Algorithmen und Datenstrukturen, Übung 4

Marouane Soussi, Lars Happel, Mustafa Miresh

April 2022

Aufgabe 1

siehe .java Datei

Aufgabe 2

a)-$T(n) = 3T(\frac{n}{9}) + n^{\frac{1}{3}}$

a=3 und b=9. Wir versuchen die Gleichung mit dem Master Theorem zu lösen. Erstmal ist : $n^{log_9 3} = n^{1/2}$. Wir sind gerade im ersten Fal. wähle $\epsilon = \frac{1}{6}$ dann wäre es $n^{\frac{1}{2}-\frac{1}{6}} = n^{\frac{1}{3}}$ und somit $n^{\frac{1}{3}} \in O(n^{\frac{1}{3}})$. Denn $\lim_{n\to\infty} |\frac{\frac{1}{n^(1/3)}}{\frac{1}{n^(1/3)}}| = 1 < \infty$ Dh $T(n) = \theta(n^{\frac{1}{2}})$

b)-$T(n) = 4T(\frac{n}{2}) + nlog(n)$

a=4 b=2. Wir versuchen die Gleichung mit dem Master Theorem zu lösen. Erstmal ist : $n^{log_2(4)} = n^2$. Wir sind auch gerade hier im ersten Fall. Wähle $\epsilon = \frac{3}{2}$ hätten wir dann : $\lim_{n\to\infty} |\frac{nlog(n)}{n^(2-3/2)}| = \lim_{n\to\infty} |\frac{nlog(n)}{n^(1/2)}| = \lim_{n\to\infty} |\frac{log(n)}{n^(1/2)}|$. Da beide Nenner und Zähler nach unendlich gehen könnten wir die L'Hopitalsche Regel anweden. erhalten wir nach Ableitung : $\lim_{n\to\infty} |\frac{ln(2)\frac{1}{n}}{\frac{1}{2n^{\frac{1}{2}}}}| = \lim_{n\to\infty} |2ln(2)n^{\frac{-1}{2}}|$ was gegen 0 konvergiert somit ist $f \in O(n^{\frac{3}{2}})$. Dh $T(n) = \theta(n^2)$

c) $T(n) = T(n-3) + n$

Wir wenden die Iterationsmethode an und betrachten zuerst einige Iterationsschritte:
0. : $T(n) =$
1. : $T(n-3) + n =$
2. : $T(n-6) + n - 3 + n = T(n-6) + 2n - 3 =$
3. : $T(n-9) + n - 6 + 2n - 3 = T(n-9) + 3n - 9 =$
4. : $T(n-12) + n - 9 + 3n - 9 = T(n-12) + 4n - 18 =$
5. : $T(n-15) + n - 12 + 4n - 18 = T(n-15) + 5n - 30$

Es stellt sich heraus, dass die Rekursionsschritte das Muster $T(n-3k)+k*n-3\sum_{i=1}^{k-1} i$ haben. Da in jedem Rekursionsschritt n um 3 verringert wird, kann es maximal eine Rekursionstiefe von $\frac{n}{3}$ geben. Wir setzen daher k auf $\frac{n}{3}$ und erhalten $T(n-3*\frac{n}{3})+\frac{n}{3}*n-3\frac{\frac{n}{3}(\frac{n}{3}-1)}{2} = T(0)+\frac{n^2}{3}-\frac{\frac{n^2}{3}-n}{2} = \frac{n^2}{3}-\frac{n^2-3n}{6} = \frac{n^2}{6}+\frac{n}{2}$.
Also $T(n) \in \Theta(\frac{n^2}{3}) = \Theta(n^2)$.

d) $T(n) = 2T(n-4)$

Mittels Iterationsmethode ermitteln wir wieder einige Iterationsschritte:
$0.: T(n) =$
$1.: 2T(n-4) =$
$2.: 4T(n-8) =$
$3.: 8T(n-12) =$
$4.: 16T(n-16) =$
$5.: 32T(n-20) =$
Es ergibt sich das Schema $2^kT(n-4k)$. Da es maximal $\frac{n}{4}$ Rekursionsschritte geben kann, setzen wir $k=\frac{n}{4}$. Dies ergibt $2^{\frac{n}{4}}T(n-4*\frac{n}{4}) = 2^{\frac{n}{4}}*T(0)$ und somit ist $T(n) \in \Theta(2^{\frac{n}{4}}) = \Theta(2^n)$

Aufgabe 3

a) DeleteSmallerThanLast(L)

Input: Eine unsortierte verkettete Liste mit natürlichen Zahlen Output: Die Input-Liste abzüglich aller Elemente die kleiner als das letzte Element der Liste sind

```
DeleteSmallerThanLast(L):
    // 1. Durchlauf: Letzten Wert finden
    x := L.head
    while (x.next != NIL):
        x := x.next
    lastValue := x.key

    // 2. Durchlauf: Kleinere Werte löschen
    x := L.head
    while (x.next != NIL):
        prev := x
        x := x.next
        if x.key < lastValue:
            prev.next := x.next
```

b) CutSubList

Input: Eine (unsortierte) verkettete Liste mit natürlichen Zahlen, ein Start- und ein Endwert Output: Die Input-Liste abzüglich der Teilliste von Start- bis Endwert

```
CutSublist(L, a, b):
    if (a > b): return
    x := L.head
    index := 0
    while (x.next != NIL):
        prev := x
        x := x.next
        index += 1
        if (index = a):
            startIndex := prev
        if (index = b):
            endIndex := x.next
            break;
    if (startIndex = NULL): return
    if (endIndex = NULL):
        startIndex.next = L.NIL
        return
    startIndex.next := endIndex
```

Für die Laufzeit machen wir eine Fallunterscheidung: 1. Fall (Worst Case): $b > n$, 2. Fall: $b < n$ 1. Fall: Der Algorithmus hat eine Laufzeit von $O(n)$ da unabhängig vom Eingabearray die While Schleife n-mal ausgeführt wird, damit alle Indizes einmal abgeprüft werden können. 2. Fall: Bei $b < n$ kann der While-Loop früher abgebrochen werden. Die Laufzeit ist nun abhängig von der Eingabe von b, denn die While Schleife kann früher verlassen werden, sobald der Index b erreicht wurde. Die Laufzeit ist somit O(b). 3. Fall: Falls $a > b$ ist die Laufzeit O(1)