Team: 03, Sebastian Diedrich – Murat Korkmaz

Aufgabenaufteilung:

- Aufgaben, für die Teammitglied 1 verantwortlich ist:
  (1) Skizze
  Dateien, die komplett/zum Teil von Teammitglied 1 implementiert/bearbeitet wurden:
  (1) folgt

- Aufgaben, für die Teammitglied 2 verantwortlich ist:
  (1) Skizze
  Dateien, die komplett/zum Teil von Teammitglied 1 implementiert/bearbeitet wurden:
  (1) folgt

Quellenangaben: Vorlesung am 04.11.15

Bearbeitungszeitraum: 10.11. (4h)

Aktueller Stand: Skizze fertig (Rückmeldung durch Prof. offen)

Skizze: (ab Seite 2)

Skizze Aufgabe 2:

Aufgabe: 2.1

Ziel: Zahlengenerator

Angaben zur Implementation:

- Das Trennungssymbol zwischen den einzelnen Zahlen soll ein Leerzeichen sein. Es sollen nur positive Zahlen erzeugt werden. Zahlen dürfen mehrfach vorkommen.
- Auch der beste und schlimmste Fall (Zahlen sind sortiert vs. Zahlen sind umgekehrt sortiert) soll mit einer beliebigen Anzahl von Zahlen generierbar sein. Dafür soll eine Methode mit gleichem Namen, aber mit zwei Parametern implementiert werden.

Vorgaben für die Implementation:

- Semantische Vorgabe:
    a. anzahlZahlen -> Datei
       Datei enthält die gewünschte Anzahl von zufällig erzeugten positiven Zahlen.
- Syntaxtische Vorgabe:
    a. Name der Klasse: SortNum
    b. Name der Methode: sortNum
    c. Aufruf der Methode:
       Sortnum.sortNum(int anzahlZahlen) -> zufällige Zahlenfolge
       Sortnum.sortNum(int anzahlZahlen, boolean bool):
           - true: komplett sortiert
           - false: komplett umgekehrt sortiert
    d. Endung der Datei: .dat

Aufgabe: 2.2

Ziel: Insertion Sort

Algorithmus

Gegeben:

- Eine unsortierte Zahlreihe mit *n* Zahlen. (n > 1)

Ablauf:

- Jede Zahl wird mit den Zahlen links von ihr verglichen. Begonnen wird mit der zweiten Zahl. Sollte im direkten Vergleich die Zahl links von der „aktuell zu betrachtenden Zahl“ größer sein, tauschen diese beiden Zahlen ihre Position. Dieses wird solange fortgesetzt, bis die Zahl links kleiner oder gleich groß ist bzw. sich die „aktuell zu betrachtenden Zahl“ an der ersten Position der Zahlreihe befindet. Hat die „aktuell zu betrachtenden Zahl“ ihre endgültige Position erreicht, wird mit der Zahl weitergemacht, die sich rechts von der ursprünglichen Startposition der „aktuell zu betrachtenden Zahl“ befand. Sollte die ursprüngliche Startposition der letzten Position der Zahlenreihe entsprechen, ist der Algorithmus beendet und die Zahlen sind sortiert!

Test

Um die Richtigkeit und die Zuverlässigkeit des Algorithmus zu gewährleisten, sollen umfangreiche JUnit-Tests implementiert werden.

Dateiname des Jar-Files:

- insertionJUt.jar

Mindestanforderung:

1. Grenzfälle
    a. Zahlenreihe mit n = 2 Zahlen (sortiert, unsortiert und beide Zahlen gleich)
    b. Zahlenreihe mit n = 1.000 Zahlen (umgekehrt sortiert)
        i. Beispiel: (1.000, 999, … , 5, 4, 3, 2, 1)
    c. Zahlenreihe mit n = 1.000 Zahlen (sortiert)
        i. Beispiel: (1, 2, …, 999, 1.000)
2. Belastung
    a. Zahlenreihe mit n > 10.000
    b. Zahlenreihe mit n > 100.000
    c. Zahlenreihe mit n > 1.000.000

Vorgaben für die Implementation:

- Semantische Vorgabe:

    a. Insertionsort: array x startPos x endPos -> array
    startPos: dabei handelt es sich um eine positive Zahl, die nicht größer sein darf als die Endposition (endPos)

- Syntaxtische Vorgabe:

    a. Name der Klasse: Insertionsort

    b. Name der Methode: insertionsort

    c. Aufruf der Methode:
    Insertionsort.insertionsort(ADTArray array, int startPos, int endPos)

Aufgabe: 2.3

Ziel: Quicksort

Gegeben:

- Eine unsortierte Zahlreihe mit *n* Zahlen. (n >= 1)

