Hendrik Wedemeier, Tristan Wieczorek

Entwurf Praktikum 2

1)

Im folgenden Bericht werden Hendrik Wedemaier und Tristan Hanno Friedensreich Karl Wieczorek den Aufbau und die Funktionsweise verscheidener Sortieralgorithmen Analysieren und beschreiben. Bei diesen Sortieralgorithmen handelt es sich im Speziellen um die Algorithmen Insertion Sort, Radix Sort, Heap Sort und Intro Sort. Außerdem wurden Ablaufdiagramme(Flowcharts) zu jedem Sortieralgorithmus erstellt.

2)

Funktionale Anforderungen

Alle zu Programmierenden Algorithmen müssen in Erlang geschrieben werden.

Arrays sollen durch Listen ersetzt werden.

Speicher darf nicht simuliert werden.

Insertion Sort

Schnittstelle: insertionS:insertionS(<Liste>)

Return: sortierte Liste

Radixsort

Schnittstelle: radixS:radixS(<Liste>, <Digit>)

<Digit> gibt dabei die maximal vorkommende Stelligkeit der Zahlen an, auch hier wird eine korrekt sortierte Rückgabe der Eingabe erwartet.

Introsort

Schnittstelle: introS:introS(<pivot-methode>, <Liste>, <switch-num>)

<pivot-method> Werte sind: “left,right,middle,median,random”. Ein Wechsel zum Heapsort erfolgt automatisch, wenn die maximale Rekursionstiefe log2(n) * 2 überschritten wird. Der Wechsel zu Insertion Sort hängt von dem Parameter <switch-num> ab.

Heapsort

Schnittstelle: heapS(<Liste>)

Verwendet wird ein binärer Baum mit folgender Struktur nach Empfehlung Klaucks:

Leerer Baum: {}

Beispielbaum: {42, {}, {}}

Ein vorberechneter Pfad zur nächsten freien Position (aus heapS.erl) ermöglicht das Einfügen.

Technische Anforderungen

- Programmiersprache muss Erlang sein
- Keine Bibliotheksfunktionen/Modulen
- Jeder Algorithmus wird in einem eigenen Modul implementiert
- Schnittstellenanforderung siehe Funktionale Anforderungen
- Fehlertoleranz: Verarbeitung von Edge-Cases (leere Listen), fehlerhafte Eingaben oder ungültige Parameter
- Maximale Laufzeit ist 30 Sekunden
- Laufzeitanalysen im Millisekunden Bereich, zeitInSswitch:messung() darf verwendet werden

3)

Insertion Sort

-Eine Liste, da das sortieren von Elementen in eine clon liste die programmierung erheblich vereinfacht.

Radix Sort(Bucket Sort)

-Eine Liste für jeden Bucket sowie eine concat Liste.

Der Bucket sort sortiert Zahlen je nach Stellenwert in den korrespondieren Bucket und konkatiniert die bucks nach sortierung wieder zu einer Liste.Die durchläuft iterationen n wobei n für die maximale Anzahl an Ziffern der Nummern steht.

Heap Sort

Binärer Baum mit Tupeln in Form von {Wert, linkerBaum, RechterBaum} und {} für einen leeren Baum .

Intro Sort

Eine Liste als übergabe Wert, da der Introsort lediglich entscheidet ob der Insertion, Heap oder Quicksort benutzt werden soll

