Prof. Dr. Ir. Joost-Pieter Katoen

Sebastian Junges, Benjamin Kaminski, David Korzeniewski, Tim Quatmann

Übung 6

– Musterlösung –

Hinweise:

- Aufgrund der Feiertage müssen die Lösungen bis **Mittwoch, den 30. Mai um 16:00 Uhr** in den entsprechenden Übungskasten eingeworfen werden. Sie finden die Kästen am Eingang Halifaxstr. des Informatikzentrums (Ahornstr. 55).
- Die Anmeldung zur Präsenzübung muss bis **Dienstag, den 29. Mai** über das Übungssystem erfolgen.
- Da die Tutorien am Donnerstag, den 31. Mai ausfallen, können Sie in der Woche vom 28. Mai bis 1. Juni eine andere Gruppe Ihrer Wahl besuchen. In der Pfingstwoche (21. bis 25. Mai) finden keine Lehrveranstaltungen statt.
- Die Übungsblätter **müssen** in Gruppen von je 3 Studierenden aus der gleichen Kleingruppenübung abgegeben werden.
- Drucken Sie ggf. digital angefertigte Lösungen aus. Abgaben z.B. per Email sind nicht zulässig.
- Namen und Matrikelnummer sowie die **Nummer der Übungsgruppe** sind auf jedes Blatt der Abgabe zu schreiben. Abgaben, die aus mehreren Blättern bestehen **müssen geheftet bzw. getackert** werden! Die **Gruppennummer muss sich auf der ersten Seite oben links** befinden.
- **Bei Nichtbeachten der obigen Hinweise müssen Sie mit erheblichen Punktabzügen rechnen!**

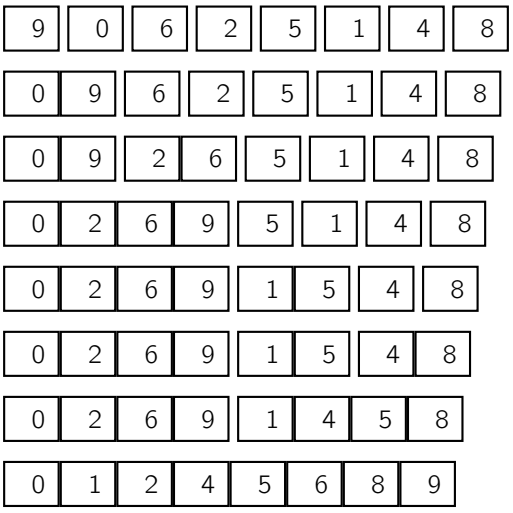
Aufgabe 1 (Mergesort): (16 Punkte)

Sortieren Sie das folgende Array mithilfe des in der Vorlesung vorgestellten Mergesort Algorithmus. Geben Sie dazu das Array nach jeder Merge-Operation an. Gehen Sie strikt nach dem Verfahren aus der Vorlesung vor.

9	0	6	2	5	1	4	8

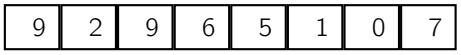
Lösung:

Die Schritte sind wie folgt.



Aufgabe 2 (Heapsort): (20+12=32 Punkte)

a) Sortieren Sie das folgende Array mithilfe des in der Vorlesung vorgestellten Heapsort Algorithmus (unter Benutzung von Max-Heaps). Geben Sie dazu das Array nach jeder Swap-Operation an und geben Sie zum jeweils noch unsortierten Arraybereich zusätzlich die grafische Darstellung als Heap an. Gehen Sie strikt nach dem Verfahren aus der Vorlesung vor.



b) Bestimmen Sie jeweils die asymptotische Best-case Laufzeit für die Methoden aus der Vorlesung

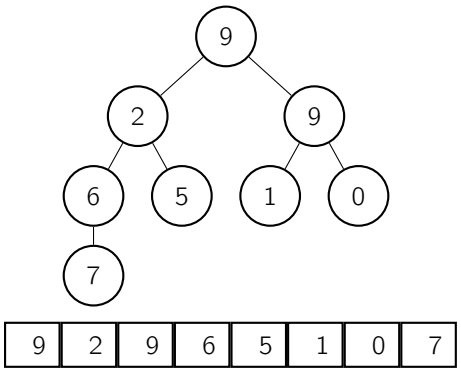
- `heapify`
- `buildHeap`
- `heapSort`

in Abhängigkeit der Eingabelänge n, d.h. bestimmen Sie jeweils ein $f(n)$ mit $B(n) \in \Theta(f(n))$. Geben Sie jeweils für jedes $n > 0$ eine Eingabe an, auf der die Laufzeit in $\Theta(B(n))$ liegt. Begründen Sie ihre Antworten.

