AD [HA] zum 20. 11. 2013

Arne Struck, Lars Thoms

20. November 2013

1. (a)

$11 \cdot \mathbb{N} + 10$. Da für jeden Elferzyklus die 11. Stelle gesucht ist müssen nach dem Zyklus noch 10 addiert werden.

(b)

$11 \cdot \mathbb{N} + 5$. Das Ergebnis entsteht aus der Tatsache, dass nun $2k \mod 11$ gilt, daher muss der Ausdruck aus (a) auch durch 2 geteilt werden, leider führt das dazu, dass jedes 2. Element aus der Ergebnismenge gestrichen wird ($\frac{11}{2}$ ist kein ganzes Vielfaches von 11). Somit muss $\frac{11}{2}$ durch 11 ersetzt werden.

(c)

$11 \cdot \mathbb{N}$. Da die Operation 10 zu addieren (um an die 11. Stelle des Zyklus zu gelangen) schon erfolgt ist, muss nur noch ein Vielfaches von 11 übergeben werden.

(d)

Wenn Elemente existieren für die $3^k - 1 \mod 11 \equiv 10$ gilt, müssen Elemente existieren für die gilt: $3^k \mod 11 \equiv 0$.

$$
\begin{array}{lllllll}
3^0 & = & 1 & & & & \\
3^1 & = & 3 & & & & \\
3^2 & = & 9 & & & & \\
3^3 & = & 27 \mod 11 & \equiv & 5 & & \\
3^4 & \equiv & 5 \cdot 3 & = & 15 \mod 11 & \equiv & 4 \\
3^5 & \equiv & 3 \cdot 4 & = & 12 \mod 11 & \equiv & 1
\end{array}
$$

Hiermit kommt man in einen Zyklus, das bedeutet es existiert kein Element auf der 11. Stelle mit der Funktion $3^k - 1 \mod 11$. Daher ist das Ergebnis $\emptyset$.

2.

$\lim_{n\to\infty}(\frac{n^n}{n!}) > 1 \Rightarrow$ n! liegt in $\mathcal{O}(n^n)$

$\lim_{n\to\infty}(\frac{\frac{n}{2}^{\frac{n}{2}}}{n!}) = \lim_{n\to\infty}(\frac{\frac{n}{2}\cdot\frac{n}{2}\cdot\frac{n}{2}\cdot\ldots}{n\cdot(n-1)\cdot(n-2)\cdot\ldots} \cdot \frac{\frac{n}{2}}{n-\lceil\frac{n}{2}\rceil} \cdot \frac{1}{(n-\lceil\frac{n}{2}\rceil-1)\cdot(n-\lceil\frac{n}{2}\rceil-2)\cdot\ldots\cdot 1}) < 1 \Rightarrow$ n! liegt in $\Omega(\frac{n}{2}^{\frac{n}{2}})$

Daraus folgen die Grenzen:

$\frac{1}{2} = \lim_{n\to\infty}(\frac{1}{2}\frac{\log\frac{n}{2}}{\log n}) = \lim_{n\to\infty}(\frac{\frac{n}{2}\log\frac{n}{2}}{n\log n}) = \lim_{n\to\infty}(\frac{\log(\frac{n}{2}^{\frac{n}{2}})}{n\log n}) < \lim_{n\to\infty}(\frac{\log(n!)}{n\log n}) < \lim_{n\to\infty}(\frac{\log n^n}{n\log n}) = 1$

Damit liegt ein konstanter Grenzwert von $\log n!$ vor. Da in der Landau-Notation konstante Faktoren wegfallen gilt die Behauptung.

3. (a)

Um den Median als Pivot zu verwenden, muss dieser erst einmal gefunden werden. Dies ist in O(n) möglich (Median of Medians). Ist das Pivot-Element immer der Median folgt daraus, dass immer der Best-case von Quicksort eintritt $\Theta(n \cdot \log(n))$. Damit ergibt sich:

$$T(n) = 2 \cdot T(\tfrac{n}{2}) + a \cdot n$$

$a \cdot n$ ist die Zeit zum auffinden des Medians. Aus dem Master-Theorem folgt: $\Theta(n \cdot \log(n))$

(b)

Diese Variante findet in der Praxis vermutlich wenig Anwendung, da die Konstante a beim Auffinden des Medians sehr groß sein kann, was eine größere Laufzeit als den average Case des Random-Pivots zur Folge haben könnte. Außerdem könnte die Berechnung des Medians den Speicher weitaus stärker belasten, als ein randomisierter Pivot.

