Abschlussprüfung Sommer 2010
Lösungshinweise

IHK

Mathematisch-technischer Softwareentwickler
Mathematisch-technische Softwareentwicklerin
6511

2 Softwareentwurf und Programmierung

Allgemeine Korrekturhinweise

Die Lösungs- und Bewertungshinweise zu den einzelnen Aufgaben sind nicht in jedem Fall Musterlösungen, sondern als Korrekturhilfen zu verstehen. Sie sollen nur den Rahmen der zu erwartenden Prüfungsleistungen abstecken. Der Bewertungsspielraum des Korrektors (z. B. hinsichtlich der Berücksichtigung regionaler, branchen- oder betriebsspezifischer Gegebenheiten) bleibt unberührt.

Zu beachten ist die unterschiedliche Dimension der Aufgabenstellung (nennen – erklären – beschreiben – usw.).

Für die Bewertung gilt folgender Punkte-Noten-Schlüssel:

Note 1	=	100 – 92 Punkte	Note 2	= unter	92 – 81 Punkte
Note 3	= unter	81 – 67 Punkte	Note 4	= unter	67 – 50 Punkte
Note 5	= unter	50 – 30 Punkte	Note 6	= unter	30 – 0 Punkte

1. Aufgabe (60 Punkte)

a) Es wird die Anzahl auftretender „0" gezählt, bis eine „1" erscheint. Die Zahl wird notiert. Dann wird die Anzahl auftretender „1" gezählt, bis eine „0" erscheint. Diese Zahl wird als nächstes notiert. Dies wird bis zum Ende der Zahlenfolge durchgeführt. Es ist wichtig, dass mit der Anzahl der auftretenden „0" begonnen wird, damit die Dekomprimierung definiert starten kann.

aa) 1 23 1 — 10 Punkte

ab) 01111111111111111111111111
In diesem Fall ist die komprimierte Version ein Zeichen länger als das Original. — 10 Punkte

ba) Die einfachste Art, sie abzulegen, ist ein Array der Länge 86.400 (das ist die Anzahl der Sekunden eines Tages) Es wird zur Ablage ein beliebiger Datentyp gewählt. Das bedeutet, dass die Datengröße mindestens 86.400 Byte ist.
Begründung: Schneller Zugriff

Da nur die Zahlen 0 und 1 auftreten, ist eine optimierte Speicherform möglich. Es können immer acht Zahlen zu einem Byte zusammengefasst werden, was die Größe des Array auf 86.400/8 und damit die Datengröße reduziert.
Begründung: Speichereffizienz bei etwas komplexerem Zugriff — 10 Punkte

bb) Die größte mögliche Anzahl ist die 86.401 (siehe ab)). Daher ist ein Type 4-Byte Integer nötig. Da die Anzahl der Wechsel nicht vorherzusehen ist und die Daten in unserer Aufgabe sequentiell verarbeitet werden, ist z. B. hier eine verkettete Liste von Variablen des Typs Longint möglich.
Begründung: Speichereffizienz

Hinweis: Jede Ablage in Form von Strings impliziert auch immer die Nutzung eines Trennzeichens, da z. B. die „123" nicht eindeutig interpretierbar ist. Dies vergößert den Speicherplatz. — 10 Punkte

c) Komprimierung
Prüfe ein Zeichen (je nach Datentyp z. B. erste Zahl oder erstes Bit in Byte
- wenn es keine „0" ist, dann speichere im Zieldatentyp eine 0
- Starte den Algorithmus:

Suche die Wechsel und speichere die bis jeweils zum Wechsel ermittelte Anzahl in den Zieldaten. Verfahre so bis zum Ende der Tagesdaten.

Antwort: Dekomprimierung
Die Zieldaten enthalten die Anzahl des Auftretens der Zustände „0" und „1" im Wechsel. Es beginnt mit der Anzahl der „0".

Zustand = 0
solange Anzahl in Zieldatenspeicher
Lese Anzahl aus Zieldatenspeicher
Wenn Anzahl>0 dann Schleife von 1 bis Anzahl Erzeuge Zustand in Originaldaten
Zustand = 1 – Zustand
ende solange — 20 Punkte

2. Aufgabe (20 Punkte)

a) Mögliche Algorithmen mit der Komplexität O(n*log(n)) sind Quicksort, Mergesort u. a.

Mögliche Beschreibung eines Algorithmus am Beispiel Quicksort:

- Rekursiver Algorithmus
- Überprüfen des Abbruchkriteriums
- Wahl eines Pivotelements, verschiedene Möglichkeiten der Wahl sind möglich
- Vorsortieren der Elemente in Abhängigkeit des Pivotelements
- Evtl. Pivotelement an die korrekte Stelle tauschen (Abh. Von der Wahl des Pivotelements und der Variante)
- Rekursiver Aufruf des Verfahrens mit den Teillisten

Abweichungen sind je nach Variante des Verfahrens möglich. 10 Punkte

b) Nachfolgende Variante mit jeweils rechtem Element einer (Teil)Liste als Pivotelement.

