[A13] MergeSort

Aufgabe

Im Package *A13_MergeSort* finden Sie die Klasse *Person*. Erweitern Sie die Klasse um die Funktion

```
public int compareTo(Person p)
```

Die Funktion soll anhand des Nachnamens, bei gleichen Nachnamen anhand des Vornamens, eine alphabetische Ordnung der beiden Objekte ermöglichen. Ähnlich der *compareTo()*-Funktion der String-Klasse soll eine negative Zahl retourniert werden, wenn die übergebene Person (*Nachname+Vorname*) später im Alphabet kommt, eine positive Zahl, wenn die übergebene Person früher im Alphabet kommt und Null, wenn Nachname und Vorname identisch sind. Aufbauend auf dieser Klasse implementieren Sie im Package die Klasse *MergeSort* die Methode

- `public void sort(Person[] personen)`

Im Package finden Sie auch passende JUnit-Tests. Diese leiten sich von der Klasse *PersonenSortTest* ab. Für die ausgeführten Tests müssen Sie also in dieser Klasse nachsehen.

Lösung

Die Implementierung eines *Merge-Sort-Algorithmus* besteht aus zwei Teilen. Wie bereits aus dem Namen ersichtlich ist, handelt es sich bei diesen zwei Teilen um die *Merge*- und die *Sort*-Methode.

In diesem Fall wird ein Feld sortiert, welches Objekte zur Darstellung einer Person enthält. Die Objekte vom Typ `Person` werden nach dem Vor- und Nachnamen sortiert.

Sort

Die Sort-Methode wurde mithilfe zweier gleichnamigen Methoden `sort` implementiert. Die Erste dient lediglich als ein erster Aufruf der zweiten rekursiven Methode, um den richtigen Anfangs- und Endpunkt zu bestimmen. Somit ist der Algorithmus auf einem beliebigem Feld einfach aufrufbar .

Die rekursive Methode ist zuständig für die Halbierung des Feldes, solange unser ursprüngliches Feld in kleine Abschnitte zertilt wird. Die maximale Länge dieser Abschnitte ist ein Element.

Abschließend wird die Methode `merge` aufgerufen, um die sortierten Abschnitte in eine sortierte Einheit zusammenzufügen.

```
public void sort(Person[] personen) {
  sort(personen, 0, personen.length - 1);
}
```

```java
public void sort(Person[] personen, int start, int end) {
  if (end <= start) return;
  int mitte = (start + end) / 2;
  sort(personen, start, mitte);
  sort(personen, mitte + 1, end);
  Person[] teil1 = Arrays.copyOfRange(personen, start, mitte + 1);
  Person[] teil2 = Arrays.copyOfRange(personen, mitte + 1, end + 1);
  merge(teil1, teil2, personen, start);
}
```

Laufzeit

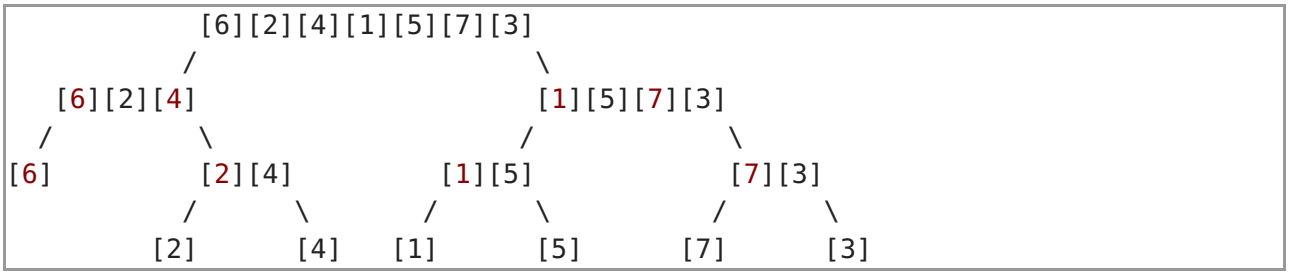
Die Laufzeit der Methode `sort` beträgt *O(n log n)*. Wobei *n* die Länge des ursprünglichen Feldes darstellt. Dabei stammt *O(n)* aus der Laufzeit der Methode `merge` und *O(log n)* aus der rekursiven Halbierung.

Beispiel

Ein Beispiel mit einem Zahlenfeld der Länge sieben.

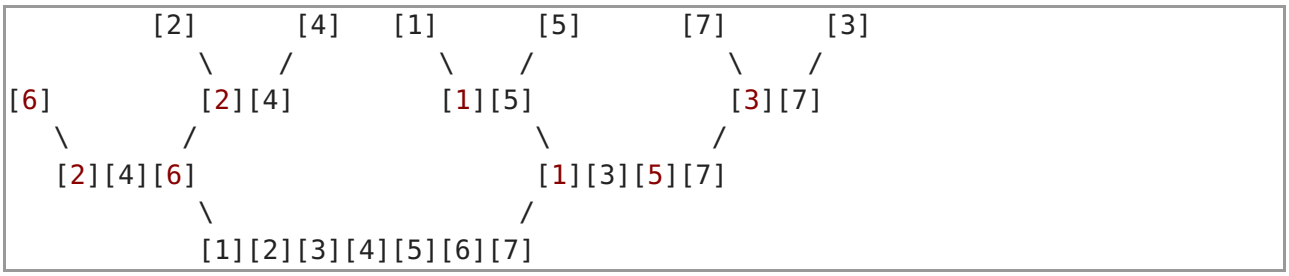
Aufteilung

Es ist sichtbar, dass die Höhe des Baumes, der die Halbierungen darstellt, proportional zum Logarithmus der Anzahl der Elemente ist.



