Alexander Neuwirth (439218)
Leonhard Segger (440145)
Jonathan Sigrist (441760)

Aufgabe 31.

(a) (i) $\mathcal{O}(1)$, da die Wurzel stets das kleinste Element ist.

(ii) $\mathcal{O}(n/2) = \mathcal{O}(n)$, da eines der Blätter das Maximum ist.

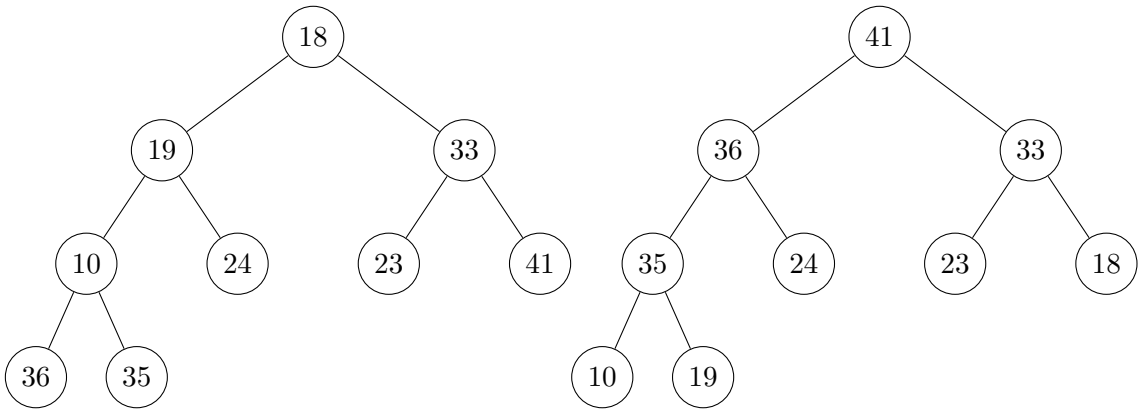
(iii) $\mathcal{O}(n)$, da jedes Element überprüft werden muss. Optimiert müssen keine Teilbäume durchsucht werden, deren Wurzel größer als das gesuchte Element sind, also $\mathcal{O}(n/2)$.

(iv) $\mathcal{O}(n)$, da man wie in (iii) nach einem Element sucht und dieses nicht finden wird.

(b) Jedes abzuspeichernde Element bekommt eine Zahl zugewiesen, welche pro Element im Heap zunimmt. Dabei wird ein max-Heap verwendet. Also `max` $= \mathcal{O}(1)$, `delete` $= \mathcal{O}(\log n)$ und `insert` $= \mathcal{O}(\log n)$.

(c) Zuerst wird der Heap aufgebaut, danach kann jeweils das größte Element aus dem Heap entfernt werden indem es mit dem letzten Element vertauscht und nicht weiter betrachtet wird. Wenn der Heap leer ist, ist das gesammte Array sortiert. Der ursprüngliche und aufgebaute Heap seien gegeben mit:

Alexander Neuwirth (439218)
Leonhard Segger (440145)
Jonathan Sigrist (441760)



18	19	33	10	24	23	41	36	35
18	19	33	36	24	23	41	10	35
18	19	41	36	24	23	33	10	35
18	36	41	19	24	23	33	10	35
18	36	41	35	24	23	33	10	19
41	36	18	35	24	23	33	10	19
41	36	33	35	24	23	18	10	19
19	36	33	35	24	23	18	10	*41*
36	19	33	35	24	23	18	10	*41*
36	35	33	19	24	23	18	10	*41*
10	35	33	19	24	23	18	*36*	*41*
35	10	33	19	24	23	18	*36*	*41*
35	24	33	19	10	23	18	*36*	*41*
18	24	33	19	10	23	*35*	*36*	*41*
33	24	18	19	10	23	*35*	*36*	*41*
33	24	23	19	10	18	*35*	*36*	*41*
18	24	23	19	10	*33*	*35*	*36*	*41*
24	18	23	19	10	*33*	*35*	*36*	*41*
24	19	23	18	10	*33*	*35*	*36*	*41*
10	19	23	18	*24*	*33*	*35*	*36*	*41*
23	19	10	18	*24*	*33*	*35*	*36*	*41*
18	19	10	*23*	*24*	*33*	*35*	*36*	*41*
19	18	10	*23*	*24*	*33*	*35*	*36*	*41*
10	18	*19*	*23*	*24*	*33*	*35*	*36*	*41*
18	10	*19*	*23*	*24*	*33*	*35*	*36*	*41*
10	*18*	*19*	*23*	*24*	*33*	*35*	*36*	*41*
10	*18*	*19*	*23*	*24*	*33*	*35*	*36*	*41*

Aufgabe 32.

(a) Die Kunden lassen sich mit einem Heapsort sortieren. Dieser hat die Komplexität $\mathcal{O}(n \log n)$ und Zugriff auf die $\log n$ Kunden braucht $\mathcal{O}(\log n)$ und auf die $\sqrt{n}$ Kunden braucht $\mathcal{O}(\sqrt{n})$. Da $\log n < \sqrt{n} < n$, folgt eine insgesammte Laufzeit von $\mathcal{O}(n \log n + \sqrt{n} + \log n) = \mathcal{O}(n \log n)$.

(b) Man kann Bucketsort einsetzen. Zuerst sucht man den Kunden mit den meisten Flügen

Alexander Neuwirth (439218)
Leonhard Segger (440145)
Jonathan Sigrist (441760)

(`max`, $\mathcal{O}(n)$). Dann sortiert man alle Kunden mittels Bucketsort mit der nun bekannten maximalen Wortlänge von $k = \log\{\texttt{max}\}$. Dieser hat eine Laufzeit von $\mathcal{O}(n \cdot k)$, wobei $k \ll n$ und k annähernd konstant, da der Kunde mit den meisten Flügen nur linear und nicht exponentiell oft fliegt. Dadurch ergibt sich eine gesammte Laufzeit von $\mathcal{O}(n + n \cdot k) = \mathcal{O}(n)$.

Aufgabe 33.

(a) Datentyp: DEQUEUE

create:		$\rightarrow$	DEQUEUE
enquenefront:	ELEM $\times$ DEQUEUE	$\rightarrow$	DEQUEUE
enqueneback:	ELEM $\times$ DEQUEUE	$\rightarrow$	DEQUEUE
empty:	DEQUEUE	$\rightarrow$	BOOL
front:	DEQUEUE	$\rightarrow$	ELEM
back:	DEQUEUE	$\rightarrow$	ELEM
dequeuefront:	DEQUEUE	$\rightarrow$	DEQUEUE
dequeueback:	DEQUEUE	$\rightarrow$	DEQUEUE

Ende Datentyp

Aufgabe 34.
Ähnlich zum Heapsort wird beim Einfügen ein Element verändert und dann die Ordnung wiederhergestellt. Die `delete`-Operation ändert ebenfalls nur einen Wert und es muss die Ordnung wiederhergestellt werden. Also ist die Laufzeit mindestens $\mathcal{O}(\log_2 \log_2 n)$.