Zahlenfolge										***Beschreibung***
29	14	24	3	30	3	20	4	7	**8**	Wahl des Pivotelements
7	4	3	3	30	24	20	14	29	**8**	Vorsortieren der Liste
7	4	3	3	**8**	24	20	14	29	30	Einsortieren des Pivotelements
7	4	3	**3**	8	24	20	14	29	**30**	Rekursiver Aufruf der Teillisten
3	4	7	**3**	8	24	20	14	29	30	Wahl des Pivot der Teilliste
3	3	7	4							Einsortieren des Pivotelements
3	3	7	4							Liste mit einem Element ist sortiert!
3	3	7	**4**	8						Wahl des Pivotelement
3	3	**4**	7	8						Einsortieren des Pivotelements
3	3	4	7	8	24	20	14	29	**30**	Wahl des Pivot der rechten Teilliste
					24	20	14	29	**30**	Vorsortierung ergab, dass Pivot am richtigen Platz ist. Rek. Aufruf des Rests
					24	20	14	**29**	30	Wahl des Pivotelements, wie zuvor!
							14	29	30	Wahl des Pivotelements
					14	20	24			Einsortieren des Pivotelements
						20	**24**			Rekursiver Aufruf der rechten Liste
3	3	4	7	8	14	**20**	24	29	30	Rekursiver Aufruf der linken einelementigen Liste

Abweichungen in der Durchführung sind möglich. Die Beschreibung der Schritte ist nicht Bestandteil der Aufgabe und dient einzig dem Überblick der Musterlösung. 10 Punkte

3. Aufgabe (20 Punkte)

Der Bisektionsalgorithmus besteht für jedes Intervall [a, b] im Wesentlichen aus den Schritten

(i). c : = (a + b)/2

(ii). Ist (b – a) nah genug an 0?

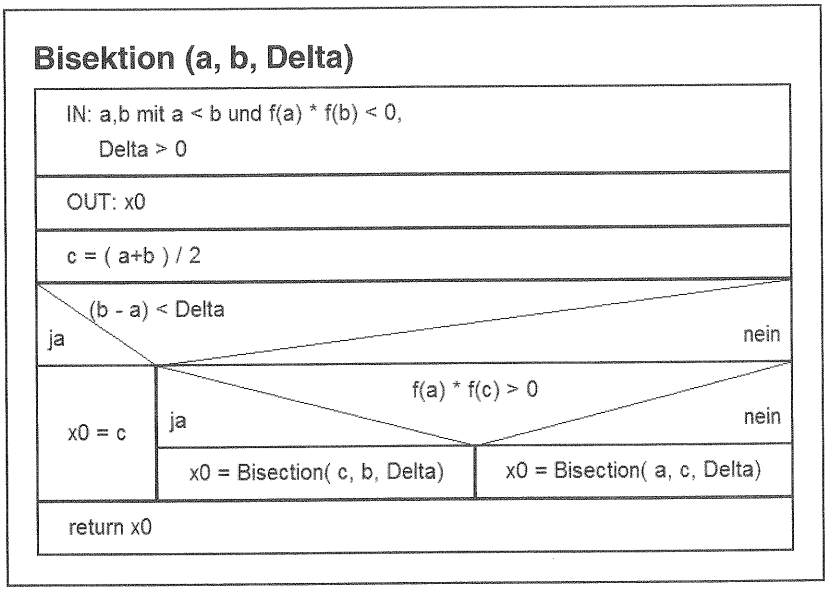
(iii). In welcher Intervallhälfte muss weitergesucht werden?

Dies ist eine klassische Rekursion, wobei Punkt (iii) die nächste Rekursion einleitet und Punkt (ii) den Abbruch der Rekursion garantieren soll. Dabei reicht es nicht aus nur den Funktionswert f(c) < Delta zu betrachten.

Formal kann dies wie folgt ausgedrückt werden:

$$x_0 := Bisection(a,b,\delta) := \begin{cases} c := \frac{(a+b)}{2} \; falls \; |b-a| < \delta \\ Bisection(c,b,\delta) sonst, falls \; f(a) \cdot f(c) > 0 \\ Bisection(a,c,\delta) sonst, falls \; f(a) \cdot f(c) < 0 \end{cases}$$

Daraus ergibt sich das folgende Struktogramm für die rekursive Lösung (Bei dieser Lösung sind jedoch keine Sonderfälle berücksichtigt!):



Alternativ ist die iterative Lösung wie rechts abgebildet (Auch hier sind keine Sonderfälle berücksichtigt!):