Aufgabe 1 (Sortieren durch Vertauschen) ***30 Punkte***

Sei A ein Array der Länge n. A ist (aufsteigend) sortiert, wenn es *keine* zwei Positionen i und j (mit $i<j$) in A gibt, so dass $A[j] < A[i]$ ist. Damit kann man einen Sortieralgorithmus entwerfen, der auf der Idee basiert, wiederholt Positionen zu finden, die das obige Kriterium verletzen, und die Elemente an diesen Positionen dann zu vertauschen. (Wir nennen ein Paar solcher Indexwerte im folgenden *unsortierte Positionen* bzw. *unsortiertes Paar*.)

Es sei *findpair* ein Algorithmus, der zwei Indexpositionen i und j mit $i<j$ in einem Array A ermittelt, für die gilt: $A[j] < A[i]$. Zur einfacheren Beschreibung des nachfolgenden Sortieralgorithmus nehmen wir an, dass der Aufruf mit *findpair*(A, i, j) erfolgt und *true* ergibt, falls unsortierte Positionen existieren und *false*, falls nicht. Im ersten Fall werden die Positionen an die Variablen i und j übergeben.

(a) Geben Sie eine (möglichst große) untere Schranke für das Problem des Auffindens unsortierter Positionen im worst-case an. Beweisen Sie die untere Schranke, d.h. zeigen Sie, dass jeder mögliche Algorithmus *findpair* entsprechend viele Schritte benötigt. *10 Punkte*

(b) Geben Sie einen optimalen Algorithmus *findpair* an. Zeigen Sie die Optimalität. *10 Punkte*

Betrachten Sie nun den folgenden Sortieralgorithmus:

```
algorithm sort(A: array[1..n] of int)
while findpair(A, i, j) do
   x := A[j];
   A[j] := A[i];
   A[i] := x;
end while
```

(c) Angenommen, man weiß über *findpair* nur, dass es eine Laufzeit von $O(f(n))$ hat. Welche Laufzeit ergibt sich dann im worst-case für den Algorithmus *sort*? Geben Sie ein Beispiel für den worst-case an. *10 Punkte*

25 Punkte Aufgabe 2 (Verbessertes Quicksort)

Quicksort gilt als das im Durchschnitt schnellste Sortierverfahren, d.h. wenn Quicksort die Laufzeitschranke $O(n \cdot \log(n))$ erreicht, ist es um einen konstanten Faktor schneller als alle anderen Verfahren mit dieser Begrenzung. Leider garantiert Quicksort diese Laufzeitschranke nicht. Wie Ihnen bekannt ist, ist die Laufzeit im *worst case* $O(n^2)$, was sehr langsam gegenüber anderen Verfahren ist. Um den *worst case* auszuschließen, kann man (mindestens) zwei Ansätze verfolgen. Zum einen könnte man die Auswahl des Pivot-Elements so verbessern, dass die zu sortierende Menge in zwei möglichst **gleich große** Teilmengen geteilt wird. In dieser Aufgabe werden wir jedoch den zweiten, im Folgenden beschriebenen, Ansatz verfolgen.

Die Idee des verbesserten Quicksort ist die folgende. Wird ein pathologischer Fall erkannt, d.h. die Laufzeit von Quicksort ist quadratisch, wird auf ein alternatives Verfahren umgeschaltet, das eine Laufzeit von $O(n \cdot \log(n))$ auch im *worst case* garantiert. Nun stellt sich die Frage, wie ein solcher Fall erkannt werden kann. Ein Verfahren, dass genau diese Fälle erkennt, ist sicher zu aufwändig. Ein möglicher Anhaltspunkt ist die Rekursionstiefe des Quicksort-Algorithmus. Übersteigt diese einen gegebenen Schwellwert, so wird die verbleibende Teilfolge mittels des alternativen Sortierverfahrens sortiert.

2 Punkte (a) Welches aus dem Kurs bekannte Sortierverfahren eignet sich hier als Alternative, wenn das verbesserte Quicksort weiterhin in situ arbeiten soll?

11 Punkte (b) Sortieren Sie die nachstehende Folge mittels des verbesserten Quicksort-Verfahrens mit einem Schwellwert von 2 für die Rekursionstiefe (der Aufruf für die gesamte Folge hat Rekursionstiefe 1). Verwenden Sie zur Auswahl des Pivot-Elements die Funktion *findx* des Kurstextes. Geben Sie dazu den Aufrufbaum von Quicksort an (Ähnlich zur Abbildung „Mergesort-Implementierung mit zwei Arrays“ im Kurstext). Sie brauchen die Merge-Schritte nicht aufzuführen. Wenn auf den alternativen Algorithmus umgeschaltet wird, muss dieser nicht angegeben werden. Markieren Sie stattdessen die Knoten des Baums, bei denen „umgeschaltet“ wird.

1, 3, 5, 7, 9, 18, 23, 19, 14, 17, 9, 87, 2, 4, 6, 8

12 Punkte (c) Beweisen Sie die folgende Aussage:
Wird der Schwellwert mit $k \cdot \log(n)$ ($k > 1$ und konstant) festgelegt, so hat das verbesserte Quicksort-Verfahren eine *worst case*-Laufzeit von $O(n \cdot \log(n))$.

Aufgabe 3 (Radixsort) *20 Punkte*

Sortieren Sie die unten angegebenen Namen aufsteigend mittels Radixsort. Geben Sie dazu den Inhalt jedes nicht leeren Behälters nach jeder Phase an. Geben Sie anschließend auch die sortierte Folge an.

Thomas Nora Carla Ben Katrin Uwe Peter Ingo Anna Alex

Aufgabe 4 (Knotenbasis) *25 Punkte*

Eine *Knotenbasis* eines gerichteten Graphen $G = (V, E)$ ist eine Menge $B \subseteq V$ mit folgenden Eigenschaften:

1. Für alle $v \in V$ gibt es ein $b \in B$, so daß es einen Pfad von b nach v gibt.
2. B ist minimal, d.h. keine echte Teilmenge von B hat diese Eigenschaft.

(a) Beweisen Sie: Ein Knoten v, der nicht in einem Zyklus enthalten ist und dessen Eingangsgrad $\neq 0$ ist, kann nicht Element einer Knotenbasis sein. *13 Punkte*

(b) Beweisen Sie: Ein azyklischer Graph hat eine eindeutig bestimmte Knotenbasis. *12 Punkte*

(***Anmerkung***: Diese Knotenbasis ist sehr einfach zu finden.)