Zu Übung 06, Besprechung: Do., 19.07.2018, 14^{00} – Neue Chemie

Aufgabe 7.01: Verständnisfragen

a) Ein Algorithmus ist eine genau definierte Handlungsvorschrift zur Lösung eines Problems oder einer bestimmten Art von Problemen in _____ vielen Schritten.

b) Ein Algorithmus kann mit _____ oder _____ beschrieben werden.

c) Dynamische Finitheit besagt, dass das Verfahren zu jedem Zeitpunkt nur endlich viel _____ benötigen darf.

d) Komplexitätstheorie bezeichnet das Verhalten von Algorithmen bezüglich Ressourcenbedarf, wie _____ und _____.

e) Berechenbarkeitstheorie besagt, dass:

i. der Algorithmus bei denselben Voraussetzungen das gleiche Ergebnis liefern muss.

ii. jeder Schritt des Verfahrens tatsächlich ausführbar sein muss.

iii. das Verhalten bezüglich der Terminierung (ob der Algorithmus überhaupt jemals erfolgreich beendet werden kann) erfüllt sein soll.

f) Merge Sort ist ein Sortieralgorithmus, der auf dem Prinzip _____ basiert.

g) Der Quicksortalgorithmus benötigt, abgesehen von dem für die Rekursion benötigten Platz auf dem Aufruf-Stack, keinen zusätzlichen Speicherplatz. Richtig / Falsch

h) Die Tiefensuche ist geeignet, um Eigenschaften von allen Knoten in einem Graphen auf dem Bildschirm auszugeben. Richtig/Falsch

i) Nennen Sie vier verschiedene Suchalgorithmen.

j) Die Laufzeit des Quicksortalgorithmus ist abhängig von
_____

k) Wann heißen zwei Graphen G1 und G2 isomorph?

l) Welche zusätzlichen Eigenschaften muss ein Graph erfüllen, damit er ein Baum ist?

Aufgabe 7.02: Arrays und Sortieren

Schreiben Sie ein Programm, welches mit Hilfe eines InsertionSort-Algorithmus ein 1-dimensionales Array in aufsteigender Reihenfolge sortiert. Testen Sie anschließend Ihr Programm mit geeigneten Testarrays.

Der folgende Pseudocode beschreibt den InsertionSort-Algorithmus,

Hinweis: Beachten Sie, dass bei C++ ein Array an der Stelle 0 beginnt, dies ist nicht im Pseudocode berücksichtigt.

```
1 for j = 2 to length[A]
2 do key = A[j]
  //Füge A[j] in die sortierte Folge A[1 .. j – 1] ein
3    i = j – 1
4    while( i > 0 and A[i] > key )
5    do A[i + 1] = A[i]
6       i = i – 1
7    A[i + 1] = key
```

Aufgabe 7.03: Laufzeitanalyse von Insertion Sort

In dieser Übung soll die Laufzeit vom "Sortieren durch Einfügen" bzw. Insertion Sort jeweils für den worst- und bestcase bestimmt werden. Ermitteln Sie anhand des nachstehenden Pseudo-Codes die Gesamtlaufzeit *T(n)* in beiden Fällen und bestimmen Sie jeweils eine obere asymptotische Schranke! Nehmen Sie dabei an, dass das zu sortierende Array *A* die Länge *n* hat. Noch zu beachten ist, dass c_i die Kosten der Codezeile *i* darstellt.

```
INSERTIONSORT( A )
1 for j = 2 to length[A]
2 do key = A[j]
  //Füge A[j] in die sortierte Folge A[1 .. j – 1] ein
3    i = j – 1
4    while( i > 0 and A[i] > key )
5    do A[i + 1] = A[i]
6       i = i – 1
7    A[i + 1] = key
```

Aufgabe 7.04: Tiefensuche

Der folgende Pseudocode beschreibt die Tiefensuche:

```
DepthFirstSearch( G )
  for alle Knoten u in G
  do farbe[u] = weiss
     vater[u] = NIL
  zeit = 0
  for alle Knoten u in G
  do if farbe[u] == weiss
     then DFS-VISIT( u )

DFS-VISIT( u )
  farbe[u] = grau; zeit = zeit + 1
  startTime[u] = zeit
  for alle Knoten v aus Adj[u]
  do if farbe[v] == weiss
     then vater[v] = u
          DFS-VISIT( v )
  farbe[u] = schwarz; zeit = zeit + 1
  endTime[u] = zeit
```

Gegeben ist die folgende Adjazenzmatrix. Wenden Sie darauf den Algorithmus der Tiefensuche an und zeichnen Sie den daraus resultierenden Suchbaum (Startknoten A). Markieren Sie dabei (mit einer fortlaufenden Zahl) im jeweiligen Knoten den Zeitpunkt des Entdeckens (**`startTime`** im Algorithmus) des Knotens und den Zeitpunkt, wann der Knoten vollständig bearbeitet worden ist (**`endTime`** im Algorithmus). Knoten mit einem kleineren Buchstaben (A ist kleiner als B) werden dabei zuerst gefunden.

	A	B	C	D	E	F	G	H	I	J	K
A	0	1	0	0	1	0	0	0	0	0	0
B	0	0	1	1	0	0	0	0	0	0	0
C	1	0	0	0	0	0	0	0	0	0	0
D	0	0	1	0	0	0	0	0	0	0	0
E	0	1	0	0	0	0	0	0	0	0	0
F	0	0	0	0	0	0	1	0	0	1	1
G	0	0	0	0	1	0	0	1	1	0	0
H	0	0	0	0	0	0	1	0	0	0	0
I	0	0	0	0	0	0	0	1	0	0	0
J	0	0	0	0	0	0	0	0	0	0	1
K	0	0	0	0	0	0	1	0	0	0	0