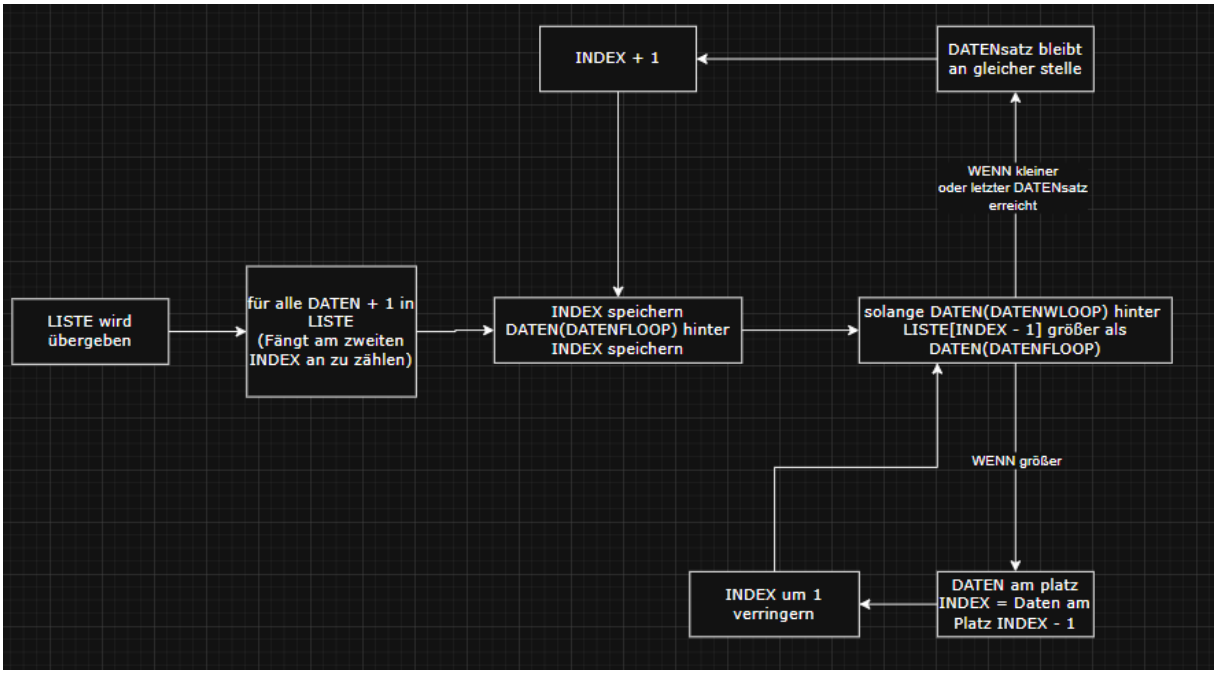
Quick Sort

Eine Liste an Elementen

N Anzahl an Pivot-Elementen und N Anzahl an Listen, wobei N für die Menge an Durchläufen steht die man braucht, um die ursprüngliche Liste zu sortieren

4)

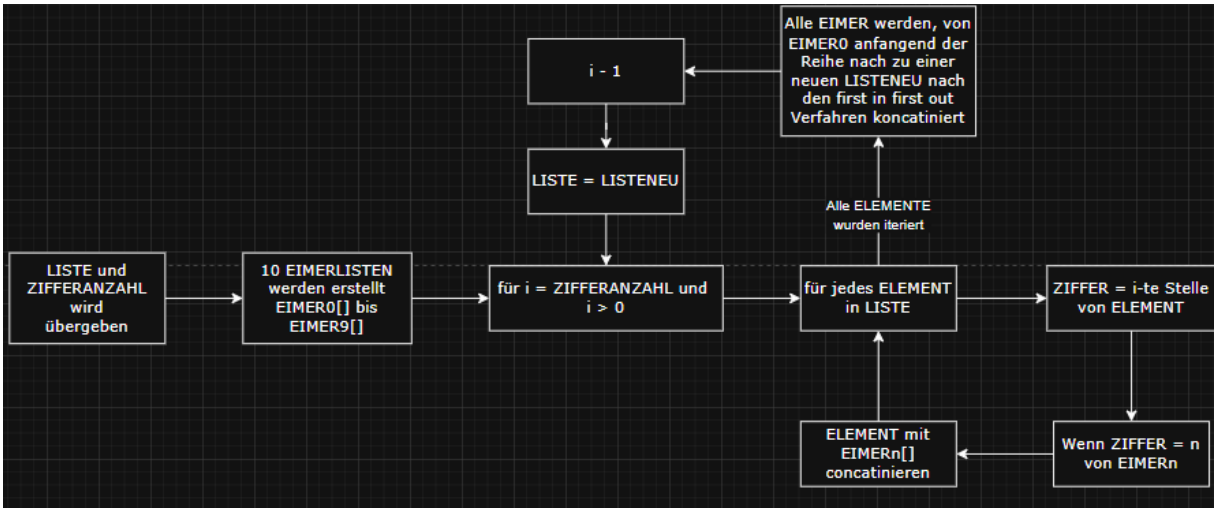
Insertion Sort



Komplexität: Simpel

Basis für viele sorts weshalb funktionalität und Leistung gewährleistet sein müssen

