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

Aufgabe 22.

(a) Insertion Sort

0	16	19	33	10	24	23	41	36	35
1	***16***	19	33	10	24	23	41	36	35
2	16	***19***	33	10	24	23	41	36	35
3	16	19	***33***	10	24	23	41	36	35
4	**10**	16	19	*33*	24	23	41	36	35
5	10	16	19	**24**	*33*	23	41	36	35
6	10	16	19	**23**	24	*33*	41	36	35
7	10	16	19	23	24	33	***41***	36	35
8	10	16	19	23	24	33	**36**	*41*	35
9	10	16	19	23	24	33	**35**	36	*41*

(b) Bubble Sort

0	16	19	33	10	24	23	41	36	35
1	16	19	10	24	23	33	36	35	41
2	16	10	19	23	24	33	35	36	41
3	10	16	19	23	24	33	35	36	41

(c) Selection Sort

0	16	19	33	10	24	23	41	36	35
1	**10**	16	19	33	24	23	41	36	35
2	10	**16**	19	33	24	23	41	36	35
3	10	16	**19**	33	24	23	41	36	35
4	10	16	19	**23**	33	24	41	36	35
5	10	16	19	23	**24**	33	41	36	35
6	10	16	19	23	24	**33**	41	36	35
7	10	16	19	23	24	33	**35**	41	36
8	10	16	19	23	24	33	35	**36**	41
9	10	16	19	23	24	33	35	36	**41**

(d) Merge Sort

0	16	19	33	10	24	23	41	36	35						
1	16	19	33	10		24	23	41	36	35					
2	16	19		33	10		24	23		41	36	35			
3	16		19		33		10		24		23		41	36	35
4	16		19		33		10		24		23		41	36		35
5	16		19		33		10		24		23		41	35	36	
6	16	19		10	33		23	24		35	36	41				
7	10	16	19	33		23	24	35	36	41						
7	10	16	19	23	24	33	35	36	41							

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

(e) Quick Sort

0	16	19	33	10	24	23	41	36	35							
1	16	19	10	23		**24**		33	41	36	35					
2	16	10		**19**		23		24		33	36	35		**41**		
3	10		**16**		19		23		24		33	35		**36**		41
4	10		16		19		23		24		**33**		35		36		41
5	10	16	19	23	24	33	35	36	41							

Aufgabe 23.

Der sprechende Hut muss einen Algorithmus verwenden, welcher in-place Elemente miteinander vertauscht ($\mathcal{O}(1)$). Heapsort ist im Durchschnitt effizienter als Insertion-, Selection- und Bubble-Sort und hat einen besseren Worst-Case. Zwar ist der Best-Case bei Insertion- oder Bubble-Sort besser, jedoch ist der durchschnittliche Fall wichtiger.

Aufgabe 24.

Man kann beide Arrays sortieren (zum Beispiel mit Quicksort) und dann miteinander vergleichen ähnlich dem Algorithmus für das Zusammenfügen beim Mergesort. Man geht mit jeweils einem Index über beide Arrays und erhöht den des kleineren Wertes solange, bis beide Elemente wertegleich sind. Nun hat man ein Duplikat gefunden. Damit zwei gleiche Werte nicht doppelt gezählt werden, muss nun ein Index so lange erhöht werden, bis ein neuer Wert erreicht wurde. Der andere Index kann ignoriert werden, da dieser wegen `a[i]` $\neq$ `b[j]` eh erhöht wird. Sobalt man das Ende eines Arrays erreicht hat, bricht der Algorithmus ab.

Aufgabe 25.

Der Stack ist minimal, falls man immer genau in der Mitte aufteilt, also Pivot Mitte. In diesem Fall wird der Stack nicht größer als $2 \cdot \log_2(n)$, da zwei Werte pro Methodenaufruf auf den Stack kommen. Bei der hier angewendeten Methode ergibt sch im schlimmsten Fall eine Größe von $2n$ also $\mathcal{O}(n)$ da man sonst erst immer das mittlere Element finden müsste.