

### Aufgabe 31

- (a) Im Folgenden wird davon ausgegangen, dass es sich um einen min-Heap handelt.
- (i) Bei der Suche nach dem kleinsten Element wird einfach die oberste Wurzel des Baumes genommen, dementsprechend gilt  $T(n) \in \Theta(1)$ .
  - (ii) Das größte Element liegt aufgrund der Struktur des Heaps in einem der Blätter. Da der Heap allerdings eine Baumstruktur umsetzt, müssen alle Knoten durchlaufen werden, bis man zu den Blättern gelangt. Also:  $T(n) \in \mathcal{O}(n)$ .
  - (iii) Bei der Suche nach einem beliebigen Element werden im worst-case alle Elemente verglichen, also  $T(n) \in \mathcal{O}(n)$ .
  - (iv) Analog bei einem nicht vorhandenen Element:  $T(n) \in \mathcal{O}(n)$ .
- (b) Der ADT Stack wird als Heap vom Prinzip so implementiert, dass jedes einzufügende Element eine aufsteigende Nummer bekommt. Im Folgenden sei der Heap ein max-Heap. Mehr dazu:
- *create*: Erstellt einen leeren Stack bzw. Heap.
  - *push*: Fügt ein Element in den Heap ein, indem dem einzufügenden Element die nachfolgende Nummer des im Wurzelknotens gespeicherten Elements zugewiesen wird. Nun wird das Element anhand der Nummer eingefügt, also an oberster Stelle. Falls der Heap leer ist, wird dem Element die Nummer 1 zugewiesen und im Wurzelknoten gespeichert.
  - *empty?*: Überprüft, ob der Stack bzw. Heap leer ist, indem überprüft wird, ob der Wurzelknoten leer ist oder nicht.
  - *top*: Gibt das Element des Wurzelknotens wieder, also das zuletzt eingefügte Element.
  - *pop*: Löscht das Element des Wurzelknotens, also das zuletzt eingefügte. Danach wird der Heap wiederhergestellt, um die Ordnung zu erhalten.

Diese Implementierung erfüllt auch die Bedingung, dass für jedes abzuspeichernde Element maximal  $\mathcal{O}(1)$  zusätzlicher Speicher verwendet werden darf.

... Welche technischen Probleme können auftreten?

- Es wird mehr Speicher verbraucht, da die Wartenummer mitgespeichert werden muss
- die Nummern können bei riesigen Bäumen irgendwann ausgehen, wenn der Wertebereich überschritten wird, aber vorher wird man eher ein Speicherproblem haben.

Im Folgenden bezeichne  $n$  die Anzahl der Elemente auf dem Stack. Die asymptotische Laufzeitkomplexität der einzelnen Operationen sieht wie folgt aus:

- *create*: Zeit zum erstellen eines leeren Heaps ist konstant:  $T(n) \in \Theta(1)$ .
- *push*: Das Einfügen erfolgt, indem man wie in der VL das einzufügende Element an die Position des nächsten Blattes gesetzt wird. Danach steigt es (nach Konstruktion) bis nach ganz oben, man spart sich deswegen Überprüfungen. D.h. es werden  $T(n) \in \Theta(\log n)$  Vertauschungen benötigt, entsprechend der Höhe des Baumes.
- *empty?*: Es muss nur überprüft werden, ob der Wurzelknoten leer ist, also  $T(n) \in \Theta(1)$ .
- *top*: Da nur der Wurzelknoten zurückgegeben wird, ist die Laufzeit konstant, d.h.  $T(n) \in \Theta(1)$ .
- *pop*: Diese Operation entspricht der Löschoperation eines Heaps. Die Zeit zum speichern des letzten Blatts im Wurzelknoten ist konstant. Das "sinken" des letzten Blatts benötigt  $\log n$  Vergleiche und Vertauschungen, also  $T(n) \in \Theta(\log n)$ .  
(Es gilt sogar  $\Theta(\cdot)$ , weil das letzte Blatt nach Konstruktion nach ganz unten sinkt)

# Aufgabe 3.1 (c)

Felix Jannen, Marcel Schoppmeier,  
Benedikt Ryse

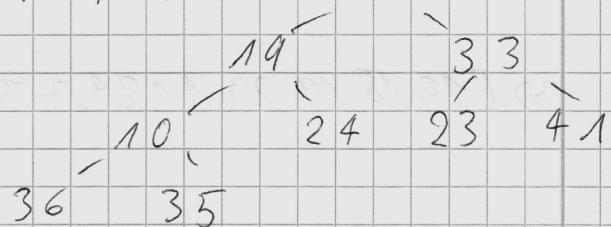
`int[] a = [18, 19, 33, 10, 24, 23, 41, 36, 35]`

