Algorithmen und Datenstrukturen

Master

Sortieren

Motivation

- Langjährige Untersuchungen haben gezeigt, dass rund ¼ der kommerziell gebrauchten Rechnerzeit auf Sortiervorgänge entfällt.
- Sortieren erfolgt in zwei Schritten
  - Beschaffe Information (Schlüssel)
  - Transportiere Daten (Verschieben, …)

Allgemeines

- Datensätze enthalten Schlüssel, nach welchen die Sätze zu sortieren sind.
- Liegen alle Daten im Speicher, nennt man das *internes* Sortierverfahren.
  - direkter schneller Zugriff auf alle Daten
- Sortierung über Dateien nennt man *externe* Sortierverfahren.
  - Datenzugriff sequentiell oder in großen Blöcken

Allgemeines

- *Stabile Sortierverfahren* behalten die relative Reihenfolge gleicher Schlüssel bei.
- Zum Sortieren werden einzelne Records oft vertauscht (swap)
- folgend wird ein Array a[N] sortiert

Selection Sort

- finde das kleinste Element a[i] aus i=0..N und tausche es gegen das erste a[0]
- finde das kleinste Element aus a[i], i=1..N und tausche es gegen a[1] usw.

- 2 geschachtelte Schleifen, d.h. $O(n^2)$
- Laufzeit abhängig von Vorsortierung

Selection Sort

A	S	D	F	G	H
A	S	D	F	G	H
A	D	S	F	G	H
A	D	F	S	G	H
A	D	F	G	S	H
A	D	F	G	H	S

Insertion Sort

- Suche a[i], hinter welchem ein neues Element eingefügt werden muß
- Verschiebe a[i+1]..a[N-1] nach hinten
- kopiere neues Element nach a[i+1]

- Achtung, wenn neu < a[0] !

- Laufzeit abhängig von Vorsortierung
- 2 geschachtelte Schleifen, d.h. $O(N^2)$

Insertion Sort

A	S	D	F	G	H

A					
A	S				
A	D	S			
A	D	F	S		
A	D	F	G	S	
A	D	F	G	H	S

Bubble Sort

- Durchlaufe wiederholt die Daten und vertausche jedesmal die Nachbarn, falls notwendig.
- Wenn ein kompletter Durchlauf ohne Tausch gelingt, sind die Daten sortiert.

- Ähnlich wie Selection Sort, nur daß viel mehr Aufwand durch ständiges Tauschen entsteht.

Bubble Sort

A	S	D	F	G	H

A	S	D	F	G	H
A	S	D	F	G	H
A	D	S	F	G	H
A	D	F	S	G	H
A	D	F	G	S	H
A	D	F	G	H	S

Shellsort

- Idee: wie Selection Sort, nur daß man eine sortierte Folge bekommt, wenn jedes *h*-te Element gelesen wird (*h-sortiert*).
- Eine h-sortierte Datei besteht also aus h unabhängigen sortieren Blöcken, die überlagert sind.
- Für eine Folge von h=groß bis h=1 erhält man *eine* sortierte Datei.

Shellsort

- Beispielfolge: 1, 4, 13, 40, 121, 364,...

$$h_n = 3h_{n-1} + 1, \quad n = 1...N/9$$

- Ist eine gute Folge, weil für viele Fälle nachweislich beste Performance
- Wichtig: keine 2^n-Folgen, da sonst die Hälfte der Elemente erst beim letzten Durchlauf verglichen werden!

Shellsort

- Algorithmus und Aufwand nicht exakt geklärt, hängt auch von *h*–Folge ab
- Laufzeit nicht abhängig von Vorsortierung der Daten
- Aufwand ca. $N^{1,25}$
- Obergrenze $O(N^{3/2})$

Shellsort

A	S	O	R	T	I	N	G	E	X	A	M	P	L	E
A													L	
	E													S
A	E	O	R	T	I	N	G	E	X	A	M	P	L	S
A				E				P				T		
	E				I				L				X	
		A				N				O				S
			G				M				R			
A	E	A	G	E	I	N	M	P	L	O	R	T	X	S
A	A	E	E	G	I	L	M	N	O	P	R	S	T	X

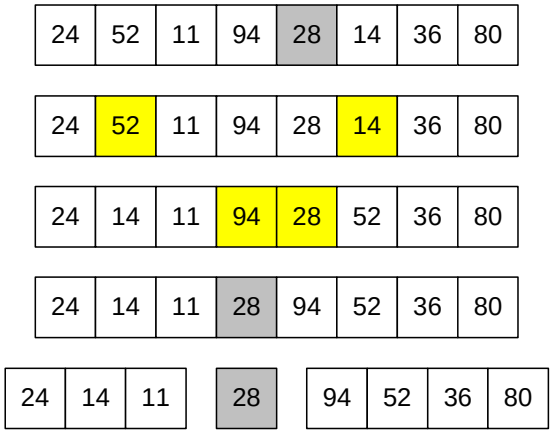
Quicksort

- Prinzip: teile die Daten in zwei Teile und sortiere diese unabhängig voneinander.
- Die Teilung erfolgt so, daß alle Elemente links von a[i] kleiner als a[i], alle rechts größer als a[i] sind.
- Teilung muß ausgewogen erfolgen
- Z.B. median-of-three:
  - Wähle 3 zufällige Elemente
  - Nimm den mittleren Wert als Teilungswert

Quicksort

- Dann werden von links und rechts jeweils nichtpassende Elemente gesucht und getauscht:
  - Von links: Werte größer als Teiler
  - Von rechts: Werte kleiner als Teiler
- Überkreuzen diese beiden Indizes, so ist das Array richtig geteilt
- Für jeden Teil wird wieder rekursiv der gleiche Algorithmus angewendet.