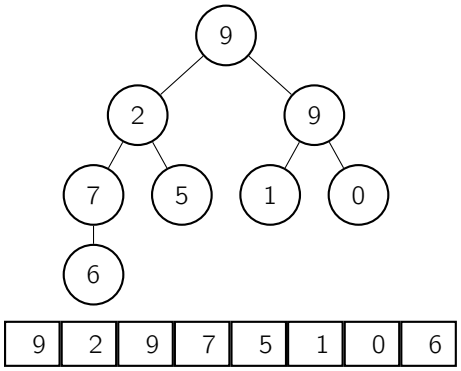
Lösung:

a) Die Schritte sind wie folgt.

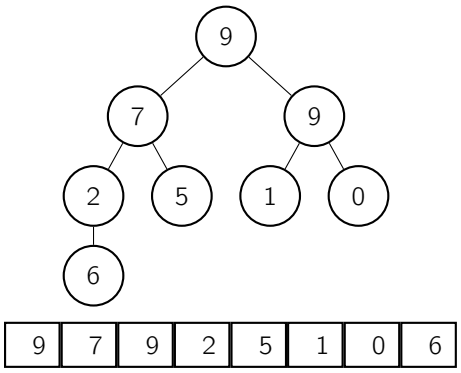
Schritt 0:



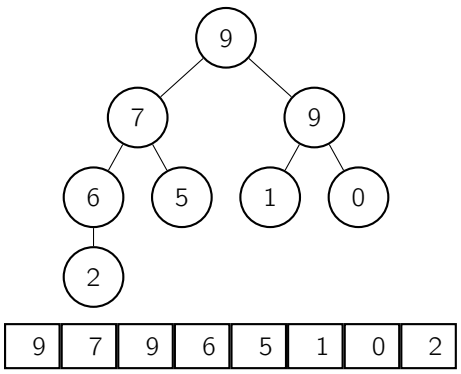
Schritt 1:



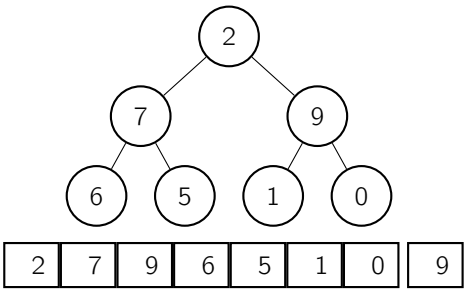
Schritt 2:



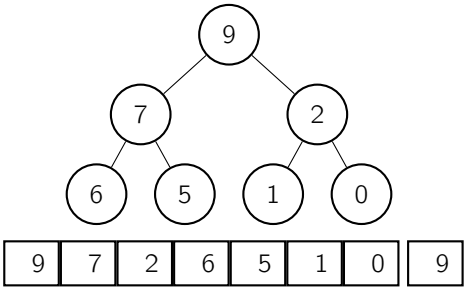
Schritt 3:



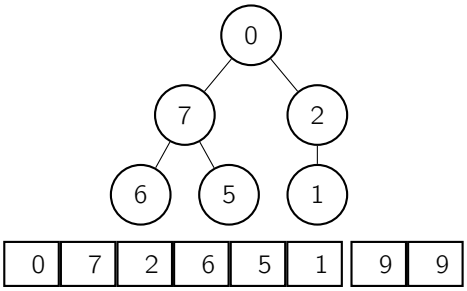
Schritt 4:



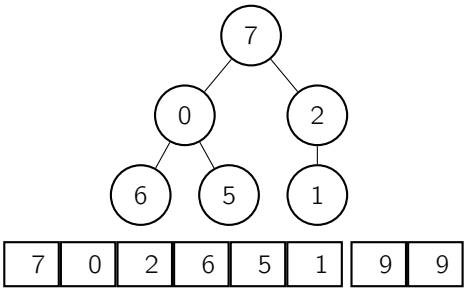
Schritt 5:



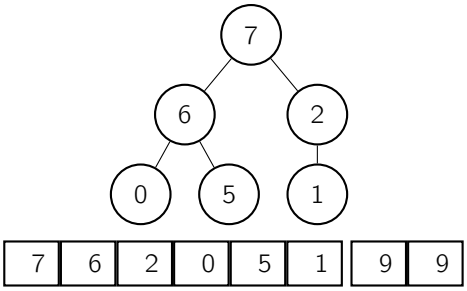
Schritt 6:



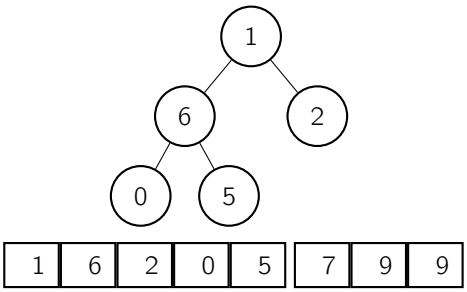
Schritt 7:



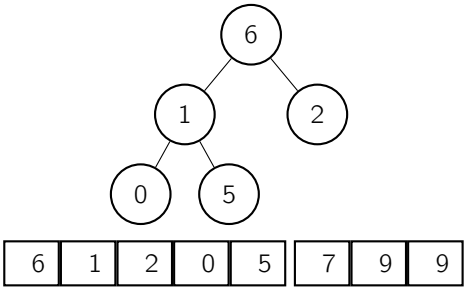
Schritt 8:



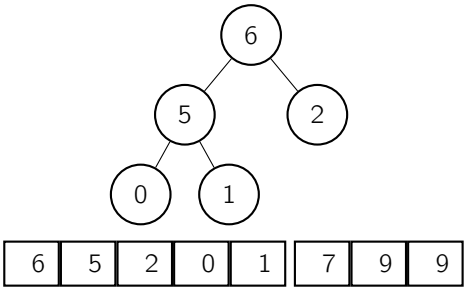
Schritt 9:



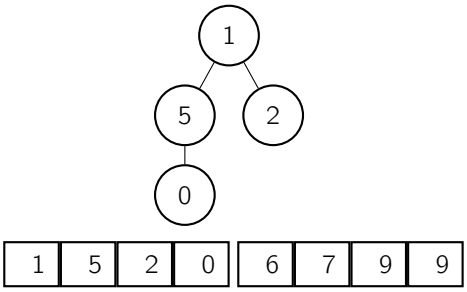
Schritt 10:



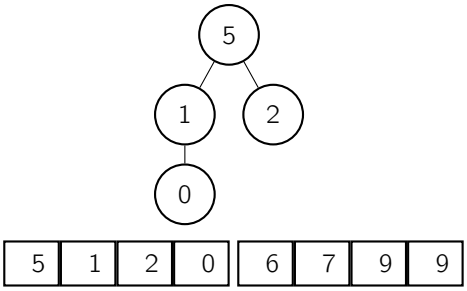
Schritt 11:



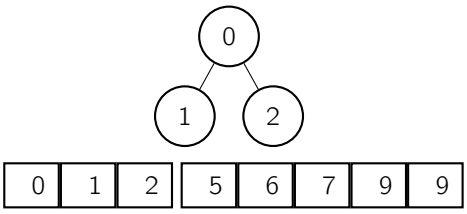
Schritt 12:



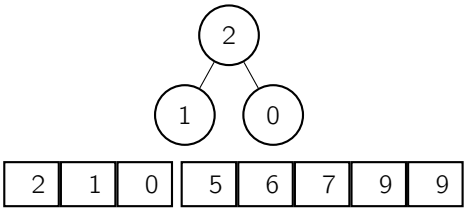
Schritt 13:



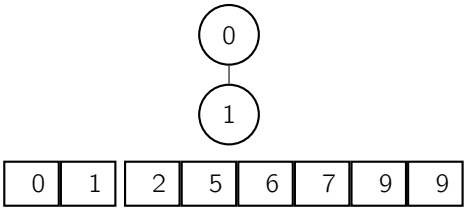
Schritt 14:



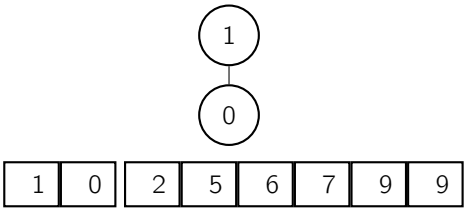
Schritt 15:



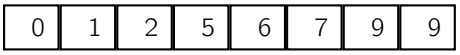
Schritt 16:



Schritt 17:



Schritt 18:



b)

- `heapify`: $B(n) \in \Theta(1)$, z.B. mit der Eingabe $\texttt{E} = [1, 1, \ldots, 1], \texttt{n} = \texttt{E.length}$ und $\texttt{pos} = 0$, da stets `E[pos] == E[next]` gilt und daher die Schleife immer in der ersten Iteration verlassen wird.
- `buildHeap`: $B(n) \in \Theta(n)$, z.B. mit der Eingabe $\texttt{E} = [1, 1, \ldots, 1]$: $B(n) \in \Omega(n)$ folgt, da die Schleife immer $\lfloor n/2 \rfloor \in \Theta(n)$ oft ausgeführt wird (unabhängig von der Eingabe). $B(n) \in \mathcal{O}(n)$ folgt, da $B(n) \leq W(n) \in \mathcal{O}(n)$ (laut Vorlesung).
- `heapSort`: $B(n) \in \Theta(n)$ z.B. mit der Eingabe $\texttt{E} = [1, 1, \ldots, 1]$, da
  a) die Laufzeit von `buildHeap` immer in $\Theta(n)$ liegt,
  b) die Laufzeit von `heapify` auf dieser Eingabe in $\Theta(1)$ liegt und
  c) die Schleife immer $n - 1 \in \Theta(n)$ mal durchlaufen wird.