Heap erstellen

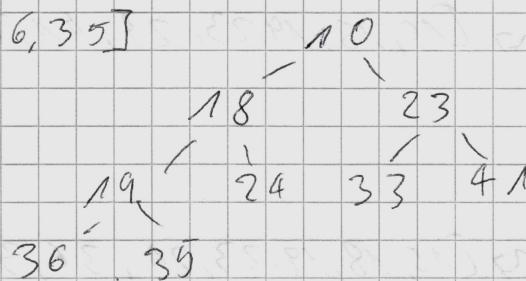
Array

$\rightsquigarrow [18, 19, 33, 10, 24, 23, 41, 36, 35]$

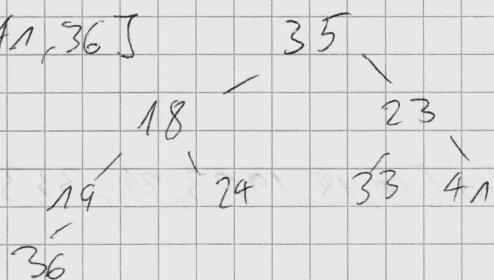
Heap



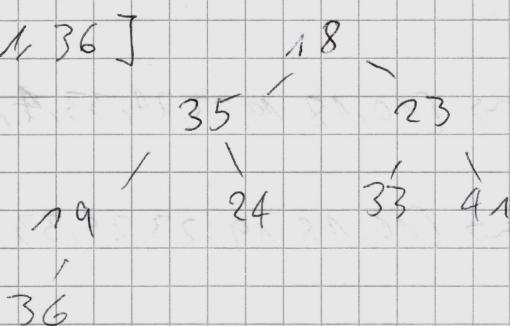
$\rightsquigarrow [10, 18, 23, 19, 24, 33, 41, 36, 35]$



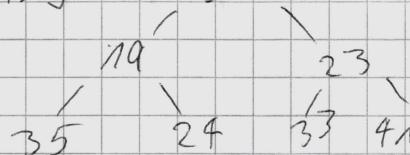
Heap sortieren:  $\rightsquigarrow [10, 35, 18, 23, 19, 24, 33, 41, 36]$



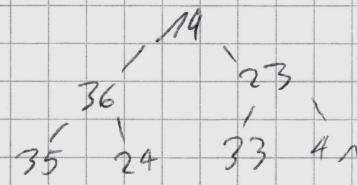
$\rightsquigarrow [10, 18, 35, 23, 19, 24, 33, 41, 36]$



$\rightsquigarrow [10, 18, 36, 19, 23, 35, 24, 33, 41]$



$\rightsquigarrow [10, 18, 19, 36, 23, 35, 24, 33, 41]$



$\rightsquigarrow [10, 18, 19, 21, 24, 23, 35, 36, 33]$

41  
/ \  
24 23  
/ / /  
35 36 33

$\rightsquigarrow [10, 18, 19, 23, 24, 41, 35, 36, 33]$

23  
/ \ 41  
24 / /  
/ 36 33  
35

$\rightsquigarrow [10, 18, 19, 23, 41, 24, 33, 35, 36]$

41  
/ \  
24 33  
/ /  
35 36

$\rightsquigarrow [10, 18, 19, 23, 24, 41, 33, 35, 36]$

24  
/ \ 33  
41 /  
/ 36  
35

$\rightsquigarrow [10, 18, 19, 23, 24, 36, 35, 33, 41]$

36  
/ \ 33  
35 /  
/ 41  
36

$\rightsquigarrow [10, 18, 19, 23, 24, 33, 35, 36, 41]$

33  
/ \ 36  
35 /  
/ 41  
36

$\rightsquigarrow [10, 18, 19, 23, 24, 33, 41, 35, 36]$

41  
/ \ 36  
35 /  
36

$\rightsquigarrow [10, 18, 19, 23, 24, 33, 35, 41, 36]$

35  
/ \ 36  
41 /  
36

$\rightsquigarrow [10, 18, 19, 23, 24, 33, 35, 36, 41]$

36  
/ 36  
41

$\rightsquigarrow \underline{\hspace{2cm}} \quad \underline{\hspace{2cm}} \quad 41$