Matthias Krause
Alexander Moch

Mannheim, den 12.10.2020

CS 307 Algorithmen und Datenstrukturen, Herbstsemester 2020

Übungsblatt 3

AUFGABE 3.1: Der folgende Algorithmus (in Pseudocode) findet das Minimum der Einträge in einem (unsortierten) Array $A[1 \cdot\cdot n]$ $(n > 1)$.

```
algorithm MINIMUM (A, n)
1    min := A[1];
2    for i := 2 to n
3       if (min > A[i])
4          min := A[i];
5    return min;
```

Für die average-case Analysen in dieser Aufgabe soll angenommen werden, dass jede Permutation von n verschiedenen Zahlen $b_1, \ldots, b_n$ mit der gleichen Wahrscheinlichkeit (nämlich $\frac{1}{n!}$) als Belegung von $A[1], \ldots, A[n]$ vorkommen kann.
Wie oft wird der Test in Zeile 3 in worst case, best case und average case ausgeführt? Kann ein anderer Algorithmus zur Bestimmung des Minimums in A mit weniger Vergleichen in worst, best oder average case auskommen?

AUFGABE 3.2: In vielen Anwendungen taucht das folgende modifizierte Sortierproblem auf: Es sind n Objekte (hier vereinfachend Integers) $a_1, \ldots, a_n$ in einem Array $A[1 \cdot\cdot n]$ gegeben. Es soll auf der Basis einer geeigneten Datenstuktur eine Routine entwickelt werden, die wiederholt aufgerufen werden kann und bei dem i-ten Aufruf, $i = 1, \ldots, n$, die i-tgrößte Zahl (genauer $a_{\pi(i)}$ für eine Permutation π mit $a_{\pi(1)} \geq \ldots \geq a_{\pi(n)}$) liefert. Die Zahl der Aufrufe k ist a priori nicht bekannt. Es soll allerdings von der Tatsache, dass k wesentlich kleiner als n sein kann, Nutzen gezogen werden; es ist also nicht ratsam, die Zahlen vorzusortieren und dann einfach eine nach der anderen auszugeben.

a) Wie würden Sie dieses Problem mit Hilfe eines Heaps lösen? (Hier und in der Teilaufgabe (b) reicht eine grobe Beschreibung der wesentlichen Schritte und deren worst-case Laufzeit; es soll kein Pseudocode angegeben werden).

b) Wie würden Sie dieses Problem mit Hilfe eines Partitionierungsansatzes, wie er bei Quicksort verwendet wird, lösen?

AUFGABE 3.3$^{\mathbf{K}}$: In dieser Aufgabe betrachten wir die Ausführung des in der Vorlesung beschriebenen Algorithmus HEAPSORT auf das Feld $A = [5, 2, 4, 3, 13, 8, 9, 11, 7, 6]$.

a) Geben Sie den Inhalt von A an, nachdem der Algorithmus zum ersten Mal einen (Max-) Heap daraus erstellt hat.

b) Geben Sie jeweils den (vollständigen) Inhalt von A an, jedes Mal wenn der Algorithmus eine weitere HEAPIFY ausgeführt hat.

AUFGABE 3.4$^{\mathbf{K}}$: Geben Sie für die unten aufgeführten Sortieralgorithmen die asymptotische Laufzeit in Θ-Form (also z.B. $\Theta(n^2)$) zum Sortieren eines n-elementigen Feldes mit dem angegebenen Inhalt an.

a) Insertionsort auf $[1, 2, 3, \ldots, n - \lfloor\sqrt{n}\rfloor, n, n-1, n-2, \ldots, n - \lfloor\sqrt{n}\rfloor + 1]$

b) Mergesort auf $[2, 1, 4, 3, \ldots, n, n-1]$

c) Quicksort auf $[1, 2, 3, \ldots, \lfloor\frac{n}{2}\rfloor, n, n-1, n-2, \ldots, \lfloor\frac{n}{2}\rfloor + 1]$

d) Heapsort auf $[n, n-1, n-2, \ldots, 3, 2, 1]$

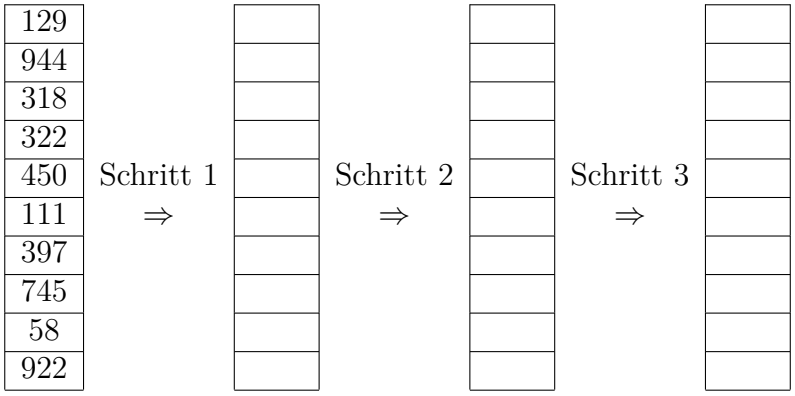
AUFGABE 3.5:
Zeichnen Sie den Entscheidungsbaum $T(n)$ für

a) Quicksort und $n = 3$,

b) Heapsort und $n = 3$.

AUFGABE 3.6$^{\mathbf{K}}$:

a) Gegeben ist das folgende Feld $A[1 \cdot \cdot 10]$, in dem Schlüssel aus der Menge $\{0, \ldots, 999\}$ gespeichert sind. Benutzen Sie den in der Vorlesung vorgestellten Algorithmus *Radix Sort*, um das Feld in aufsteigender Reihenfolge zu sortieren. Tragen Sie dabei die Ergebnisse der Zwischenschritte in nachfolgendes Schema ein!



b) Wir betrachten nun das allgemeine Schema für *Radix Sort*. Es soll ein Feld mit n Elementen sortiert werden. Die Schlüssel seien dabei d-stellige Zahlen mit Ziffern aus $\{0, \ldots, k-1\}$. Im Teil (a) gilt also beispielsweise $n = 10$, $d = 3$, $k = 10$.
In der Vorlesung wurde dargestellt, dass *Radix Sort* für jeden Zwischenschritt ein geeignetes Sortierverfahren verwenden kann. Geben Sie an, welche der nachfolgend aufgeführten (aus der Vorlesung bekannten) Algorithmen (ohne Modifizierung) für eine Implementierung von *Radix Sort* in Frage kommen. Geben Sie für die geeigneten Algorithmen die resultierende (worst case) Laufzeit von *Radix Sort* (in Abhängigkeit von n, d und k) in der O-Notation an.

	Geeignet? (ja/nein)	Falls geeignet: Laufzeit Radix Sort
Insertion Sort		
Merge Sort		
Quick Sort		
Heap Sort		
Counting Sort		