Aufgabe 3 (Quicksort): (16 Punkte)

Sortieren Sie das folgende Array mithilfe des in der Vorlesung vorgestellten Quicksort Algorithmus. Geben Sie dazu das Array nach jeder Partition-Operation an und markieren Sie das jeweils verwendete Pivot-Element. Gehen Sie strikt nach dem Verfahren aus der Vorlesung vor.

6	4	4	9	3	2	7	5

Lösung:

Die Schritte sind wie folgt.

6	4	4	9	3	2	7	5
2	4	4	3	**5**	6	7	9
2	**3**	4	4	5	6	7	9
2	3	**4**	4	5	6	7	9
2	3	4	4	5	6	7	**9**
2	3	4	4	5	6	**7**	9

Aufgabe 4 (Stabilisieren): (20 Punkte)

Wir betrachten die Klasse `Element`:

```
class Element {
    int key;
    String name;
}
```

Sei `sort(Element E[])` ein beliebiger Sortieralgorithmus, der Elemente bezüglich ihres Schlüssels (`key`) aufsteigend sortiert, jedoch gegebenenfalls nicht stabil ist.
Aufgabe: Erstellen Sie einen stabilen Sortieralgorithmus `stableSort(Element E[])`, der unter Benutzung von `sort` das Eingabearray *stabil* sortiert. `stableSort` soll höchstens linearen Overhead haben, das heißt für die Worst-case Laufzeit von `stableSort` auf Eingaben der Länge $n = \texttt{E.length}$ soll gelten

$$W_{\texttt{stableSort}}(n) = W_{\texttt{sort}}(n) + f(n),$$

wobei $W_{\texttt{sort}}(n)$ die Worst-case Laufzeit von `sort` ist und $f(n) \in \mathcal{O}(n)$ gelten soll.

Hinweise:

- Wir nehmen an, dass Operationen wie $+$, $-$, $\cdot$ und $/$ sowie Vergleiche jeweils konstante Laufzeit benötigen.
- Die Klasse `Element` darf nicht erweitert werden.
- Die Methode `sort` arbeitet nur auf Elementen vom Typ `Element`.
- Der Datentyp `int` kann beliebig große Zahlen speichern.

Lösung:

Wir transformieren die Schlüssel der Elemente, sodass keine doppelten Schlüssel mehr vorkommen. Eine (potentiell nicht stabile) Sortierung der transformierten Schlüssel soll dann einer stabilen Sortierung der originalen Schlüssel entsprechen.

```
void stableSort (Element E[]) {
    int n = E.length;
    for (int i = 0; i < n; i++) {
      E[i].key = E[i].key * n + i; // Transformiere Schluessel
    }

    sort(E);

    for (int i = 0; i < n; i++) {
      E[i].key = floor(E[i].key / n); // Kehre Transformation um
    }
}
```

Aufgabe 5 (Sortierszenarien): (16 Punkte)

Geben Sie zu jedem der folgenden Sortierszenarien an, welches Sortierverfahren aus der Vorlesung (Insertionsort, Mergesort, Heapsort, Quicksort, Bubblesort, Countingsort) angewendet werden sollte. Begründen Sie Ihre Antworten.

a) 1000000 Arrays der Länge 10 sollen sortiert werden.

b) Die zu sortierende Eingabe ist sehr lang, enthält jedoch nur Schlüssel in $\{0, 1, 2, 3, 4\}$.

c) Die Wahrscheinlichkeit, dass `E[i] < E[i+1]` gilt ist für alle $i \in \{0, \ldots, \texttt{E.length} - 2\}$ sehr hoch.

d) 8 Assistenten möchten 1000 Klausuren nach der Matrikelnummer des Klausurteilnehmers sortieren.

Lösung:

a) Insertionsort, da die Methode wenig Overhead hat.

b) Countingsort, da hier die Laufzeit nur linear in der Arraylänge ist.

c) Insertionsort, da die Eingabe bereits fast sortiert ist und daher nur wenige Einsetzungen notwendig sind.

d) Mergesort, da sich das Verfahren durch das Divide and Conquer Prinzip leicht parallelisieren lässt.