Musterlösung zur Vordiplomklausur Datenstrukturen und Programmierverfahren Sommersemester 1999

1. (a)

	Hash-Tabelle der Größe m mit Kollisionslisten	einfach verkette Liste	Rot-Schwarzbaum	Binary Heap
Search(S,k)	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$
Insert(S,x)	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$
Delete(S,x)	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$
Minimum(S)	$\Theta(n)$	$\Theta(n)$	$\Theta(\lg n)$	$\Theta(n)$
Maximum(S)	$\Theta(n)$	$\Theta(n)$	$\Theta(\lg n)$	$\Theta(1)$
Successor(S,x)	$\Theta(n)$	$\Theta(n)$	$\Theta(1)$	$\Theta(1)$
Predecessor(S,x)	$\Theta(n)$	$\Theta(n)$	$\Theta(1)$	$\Theta(1)$

(b)

	Hash-Tabelle der Größe m mit Kollisionslisten	einfach verkette Liste	Rot-Schwarzbaum	Binary Heap
Search(S,k)	$\Theta(1)$	$\Theta(n)$	$\Theta(\lg n)$	$\Theta(n)$
Insert(S,x)	$\Theta(1)$	$\Theta(1)$	$\Theta(\lg n)$	$\Theta(\lg n)$
Delete(S,x)	$\Theta(1)$	$\Theta(n)$	$\Theta(\lg n)$	$\Theta(\lg n)$
Minimum(S)	$\Theta(n+m)$	$\Theta(n)$	$\Theta(\lg n)$	$\Theta(n)$
Maximum(S)	$\Theta(n+m)$	$\Theta(n)$	$\Theta(\lg n)$	$\Theta(1)$
Successor(S,x)	$\Theta(n+m)$	$\Theta(n)$	$\Theta(\lg n)$	$\Theta(n)$
Predecessor(S,x)	$\Theta(n+m)$	$\Theta(n)$	$\Theta(\lg n)$	$\Theta(n)$

2. (a) Z.z.: $\exists c, n_0$ mit $c > 0$, so daß für alle $n \geq n_0$ gilt: $0 \leq 2^{n+1} \leq c2^n$ Für $c = 2, n = 1$ sind beide Ungleichungen offensichtlich erfüllt.

   (b) Gegenbeispiel: $f(n) = 2n$ und $g(n) = n$

   (c) Gegenbeispiel: $f(n) = n$

3. (a) $a = 3, b = 2, \log_b a = 1 + \epsilon$
   Fall 1 des MT: $T(n) = \Theta(n \log_2 3)$

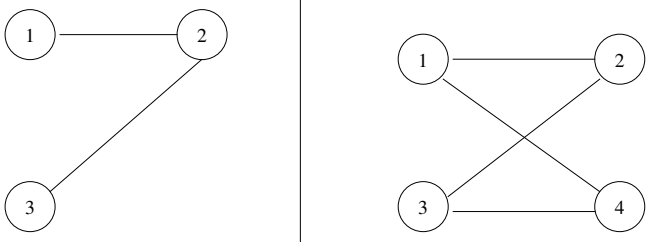
   (b) $a = 1, b = 3/2, \log_b a = 0$
   Fall 3 des MT: $T(n) = n$

   (c) $a = 2, b = 2, log_b a = 1$
   MT kann nicht angewendet werden. Anwenden der Iterationsmethode:

$$\begin{aligned} T(n) &= 2T(n/2) + n \lg n \\ &= n \lg n + 2n/2 \lg n/2 + \ldots + 2^{\lg n - 1} \frac{n}{2^{\lg n - 1}} \lg \frac{n}{2^{\lg n - 1}} \end{aligned}$$

$$
\begin{aligned}
&= n \sum_{i=0}^{\lg n - 1} \lg n - 2^i \\
&= n \lg^2 n - \frac{n-1}{2-2} \\
&= \Theta(n \lg^2 n)
\end{aligned}
$$

4. (a)



(b) i. nein

ii. ja

iii. ja

5.

S	d[s]	π[s]	d[u]	π[u]	d[v]	π[v]	d[w]	π[w]	d[x]	π[x]
	0	nil	∞	nil	∞	nil	∞	nil	∞	nil
{ s }	0	nil	7	s	3	s	∞	nil	∞	nil
{ s,v }	0	nil	5	v	3	s	8	v	6	v
{ s,v, u }	0	nil	5	v	3	s	8	v	6	v
{ s, v, u, x }	0	nil	5	v	3	s	7	x	6	v
{ s, v, u, x, w }	0	nil	5	v	3	s	7	x	6	v

6. Idee: Pfad zwischen u und v bestimmen. Im Pfad die Kante mit maximalen Gewicht bestimmen. Falls deren Gewicht größer ist als das Gewicht der neu einzufügenden Kante, wird diese Kante im MST T' ersetzt.