4. (a)

```
function chooseRandomElement(A)
    n = length(A)
    if n == 1 then
        return A[0]
    end if
    if random() == 0 then
        return chooseRandomElement(slice(A, 0, n/2))
    else
        return chooseRandomElement(slice(A, n/2, n))
    end if
end function
```

A sei ein input der Länge 2^k. Damit wird rekursiv ein Blatt in einem vollständigen Binärbaum gefunden. Die Binärbaumsuche hat einen Zeitaufwand von O(log n). Da alle Blätter die gleiche Anzahl an Kanten brauchen um erreicht zu werden, besitzen auch alle die gleiche Wahrscheinlichkeit.

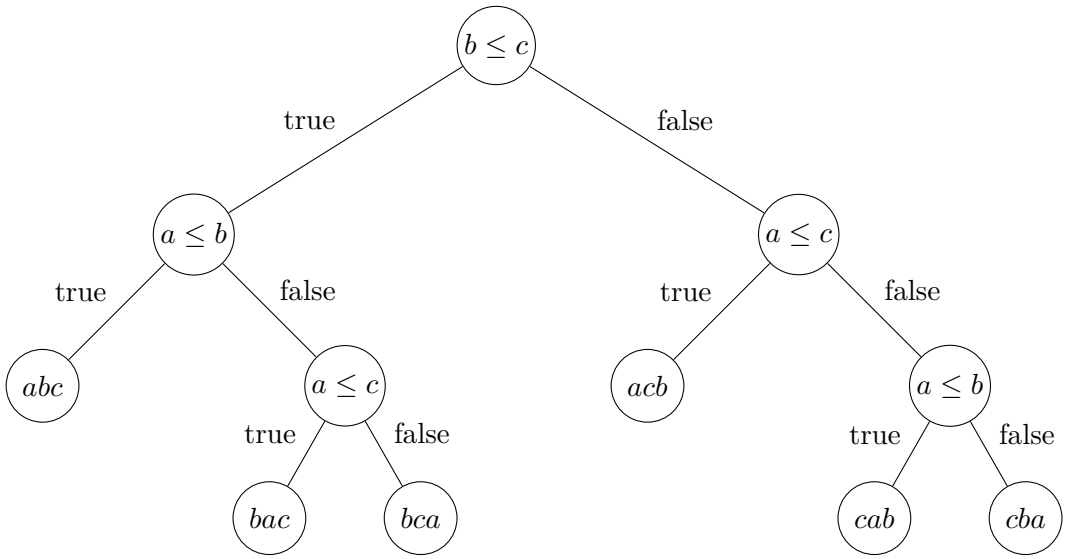
(b)

Die Lösung von a) funktioniert auch für b). Der Baum ist nicht mehr vollständig, was dazu führt, dass einige Blätter schneller als andere zu erreichen sind. Damit sind die Wahrscheinlichkeiten nicht ganz die selben, was die Aufgabe erlaubt.

(c)

Wir bilden einen vollständigen binären Baum mit m Elementen, wobei m sich durch aufrunden von n auf die nächste Zweierpotenz (2^k) ergibt. Blätter für die keine Elemente mehr in A vorhanden sind (Überschuss) werden markiert. Es folgt der Algorithmus aus a), ist das gefundene Blatt markiert, wird das ganze wiederholt.
Im worst-case (n ist eine Zweierpotenz plus eins) sind $\frac{n}{2} - 1$ Blätter markiert. Bis zum ersten Auffinden eines Elementes von A werden deswegen 2 Versuche erwartet. Die worst-case Laufzeit liegt also im Erwartungswert bei $\mathcal{O}2\log(2n) \in \mathcal{O}(\log(n))$. Die best-case Laufzeit (n ist eine Zweierpotenz) beträgt $\mathcal{O}(\log(n))$ (siehe a)). Somit muss auch die average-case Laufzeit in $\mathcal{O}(\log(n))$ liegen.

5. (a)



(b)

Bei einer Eingabe von 4 Elementen erhielte der Baum 4! Blätter, bei einer Eingabe von 26 erhielte er 26! Blätter.

6.

Im ersten Schritt wird ein Bucketsort mit 2 Buckets durchgeführt. Der erste Bucket geht bis exklusiv k und der zweite geht bis n. Nach dem Auftrennen wird ein einfacher Mergesort aufgerufen, welches eine sortierte Liste ab k zurückgibt. Dadurch kommt eine Laufzeit von $\mathcal{O}(n \log k)$ zustande.

```
GetSortedMins(A, k)
{
    var new_A;

    foreach(A as item)
    {
        if(item >= k)
        {
            new_A.insert(item);
        }
    }

    return quicksort(new_A);
}
```