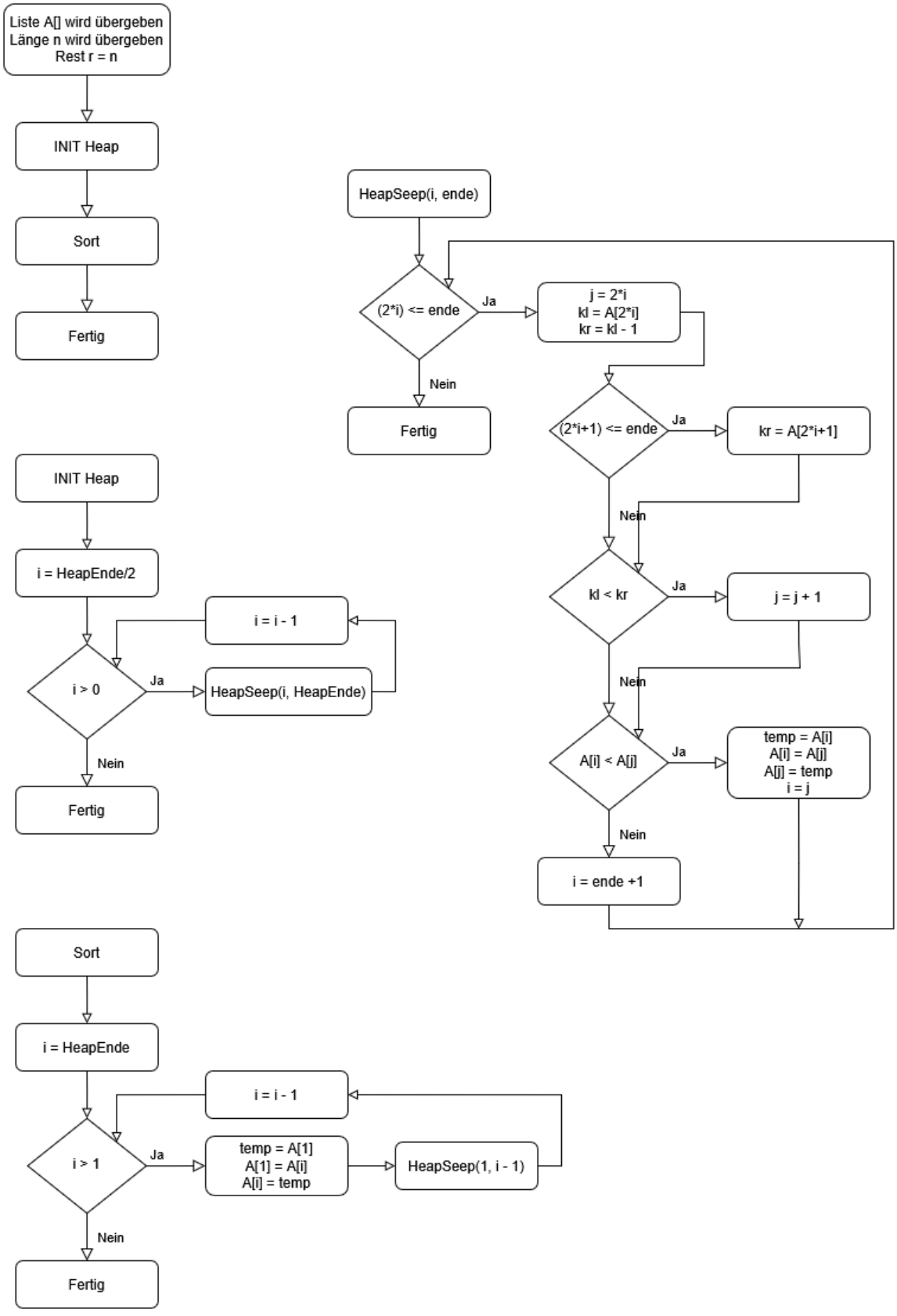
Bucket Sort



Komplexität: Relativ simpel

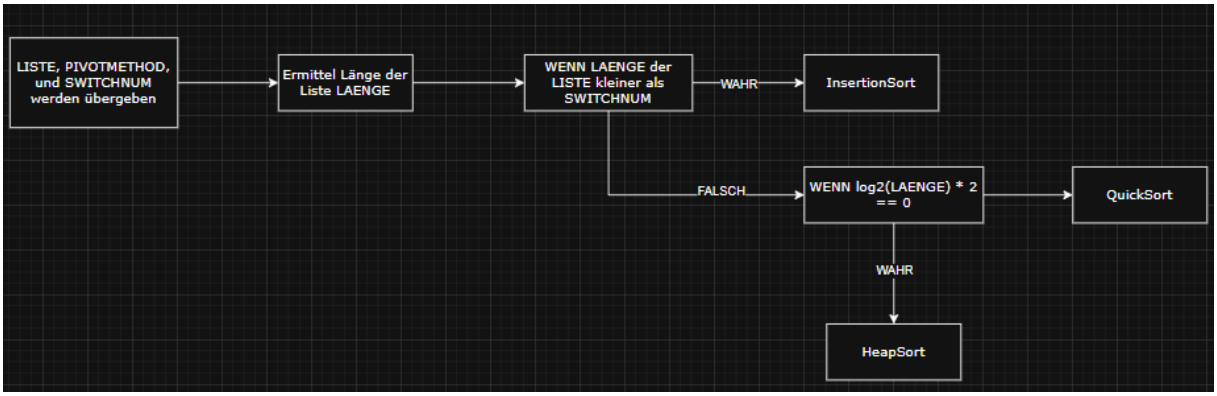
Basis für viele sorts weshalb funktionalität und Leistung gewährleistet sein müssen