Zusammenfügen

Bei den Aufrufen der Methode `merge` wird ein Spiegelbild der Aufteilung ersichtlich. Dies ist ebenfalls proportional zum Logarithmus.



Merge

Die Merge-Methode ist in `merge` implementiert. Diese Methode fügt die Elemente zweier Felder `pers1` und `pers2` zusammen.

Zu diesem Zweck wird kein neues Feld erstellt, sondern es wird das ursprüngliche Feld `resultab` dem Startpunkt `start` mit den zusammengefügten bzw. sortierten Personenobjekten aufgefüllt. Der richtige Startpunkt wird in der Methode `sort` berechnet.

Am Ende ist ein sortierter Abschnitt vorhanden, der im `start` anfängt. Seine Länge gleicht der Summe der Längen der beiden Felder `pers1` und `pers2`.

```
public void merge(Person[] pers1, Person[] pers2, Person[] result, int start) {
  int i = 0;
  int j = 0;
  while (i < pers1.length && j < pers2.length) {
    if (pers1[i].compareTo(pers2[j]) < 0) {
      result[start] = pers1[i];
      i++;
    } else {
      result[start] = pers2[j];
      j++;
    }
    start++;
  }
  while (i < pers1.length) {
    result[start] = pers1[i];
    i++;
    start++;
  }
  while (j < pers2.length) {
    result[start] = pers2[j];
    j++;
    start++;
  }
}
}
```

Laufzeit

Die Laufzeit der Methode `merge` beträgt *O(n)*. Wobei *n* die Länge des sortierten Abschnittes darstellt.

Beispiel

Wie das Zusammenfügen funktioniert wird im letzten Schritt des vorigen Beispiels gezeigt. Es wird ersichtlich, dass das Zusammenfügen eine lineare Laufzeit hat, welche proportional der Länge des sortierten Abschnittes ist. Das heißt: sieben Schritte für sieben Elemente.

Anfang

```
pers1 := [2][4][6]    pers2 := [1][3][5][7]
          ^                      ^

result := [2][4][6][1][3][5][7]
           ^
```

Schritt 1

```
pers1 := [2][4][6]    pers2 := [1][3][5][7]
          ^                      ^
result := [2][4][6][1][3][5][7]
           ^

2 < 1 ? Nein!

pers1 := [2][4][6]    pers2 := [1][3][5][7]
          ^                          ^
result := [1][4][6][1][3][5][7]
              ^
```

Schritt 2

```
pers1 := [2][4][6]    pers2 := [1][3][5][7]
          ^                          ^
result := [1][4][6][1][3][5][7]
              ^

2 < 3 ? Ja!

pers1 := [2][4][6]    pers2 := [1][3][5][7]
             ^                       ^
result := [1][2][6][1][3][5][7]
                 ^
```

Schritt 3

```
pers1 := [2][4][6]    pers2 := [1][3][5][7]
             ^                       ^
result := [1][2][6][1][3][5][7]
                 ^

4 < 3 ? Nein!

pers1 := [2][4][6]    pers2 := [1][3][5][7]
             ^                          ^
result := [1][2][3][1][3][5][7]
                    ^
```

Schritt 4

```
pers1 := [2][4][6]    pers2 := [1][3][5][7]
             ^                        ^
result := [1][2][3][1][3][5][7]
                    ^

4 < 5 ? Ja!

pers1 := [2][4][6]    pers2 := [1][3][5][7]
                ^                        ^
result := [1][2][3][4][3][5][7]
                      ^
```

Schritt 5

```
pers1 := [2][4][6]    pers2 := [1][3][5][7]
                ^                        ^
result := [1][2][3][4][3][5][7]
                      ^

6 < 5 ? Nein!

pers1 := [2][4][6]    pers2 := [1][3][5][7]
                ^                          ^
result := [1][2][3][4][5][5][7]
                         ^
```

Schritt 6

```
pers1 := [2][4][6]    pers2 := [1][3][5][7]
                ^                          ^
result := [1][2][3][4][5][5][7]
                         ^

6 < 7 ? Ja!

pers1 := [2][4][6]    pers2 := [1][3][5][7]
                  ^                        ^
result := [1][2][3][4][5][6][7]
                           ^
```

Schritt 7

```
pers2 := [1][3][5][7]
                  ^
result := [1][2][3][4][5][6][7]
                           ^

pers2 := [1][3][5][7]
                    ^
result := [1][2][3][4][5][6][7]
                             ^
```