ADP Aufgabe 1

Vorarbeiten

1 Installieren Sie von der Seite https://algs4.cs.princeton.edu/code/ die Bibliothek algs4.jar und laden auch die Daten algs4-data.zip herunter.
2 Fügen Sie den algs4.jar Datei dem CLASSPATH in der Systemumgebung hinzu.
3 Machen Sie sich mit den Klassen der Princeton-Bibliothek vertraut. Sie dürfen diese Klassen kopieren und ggf. anpassen.

Teil A

Lernziel

1. Anwenden der rekursiven Inplace-Suche in einem Array: Die rekursive Inplace-Suche arbeitet nur mit dem Original-Array. Es dürfen keine Kopien erzeugt werden. Der zu untersuchende Bereich wird beim Aufruf als minimaler (`lo`) und maximaler Index (`hi`) übergeben. Das Verfahren der Inplace-Verarbeitung von Arrays wurde mit der binären Suche eingeführt und wird uns in vielen Sortieralgorithmen wieder begegnen.

Aufgabenstellung

Implementieren Sie eine statische Methode, die in einem `int[]` Array `a` der Länge `N` ein lokales Maximum in einem vorgegebenen `radius` bestimmt. Gehen Sie wie folgt vor:

1. Betrachten Sie zuerst das mittlere Element an der Position `N/2`.
    a. Wenn für alle Elemente, die im Intervall `[N/2-radius,N/2)` liegen, gilt, dass `a[j]<a[j+1]` ist und für alle Elemente, die im Intervall `[N/2,N/2+radius)` liegen gilt, dass `a[j]>a[j+1]` ist, dann ist das Element an Position `N/2` das lokale Maximum und die Methode gibt den Bereich des lokalen Maximus zurück.
2. Ansonsten arbeiten Sie mit der Hälfte weiter, die den kleineren Nachbarn enthält. Dabei bleibt die Mitte Bestandteil der jeweiligen Hälfte.

Beispiel:

Für `ary1 = [1, 61, 89, 75, 16, 33, 89, 59, 28, 3, 3, 97, 61, 85, 47, 38, 78, 7, 6, 15]`

Berechnet der Aufruf von `localMax(ary,2): [1, 61, 89, 75, 16]`

Für `ary2 = [1, 61, 16, 75, 89, 133, 89, 59, 28, 3, 3, 97, 61, 85, 47, 38, 78, 7, 6, 15]`

Berechnet der Aufruf von `localMax(ary2,3): [16, 75, 89, 133, 89, 59, 28]`

Für `ary3 = [99,1, 61, 89, 75, 16, 33, 89, 59, 28, 3, 3, 97, 61, 85, 47, 38, 78, 7, 6, 15]`

Berechnet der Aufruf von `localMax(ary3,1): null`

Teil B

Lernziele

1. Inplace Verarbeitung von Arrays mit sich verändernden Bereichen.
2. Anwendung des Java Pipelining (Programme von der Konsole starten).
3. Umgang mit den und ggf. Anpassung der Princeton-Bibliotheken.

Aufgabenstellung

1. Implementieren Sie eine Klasse `NumberGenerator`, die `N` Zufallszahlen (`int` und `double` Werte) in einem vorgegebenen Intervall [`min`,`max`] generiert. `N`, `min` und `max` sollen dem Programm beim Start übergeben werden. `int` und `double` Werte sollten ungefähr gleich verteilt sein. `NumberGenerator` gibt die Zahlen durch Leerzeichen separiert auf der Konsole aus.
2. Implementieren Sie eine Klasse `NPlusEvenFilter`, die aus einem Eingabestrom nur die positiven ganzen geraden Zahlen filtert und diese auf der Konsole ausgibt.
3. Implementieren Sie eine Clientklasse `AccumulatorClient`, die die Daten für die in der Vorlesung gezeigten Klassen `SimpleAccumulator` und `VisualAccumulator` aus einem Eingabestrom einliest und an den Akkumulator übergibt. Wird das Programm `AccumulatorClient` mit `0` aufgerufen, dann wird ein `SimpleAccumulator` erzeugt, mit einer anderen Zahl im ersten Programmargument wird ein `VisualAccumulator` erzeugt. Dann muss im 2'ten und 3'ten Programmargument noch die Anzahl der Daten und der maximale Wert angegeben werden. Alternativ berechnen Sie die Werte der x-Achse und den maximalen Wert, indem Sie die Werte von `StdIn` lesen und in einer Liste zwischenspeichern.
4. Erzeugen Sie eine Pipeline, in der der `NumberGenerator` Zufallszahlen generiert, der `NPlusEvenFilter` die geraden ganzen Zahlen aus dem Eingabestrom filtert und der `AccumulatorClient` die Daten aus dem Eingabestrom aggregiert und darstellt.