Prof. Dr. Gerd Stumme, Folke Eisterlehner, Dominik Benz
Fachgebiet Wissensverarbeitung

02.06.2009

7. Hausübung „Algorithmen und Datenstrukturen“

Sommersemester 2009

Abgabetermin: Dienstag, 09.06.2009, 10:00 Uhr

1 Algorithmenbegriff

(*A*) Was ist ein *Algorithmus* gemäß der Vorlesung? (3 Punkte)

(*B*) Für die Suche nach einem Schlüssel in einem unsortierten Feld wird folgender Algorithmus verwendet:

Algorithm `SillySearch` $(F, k) \rightarrow i$

Eingabe: Unsortiertes Feld F der Länge $n > 0$, gesuchter Schlüssel k

Ausgabe: Index i mit $F[i] = k$ falls ein solcher existiert und `NO_KEY` sonst

1) Wähle zufällig zwei ganze Zahlen $1 \leq i_1 \leq i_2 \leq n$
2) Sortiere $A \coloneqq F[1], \ldots, F[i_1 - 1]$ mittels *BubbleSort*
3) Sortiere $B \coloneqq F[i_1], \ldots, F[i_2 - 1]$ mittels *QuickSort*
4) Sortiere $C \coloneqq F[i_2], \ldots, F[n]$ mittels *MergeSort*
5) Wähle zufällig einen der folgenden Teilschritte *a*), *b*) oder *c*) aus:
   a) Vertausche die Reihenfolge der Teilfelder A, B und C in F zu $F = A, C, B$
   b) Vertausche die Reihenfolge der Teilfelder A, B und C in F zu $F = C, B, A$
   c) Vertausche die Reihenfolge der Teilfelder A, B und C in F zu $F = B, A, C$
6) Starte den Algorithmus `BinarySearch` (wie in der Vorlesung auf Folie 3-8 beschrieben) auf dem so vertauschten Feld und gib dessen Rückgabewert als Ergebnis aus

Beantworten Sie folgende Fragen:

- Findet `SillySearch` in jedem Fall einen im Eingabefeld F enthaltenen Schlüssel k?
- Ist dieser Algorithmus
  - deterministisch (im Ablauf)?
  - determiniert (im Ergebnis)?
  - terminierend?

Begründen Sie jeweils kurz.

(6 Punkte)

2 Komplexität

(*A*) Nennen Sie drei Gründe, warum ein Algorithmus mit der best–case Laufzeit $O(n^2)$ bei Laufzeitmessungen auf einer konkreten Eingabe besser abschneiden kann als ein Algorithmus mit der best–case Laufzeit $O(n)$. Beide Algorithmen sind in der gleichen Programmiersprache implementiert und die Messungen werden unter identischen Rahmenbedingungen durchgeführt. (3 Punkte)

(*B*) Betrachten Sie folgende Programmausschnitte und ordnen Sie diesen in der nachfolgenden Tabelle jeweils alle korrekten Oberen Schranken in O–Notation zu (pro Programmausschnitt kann mehr als eine obere Schranke korrekt sein). Für `println` ist dabei konstanter Zeitaufwand anzusetzen: (5 Punkte)

Programm **A**:
```
for( int i=0; i<n; i++ ) {
   if( i%2==0 )
      for( int j=0; j<n; j++ ) {
         System.out.println(i+j);
      }
}
```

Programm **B**:
```
for( int i=0; i<n; i++ ) {
   if( i%n==0 )
      for( int j=0; j<n; j++ ) {
         System.out.println(i+j);
      }
}
```

Programm **C**:
```
for( int i=1; i<n; i=i*2 ) {
    System.out.println(i);
}
```

Tragen Sie in folgender Tabelle jeweils „**+**“ für „trifft zu“ und „**-**“ für „trifft nicht zu“ ein. Kein Eintrag wird als Fehler gewertet.

Programm	$O(1)$	$O(\log(n))$	$O(\sqrt{n})$	$O(n)$	$O(n^2)$
A					
B					
C					

(*C*) Betrachten Sie wieder den Algorithmus **SillySearch** aus der vorhergehenden Aufgabe.

i) Welche Worst-Case-Komplexität hat der Algorithmus?

ii) Welche Best-Case-Komplexität hat der Algorithmus?

Begründen Sie Ihre Antworten kurz.

(4 Punkte)

3 Sortieralgorithmen

(*A*) Während des Sortiervorgangs einer Zahlenfolge ergibt sich folgender Ablauf. Welcher Sortieralgorithmus aus der Vorlesung oder Übung kam hier zum Einsatz?

(1 Punkte)

```
Eingabe: 7 3 6 8 1 2 5 4
    -->  1 7 3 6 8 2 4 5
    -->  1 3 6 7 2 4 5 8
    -->  1 2 3 6 7 4 5 8
    -->  1 2 3 6 4 5 7 8
    -->  1 2 3 4 6 5 7 8
Ausgabe: 1 2 3 4 5 6 7 8
```

(*B*) Folgende Zahlenfolgen sollen möglichst effizient sortiert werden:

F1: `1 2 4 3 5 7 6 9 10 11 10 14 13 12 15`

F2: `15 14 13 12 11 10 9 8 7 7 6 5 4 3 2 1 1`

F3: `1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 1`

- Welches der folgenden Sortierverfahren eignet sich besonders gut / besonders schlecht zum Sortieren der jeweiligen Folge? Markieren Sie in der folgenden Tabelle besonders gute Eignung mit `+`, besonders schlechte Eignung mit `-`, keine besondere Eignung mit `0` und begründen Sie jeweils kurz. Markieren Sie zudem, welches der Verfahren beim Sortieren in-place arbeitet und welches stabil ist (jeweils "ja" / "nein"). Kein Eintrag wird als Fehler gewertet.

	F1	**F2**	**F3**	**in-place?**	**stabil?**
InsertionSort					
SelectionSort					
BubbleSort					

(5 Punkte)

4 Korrektheit / Schleifeninvarianten

Welche der folgenden Aussagen über Schleifeninvarianten sind richtig / falsch? Begründen Sie jeweils kurz.

1. Eine Schleifeninvariante gilt während des gesamten Programmablaufs.

2. Eine Schleifeninvariante ist immer ein arithmetischer Ausdruck.

3. Es gibt ein klar definiertes Vorgehen, um Schleifeninvarianten zu finden.
4. Eine Schleifeninvariante hat in jedem Schleifendurchlauf einen konstanten numerischen Wert (z.B. 5).
5. Eine Schleifeninvariante beschreibt eine Eigenschaft einer Schleife, die unabhängig von der Anzahl der Durchläufe ist.
6. Die Schleifeninvariante gilt immer unmittelbar vor dem ersten Durchlauf der Schleife.
7. Eine Schleife wird genau dann verlassen, wenn die Schleifeninvariante nicht mehr gilt.
8. Für jede Schleife in einem Programm gibt es genau eine korrekte Schleifeninvariante.
9. Mit der Schleifeninvariante kann man immer die totale Korrektheit eines Programms nachweisen.
10. Die Anweisungen im Schleifenrumpf verändern den Wert der Schleifeninvarianten nicht.

(5 Punkte)

Viel Spaß und viel Erfolg!