t-größte Element der Prioritätswarteschlange.
- `Delete(`$x$`)` entfernt Element $x$ aus der Prioritätswarteschlange.
- `Fusion(`$x,y$`)` entfernt Elemente $x$ und $y$ aus der Prioritätswarteschlange und fügt ein neues Element $z$ mit Schlüssel $x$.key + $y$.key ein.
- `FindLarger(`$k$`)` gibt all jene Elemente der Prioritätswarteschlange aus, deren Schlüssel mindestens so groß wie $k$ ist.
- `ExtractMin()` gibt das Element der Prioritätswarteschlange mit dem kleinsten Schlüssel aus und entfernt es.

Wir wollen diese Operationen effizient implementieren, ohne dass sich die Komplexität der Standardoperationen `Insert`, `IncreaseKey`, `Max` und `ExtractMax` ändert.

Sei $n$ die Anzahl der Elemente in der Prioritätswarteschlange. Löse die folgenden Teilaufgaben:

a) Erkläre wie sich `RemoveLargest(`$m$`)` mit Zeitbedarf $O(m \log n)$ implementieren lässt.
b) Erkläre wie sich `Delete(`$x$`)` und `Fusion(`$x,y$`)` mit Zeitbedarf $O(\log n)$ implementieren lässt.
c) (schwer) Erkläre wie sich `FindLarger(`$k$`)` mit Zeitbedarf $O(m)$ implementieren lässt, wobei $m$ die Anzahl der Elemente mit Schlüssel $\geq k$ ist.
d) (schwer) Erkläre wie sich `ExtractMin()` mit Zeitbedarf $O(\log n)$ implementieren lässt.

## Donnerstag

**Aufgabe 4 (Zusätzliche Daten).** Sei $A[0..n-1]$ ein als Feld gespeicherter Heap. Jedes Element `x` in dem Heap wird durch einen Index $i$ repräsentiert und hat einen Schlüssel `x.key`, der als $A[i]$ gespeichert ist. Es ist oftmals nützlich, zusätzliche Daten `x.data` zu speichern, die mit einem Element `x` assoziiert sind. Modifiziere die Datenstruktur so, dass eine neue Operation `Data(`$i$`)` in Zeit $O(1)$ die zusätzlichen Daten des Elements mit Index $i$ zurückliefert. Hierbei dürfen sich die asymptotischen Laufzeiten der Standardoperationen des Heaps nicht verändern.

**Aufgabe 5 (Eigenschaften von Heaps).** Sei $T = (V, E)$ ein vollständiger Binärbaum von Höhe $h$. Löse die folgenden Teilaufgaben.

a) Zeige, dass für die Anzahl an Knoten $|V| = 2^{h+1} - 1$ gilt.
*Hinweis: Begründe, dass* $|V| = 1 + 2 + 4 + \ldots + 2^h$ *gilt und betrachte diesen Wert als Binärzahl.*

b) Zeige: Für die Summe $S$ mit $S = n/4 \cdot 1 + n/8 \cdot 2 + n/16 \cdot 3 + n/32 \cdot 4 + \ldots$ gilt $S = \Theta(n)$.
*Hinweis: Berechne* $S - S/2$

**Aufgabe 6 (Summen).** Sei $A[0..n-1]$ ein Feld von ganzen Zahlen. Wir interessieren uns für die folgenden Operationen:

- `Sum`$(i, j)$ gibt $A[i] + A[i+1] + \ldots + A[j]$ aus.
- `Change`$(i, x)$ setzt $A[i]$ auf den Wert $x$.

Löse die folgenden Teilaufgaben:

a) (einfach) Entwirf eine Datenstruktur, die `Sum` mit $O(1)$ Zeit und $O(n^2)$ Platz unterstützt.

b) (schwer) Entwirf eine Datenstruktur, die `Sum` mit $O(1)$ Zeit und $O(n)$ Platz unterstützt.

c) (sehr schwer) Entwirf eine Datenstruktur, die `Sum` und `Change` beide mit $O(\log n)$ Zeit und $O(n)$ Platz unterstützt.