Heap-Sort



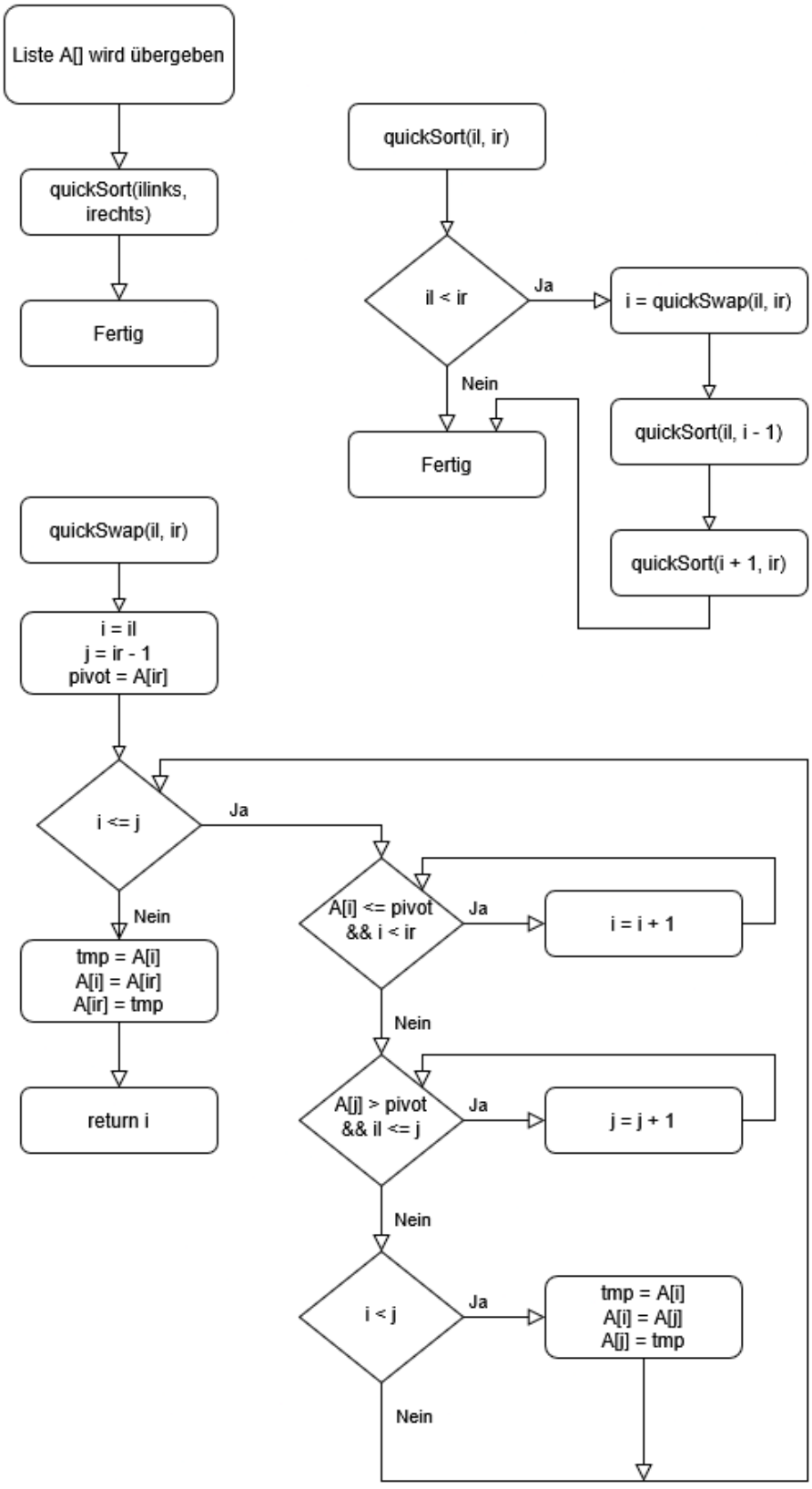
Komplexität: Erheblich

Introsort



Komplexität: Simpel aber leistung baut auf den unterliegenden Sortieralgorythmen auf

Quick Sort


Liste A[] wird übergeben
quickSort(ilinks, irechts)
Fertig
quickSort(il, ir)
il < ir
Ja
i = quickSwap(il, ir)
Nein
Fertig
quickSort(il, i - 1)
quickSort(i + 1, ir)
quickSwap(il, ir)
i = il
j = ir - 1
pivot = A[ir]
i <= j
Ja
Nein
tmp = A[i]
A[i] = A[ir]
A[ir] = tmp
return i
A[i] <= pivot
&& i < ir
Ja
i = i + 1
Nein
A[j] > pivot
&& il <= j
Ja
j = j + 1
Nein
i < j
Ja
tmp = A[i]
A[i] = A[j]
A[j] = tmp
Nein

Komplexität: Mittelschwer

5)

Es ist uns möglich den Code auf unterschiedlicher hardware, unterschiedlichen Betriebssystemen zu testen.

Code wird iterativ konstruiert, tests gewährleisten funktionalität, sowie Introspektieve zu vergessenen Edgecases.

Die Abgabe wird zwischen zwei personen aufgeteilt. Person A baut Insertion Sort und Bucket Sort.

Person B baut Heap Sort und Quick Sort mit Intro Sort für die Person die als erstes fertig ist.

Die Algorithmen werden gemessen und dürfen eine gesetzte Zeit nicht überschreiten.

Die testdaten sind zufällig große Listen, welches die Zuferlässigkeit der Algorithmen gewährleistet.

Testfälle

- Eine Liste gleicher, sich wiederholender Zahlen
- Eine leere Liste
- Eine Liste aus Strings
- Eine Liste aus Ganzzahlen und Kommazahlen
- Eine Liste aus Kommazahlen
- Eine Liste aus String uns Zahlen
- Eine Liste mit einem String am Anfang
- Eine Liste mit einem String am Ende
- Eine Liste aus Listen