Grundlegende Algorithmen

Abgabe: bis 7. November, 16:00 Uhr, Briefkasten bei S0314

Aufgabe 1 (10 Punkte)

Entwerfen Sie ein RAM Programm für die Funktion

$$\begin{array}{rcl} f : \mathbf{N} & \longrightarrow & \mathbf{N} \\ n & \longmapsto & \left\lfloor \frac{n}{3} \right\rfloor \end{array}$$

und geben Sie die uniforme Zeitkomplexität an. Beschreiben Sie die Arbeitsweise Ihres Programms in Pseudocode! Testen Sie Ihr Programm mit den Eingaben $5, 7, 9$. Hinweis: Sie benötigen nur $+, -$ und keine Division/ Multiplikation.

Aufgabe 2 (10 Punkte)

(a) Bestimmen Sie die Anzahl der Vergleiche (zwischen Elementen des Arrays) $T(n)$, die BinSearch auf eine Eingabe der Länge $n = 2^k$ benötig. Versuchen Sie hierzu, eine Rekursionsformel für $T(n)$ aufzustellen (Hinweis: $T(1) = 1$).

Algorithmus 1: BinSearch
Eingabe: aufsteigend sortiertes nichtleeres Array $A[]$ der Länge n, Element a
Ausgabe: `true` falls $a \in A[]$, `false` sonst
(1) **if** n == 1 **then** return($A[1] == a$)
(2) $m := \lfloor \frac{n}{2} \rfloor$;
(3) **if** $a \leq A[m]$ **then** return(BinSearch($A[1..m], a$))
(4) **else** return(BinSearch($A[m+1..n], a$))

(b) Modifieren Sie InsertionSortRec so, dass $O(n \log n)$ Vergleiche zwischen den Elementen benötigt werden. Führen Sie den Algorithmus mit der Eingabe $[7, 5, 6, 1, 3, 9]$ aus.

Eingabe: Array A mit n Elementen
Ausgabe: A aufsteigend sortiert
InsertionSortRec($(A[])$)

if $n = 1$ **then return** (A_1)
else
  $A' :=$InsertionSortRec($(A[1..n-1])$);
  while $i \leq n-1$ and $A[n] < A'[i]$ **do** i++ **endwhile**
  $A'[]$ ab Position i um 1 nach rechts schieben
  $A'[i] := A[n]$
  return $(A'[])$
end

Aufgabe 3 (10 Punkte)

Entwerfen Sie einen rekursiven oder iterativen Algorithmus (Pseudocode) der die Funktion

$$\begin{aligned} f : \mathbf{N} &\longrightarrow \mathbf{N} \\ n &\longmapsto 2^{2^n} \end{aligned}$$

berechnet, ohne die Funktion $\texttt{exp}(a,b) = a^b$ zu verwenden. Begründen Sie die Korrekteit und machen Sie eine möglichst gute Abschätzung der Laufzeit (mit $O()$) von Ihrem Algorithmus. Hinweis zum rekursiven Algorithmus: Überlegen Sie Sich, wie man von $2^{2^{n-1}}$ auf 2^{2^n} kommt.