MaintainMST$(G, T, (u, v, w_e))$
  $G^T := (V, T)$;
  $T' := T$;
  BFS(G^T, u);
  $u_m := \pi[v]$;
  $v_m := v$;
  $w_m := w(\pi[v], v)$;
  $c := \pi[v]$;

while $\pi[c] \neq$nil **do**
  if $w_m \leq w(\pi[c], c)$ **then**
    $u_m := \pi[c]$;
    $v_m := c$;
    $w_m := w(\pi[c], c)$;
  endif
  $c := \pi[c]$;
endwhile
if $w_e < w_m$ **then**
  $T' := T' \setminus \{(u_m, v_m, w_m)\} \cup \{(u, v, w_e)\}$;
endif
return T';

7. Idee: Schleife über das Array A. In der Schleife wird jedes zweite Element angesprungen. Vergleich des angesprungen Elements mit dem Nachfolgerelement zur Bestimmung eines lokalen Minimums und Maximums. Vergleich des lokalen Minimums mit dem globalen Minimum. Gegebenenfalls globales Minimum durch lokales Minimum ersetzen. Bezüglich des Maximums entsprechend verfahren. Es werden $n/2$ Iterationen durchgeführt. Pro Iteration werden drei Vergleiche durchgeführt.

   Sei length$[A] > 0$ und gerade.

   MinMax(A)
     $n :=$length$[A]$;
     min:= 0;
     max:= 1;
     $i := 0$;
     while $i \leq n - 2$ **do**
       if $A[i] < A[i + 1]$ **then**
         $\min_r := i$;
         $\max_r := i + 1$;
       else
         $\min_r := i + 1$;
         $\max_r := i$;
       endif
       if $A[\min_r] < A[\min]$ **then**
         min := $\min_r$;
       endif
       if $A[\max_r] > A[\max]$ **then**
         max := $\max_r$;
       endif
       $i := i + 2$;
     endwhile
     return (min,max);

8. Idee: Greedy-Algorithmus verwenden. Reelle Zahlen aufsteigend sortieren. Durch das Array laufen. Falls man auf eine Zahl trifft, die nicht im zuletzt gebildeten Intervall liegt, neues Intervall bilden, dessen unterer Grenze durch die betrachtete Zahl gebildet wird.

   (a) Greedy-Algorithmus

   ```
   Interval(R)
     Sort(R);
     r := R[0];
     I := {[r, r + 1];
     for i := 1 to length[R] − 1 do
       if R[i] > r + 1 then
         r := R[i];
         I := I ∪ {[r, r + 1];
       endif
     endfor
     return I;
   ```

   Laufzeit: $\Theta(n \lg n)$

   (b) Theorem: Für alle k existiert eine optimale Lösung, die mit k Greedy-Choices beginnt.

   Beweis per Induktion über k.

   Basis ($k = 1$):
   Es existiert eine optimale Lösung, beginnend mit der Greedy-Choice.

   Beweis:
   Sei $G = \{g_1, g_2, \ldots\}$ die geordnete Sequenz von Intervallen in der Greedy-Lösung. Sei $A = \{a_1, a_2, \ldots\}$ eine optimale Lösung. Falls $a_1 = g_1$ beginnt die optimale Lösung mit einer Greedy-Choice. Falls $a_1 \neq g_1$ kann a_1 in A durch g_1 ersetzt werden, ohne die Optimalität von A zu gefährden.

   Schluß ($k \rightarrow k + 1$):
   Falls eine Lösung existiert, die mit k Greedy-Choices beginnt, dann existiert eine mit $k + 1$.

   Beweis:
   Sei $G = \{g_1, g_2, \ldots, g_k, g_{k+1}, \ldots\}$ die geordnete Sequenz von Intervallen in der Greedy-Lösung. Sei $A = \{a_1, a_2, \ldots, a_k, a_{k+1}\}$ eine optimale Lösung, die mit k Greedy-Choices beginnt. Falls $a_{k+1} = g_{k+1}$ beginnt die optimale Lösung mit $k + 1$ Greedy-Choices. Falls $a_{k+1} \neq g_{k+1}$, kann a_{k+1} in A durch g_{k+1} ersetzt werden, ohne die Optimalität von A zu gefährden.