Einzelprüfung „Theoretische Informatik / Algorithmen (vertieft)“

Einzelprüfungsnummer 66115 / 2021 / Frühjahr

Thema 1 / Teilaufgabe 2 / Aufgabe 2

(*Minimum und Maximum*)

Stichwörter: Lineare Suche

(a) Argumentieren Sie, warum man das Maximum von n Zahlen nicht mit weniger als $n - 1$ Vergleichen bestimmen kann.

Lösungsvorschlag

Wenn die n Zahlen in einem unsortierten Zustand vorliegen, müssen wir alle Zahlen betrachten, um das Maximum zu finden. Wir brauchen dazu $n - 1$ und nicht n Vergleiche, da wir die erste Zahl zu Beginn des Algorithmus als Maximum definieren und anschließend die verbleibenden Zahlen $n - 1$ mit dem aktuellen Maximum vergleichen.

(b) Geben Sie einen Algorithmus im Pseudocode an, der das Maximum eines Feldes der Länge n mit genau $n - 1$ Vergleichen bestimmt.

Lösungsvorschlag

```java
public static int bestimmeMaximum(int[] a) {
  int max = a[0];
  for (int i = 1; i < a.length; i++) {
    if (a[i] > max) {
      max = a[i];
    }
  }
  return max;
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66115/jahr_2021/fruehjahr/MinimumMaximum.java`

(c) Wenn man das Minimum und das Maximum von n Zahlen bestimmen will, dann kann das natürlich mit $2n - 2$ Vergleichen erfolgen. Zeigen Sie, dass man bei jedem beliebigen Feld mit deutlich weniger Vergleichen auskommt, wenn man die beiden Werte statt in zwei separaten Durchläufen in einem Durchlauf geschickt bestimmt.

Lösungsvorschlag

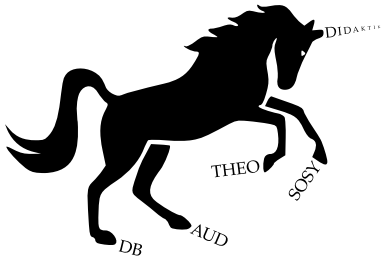
```java
/**
 * Diese Methode ist nicht optimiert. Es werden 2n - 2 Vergleiche benötigt.
 *
 * @param a Ein Feld mit Zahlen, in dem nach dem Minimum und dem Maximum gesucht
 *          werden soll.
 *
 * @return Ein Feld mit zwei Einträgen. Der erste Einträg enthält das Minimum,
 *         der zweite Eintrag das Maximum.
 */
public static int[] minMaxNaiv(int[] a) {
  int max = a[0];
  int min = a[0];
```

```
    for (int i = 1; i < a.length; i++) {
      if (a[i] > max) {
        max = a[i];
      }
      if (a[i] < min) {
        max = a[i];
      }
    }
    return new int[] { min, max };
  }

  /**
   * Diese Methode ist optimiert. Es werden immer zwei Zahlen paarweise
   * betrachtet. Die Anzahl der Vergleiche reduziert sich auf 3n/2 + 2 bzw.
   * 3(n-1)/2 + 4 bei einer ungeraden Anzahl an Zahlen im Feld.
   *
   * nach <a href=
   * "https://www.techiedelight.com/find-minimum-maximum-element-array-using-
↪  minimum-comparisons/">techiedelight.com</a>
   *
   * @param a Ein Feld mit Zahlen, in dem nach dem Minimum und dem Maximum gesucht
   *          werden soll.
   *
   * @return Ein Feld mit zwei Einträgen. Der erste Einträg enthält das Minimum,
   *         der zweite Eintrag das Maximum.
   */
  public static int[] minMaxIterativPaarweise(int[] a) {
    int max = Integer.MIN_VALUE, min = Integer.MAX_VALUE;
    int n = a.length;

    boolean istUngerade = (n & 1) == 1;
    if (istUngerade) {
      n--;
    }

    for (int i = 0; i < n; i = i + 2) {
      int maximum, minimum;

      if (a[i] > a[i + 1]) {
        minimum = a[i + 1];
        maximum = a[i];
      } else {
        minimum = a[i];
        maximum = a[i + 1];
      }

      if (maximum > max) {
        max = maximum;
      }

      if (minimum < min) {
        min = minimum;
      }
    }
```

```
    if (istUngerade) {
      if (a[n] > max) {
        max = a[n];
      }

      if (a[n] < min) {
        min = a[n];
      }
    }
    return new int[] { min, max };
  }

  /**
   * Diese Methode ist nach dem Teile-und-Herrsche-Prinzip optimiert. Er
   * funktioniert so ähnlich wie der Mergesort.
   *
   * nach <a href=
   * "https://www.enjoyalgorithms.com/blog/find-the-minimum-and-maximum-value-in-
↪   an-array">enjoyalgorithms.com</a>
   *
   * @param a Ein Feld mit Zahlen, in dem nach dem Minimum und dem Maximum
   *          gesucht werden soll.
   * @param l Die linke Grenze.
   * @param r Die rechts Grenze.
   *
   * @return Ein Feld mit zwei Einträgen. Der erste Einträg enthält das Minimum,
   *         der zweite Eintrag das Maximum.
   */
  int[] minMaxRekursiv(int[] a, int l, int r) {
    int max, min;
    if (l == r) {
      max = a[l];
      min = a[l];
    } else if (l + 1 == r) {
      if (a[l] < a[r]) {
        max = a[r];
        min = a[l];
      } else {
        max = a[l];
        min = a[r];
      }
    } else {
      int mid = l + (r - l) / 2;
      int[] lErgebnis = minMaxRekursiv(a, l, mid);
      int[] rErgebnis = minMaxRekursiv(a, mid + 1, r);
      if (lErgebnis[0] > rErgebnis[0]) {
        max = lErgebnis[0];
      } else {
        max = rErgebnis[0];
      }
      if (lErgebnis[1] < rErgebnis[1]) {
        min = lErgebnis[1];
      } else {
```

```
        min = rErgebnis[1];
      }
    }
    int[] ergebnis = { max, min };
    return ergebnis;
  }

}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66115/jahr_2021/fruehjahr/MinimumMaximum.java`



Die Bschlangaul-Sammlung
Hermine Bschlangaul and Friends

Eine freie Aufgabensammlung mit Lösungen von Studierenden für Studierende zur Vorbereitung auf die 1. Staatsexamensprüfungen des Lehramts Informatik in Bayern.

Hilf mit! Die Hermine schafft das nicht allein! Das ist ein Community-Projekt! Verbesserungsvorschläge, Fehlerkorrekturen, weitere Lösungen sind herzlich willkommen - egal wie - per Pull-Request oder per E-Mail an hermine.bschlangaul@gmx.net.Der TEX-Quelltext dieses Dokuments kann unter folgender URL aufgerufen werden: `https://github.com/bschlangaul-sammlung/examens-aufgaben/blob/main/Staatsexamen/66115/2021/03/Thema-1/Teilaufgabe-2/Aufgabe-2.tex`