

Lösungen zum Thema Laufzeit

(1) (a) $n=0$: / $n=1$: 1 $n=2$: $\begin{array}{c} 1 \\ | \\ 2 \end{array}$ $n=3$: $\begin{array}{cc} 1 & \\ / \backslash & \\ 2 & 3 \end{array}$ $\begin{array}{cc} 1 & \\ / \backslash & \\ 2 & 3 \end{array}$

$n=4$: $\begin{array}{cc} 1 & \\ / \backslash & \\ 2 & 3 \\ / & \\ 4 & \end{array}$ $\begin{array}{cc} 1 & \\ / \backslash & \\ 2 & 4 \\ / & \\ 3 & \end{array}$ $\begin{array}{cc} 1 & \\ / \backslash & \\ 3 & 2 \\ / & \\ 4 & \end{array}$ $\begin{array}{cc} 1 & \\ / & \\ 2 & \\ / \backslash & \\ 3 & 4 \end{array}$ $\begin{array}{cc} 1 & \\ / & \\ 2 & \\ / \backslash & \\ 4 & 3 \end{array}$

(b) Alle Heaps für $n \leq 3$, bei $n = 4$ die ersten drei Heaps.

(c)

1	2	6	3	⊥	8	7	⊥	⊥	⊥	⊥	9
---	---	---	---	---	---	---	---	---	---	---	---

(2) Bemerkung: Die Lösungen hängen natürlich von der gewählten Datenstruktur ab. Dies ist nur eine mögliche Lösung.

(a) Doppelt verkettete Liste mit zusätzlichem Attribut *min*.

(b) `add(L, x) //füge x in die Liste L ein`
 `// hier wäre der Code für das reine Einfügen in die Liste`
 `if x < L.min`
 `L.min = x`

(c) `minimum(L) //gib Minimum aus L zurück`
 `return L.min`

(d) Falls das zu löschende Element das minimale war, muss die komplette Liste nach dem neuen Minimum durchsucht werden. Die Laufzeit ist $\Theta(n)$. Bei einer normalen doppelt verketteten Liste wäre das Löschen (bei Kenntnis des Pointers) aber in konstanter Zeit möglich!

(3) (a) Lösungsvorschlag: Hashtabelle mit Verkettung - Operationen laufen bei hinreichender Größe des Feldes in $O(1)$, Speicherplatz ist nicht beschränkt (außer durch Größe des Speichermediums).

(b) Lösungsvorschlag: Warteschlange (Queue) - durch die FiFo-Strategie ändert sich die Reihenfolge der Anfragen beim Puffern nicht, außerdem werden ENQUEUE und DEQUEUE in jeweils konstantem Zeitaufwand ausgeführt.

(c) Lösungsvorschlag: Max-Heap, da hier sowohl die Auswahl des Elements mit dem größten Schlüssel als auch das Einfügen neuer Prozesse hier besonders schnell ($O(\log(n))$) erfolgen kann.