Ablauf:

- Definition Pivot-Element:
    a. Dient als Referenzelement, dass mit allen anderen Zahlen der Zahlenreihe verglichen wird und ggf. die Position tauscht. Dieses passiert so lange, bis alle Zahlen links dieses Pivot-Elementes kleiner (gleich) und rechts von diesem Element größer sind.
- Zunächst wird das Pivot-Element bestimmt. Wie dieses geschieht, ist nicht fest vorgegeben. Für die Erläuterung des Algorithmus wird in diesem Fall immer die Zahl als Pivot-Element bestimmt, welche sich ganz links in der (Teil)Zahlenreihe befindet.
  Das aktuelle Pivot-Element wird mit allen anderen Zahlen der (Teil)Zahlenreihe verglichen und wie oben beschrieben ggf. um dessen Position getauscht. Hat es seine endgültige Position gefunden, wird die (Teil)Zahlenreihe in zwei Teile aufgeteilt, wobei das Pivot-Element die „Grenze“ der beiden Teil-Zahlenreihen darstellt. Da das Pivot-Element sich bereits an seiner endgültigen Position befindet, wird es weder der linken noch der rechten Teil-Zahlenreihe zugeordnet.
  Der Algorithmus wird nun auf jede Teil-Zahlenreihe angewandt, bis jede Teil-Zahlenreihe nur noch aus einer Zahl besteht. Nun ist die Zahlenreihe sortiert und der Algorithmus endet!

Vorgaben für die Implementation:

- Semantische Vorgabe:
    a. Quicksort: array x method_pivot -> array
    method_pivot: bestimmt die Pivot-Element-Auswahl
    Folgende Auswahlmöglichkeiten sollen implementiert werden:
        i. Immer ganz links
        ii. Immer ganz rechts
        iii. Medianof3
            1. Median von folgenden Zahlen:
                a. Zahl an erster Position
                b. Zahl an letzter Position
                c. Zahl an mittlerer Position
        iv. Random

- Syntaxtische Vorgabe:
    a. Name der Klasse: Quicksort
    b. Name der Methode: quicksort
    c. Aufruf der Methode: Quicksort.quicksort(ADTArray array)
- Sollte die (Teil)Zahlenreihe weniger als 12 Elemente beinhalten, werden die (Teil)Zahlenreihen mittels Insertionsort sortiert.

Test

Um die Richtigkeit und die Zuverlässigkeit des Algorithmus zu gewährleisten, sollen umfangreiche JUnit-Tests implementiert werden.

Dateiname des Jar-Files:

- quicksortJUt.jar

Mindestanforderung:

3. Grenzfälle
    a. Zahlenreihe mit n = 2 Zahlen (sortiert, unsortiert und beide Zahlen gleich)
    b. Zahlenreihe mit n = 1.000 Zahlen (umgekehrt sortiert)
        i. Beispiel: (1.000, 999, … , 5, 4, 3, 2, 1)
    c. Zahlenreihe mit n = 1.000 Zahlen (sortiert)
        i. Beispiel: (1, 2, …, 999, 1.000)
4. Belastung
    a. Zahlenreihe mit n > 10.000
    b. Zahlenreihe mit n > 100.000
    c. Zahlenreihe mit n > 1.000.000

Aufgabe 2.4

Messung

Es wird eine Kopie der Algorithmus-Implementation erstellt und um folgende Komponenten erweitert:

- Laufzeit:
    a. Zu Beginn des Algorithmus wird die aktuelle Zeit festgehalten. Am Ende des Algorithmus wird die aktuelle Zeit festgehalten und der Betrag der Differenz der beiden ausgegeben. Dieser Betrag entspricht der Laufzeit. (zu beachten ist dabei, dass diese Laufzeit NUR mit Laufzeiten verglichen werden darf, die ebenfalls auf dem gleichen Ausführungssystem gemessen worden sind)
    b. Bei der Laufzeitmessung von Quicksort muss beachtet werden, dass die Laufzeit von Insertionsort abgezogen wird. (Sortierung von Zahlenreihen mit weniger als 12 Zahlen)
- Zugriffe (Lesen und Schreiben)
    a. Lesen
        i. Ein Lesezugriff ist definiert, durch das Auslesen eines Wertes aus der Zahlenreihe.
    b. Schreiben
        i. Ein Schreibzugriff ist definiert, durch das mutieren der Zahlenreihe oder durch die Belegung einer Variablen mit einem Wert.

Dokumentation

Alle Messungen eines Algorithmus sollen in eine CSV-Datei geschrieben werden. Später sollen die Messwerte interpretiert werden und der dazugehörige Versuchsaufbau und die Resultate in einem PDF dokumentiert werden.