Quicksort



Quicksort

- Häufig eingesetzt, einfache Implementierung
- 1960 von C.A.R. Hoare entwickelt

- Mittlerer Aufwand O(N log N)
- Worst Case Aufwand $O(N^2)$

- Praktisch unabhängig von Vorsortierung
- Worst Case Verhalten kann durch geschickte Auswahl des Teilungsverfahrens praktisch ausgeschlossen werden

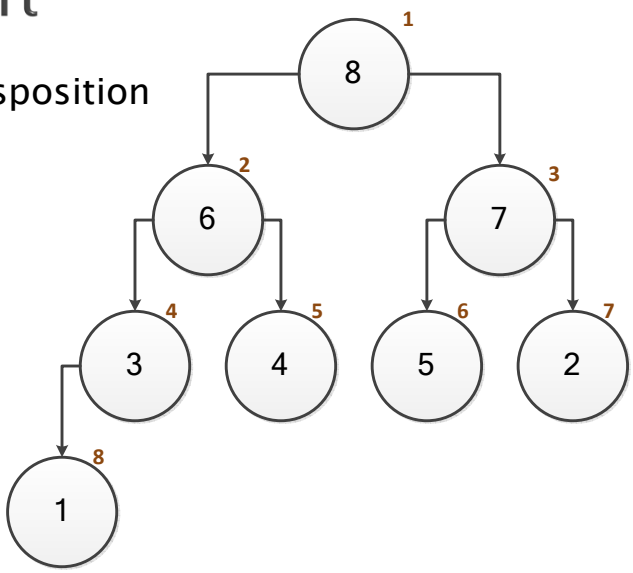
Heapsort

- Ziel ist das Sortieren einer Datenmenge so, dass immer das grösste Element zugreifbar ist.
- Datenstruktur Heap
  - N Schlüssel werden so organisiert, dass gilt:

$$k_i \leq k_{i/2} \text{ für } 2 \leq i \leq N$$

Heapsort

- Ausgangsposition

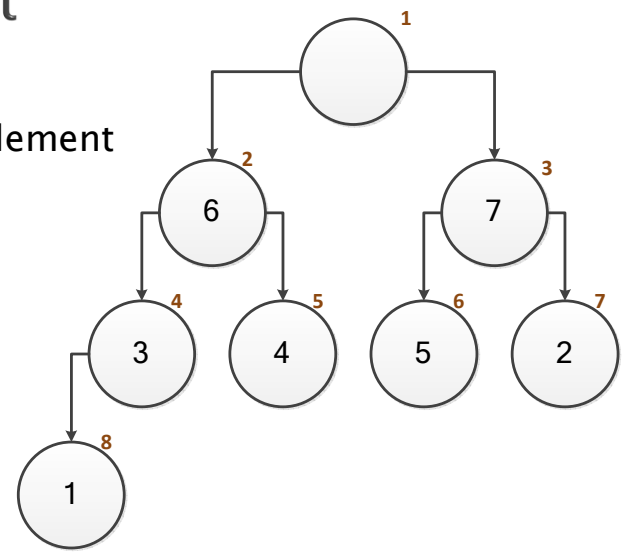


Heapsort

- Heap sortiert Elemente durch Versickern:
  - Neues Element wird an Spitze eingetragen
  - Vergleiche beide Kind-Elemente
    - Tausche mit grösserem Platz
    - Solange bis beide Kind-Elemente kleiner sind oder das ursprüngliche Element am Boden angelangt ist.
- Grösstes Element ist immer am Root-Knoten

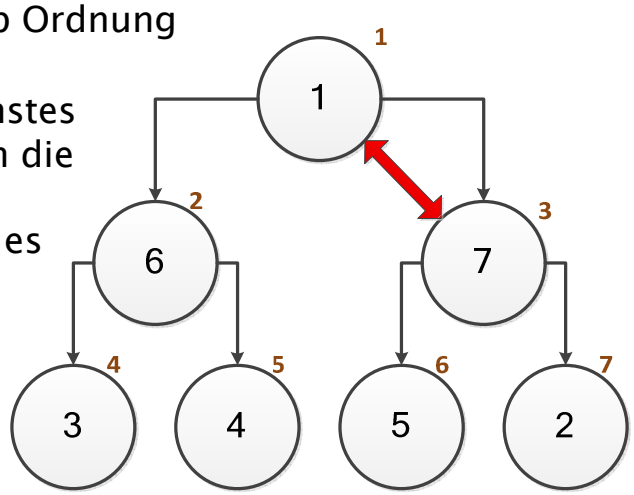
Heapsort

- Entferne grösstes Element



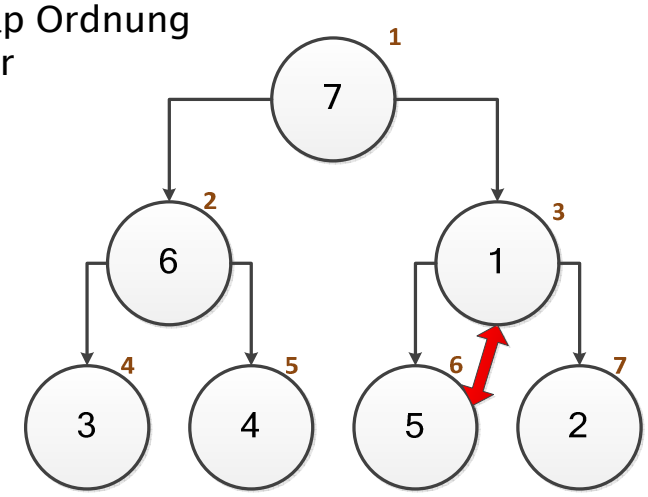
Heapsort

- Stelle Heap Ordnung wieder her
- Setze kleinstes Element an die Spitze und versickere es



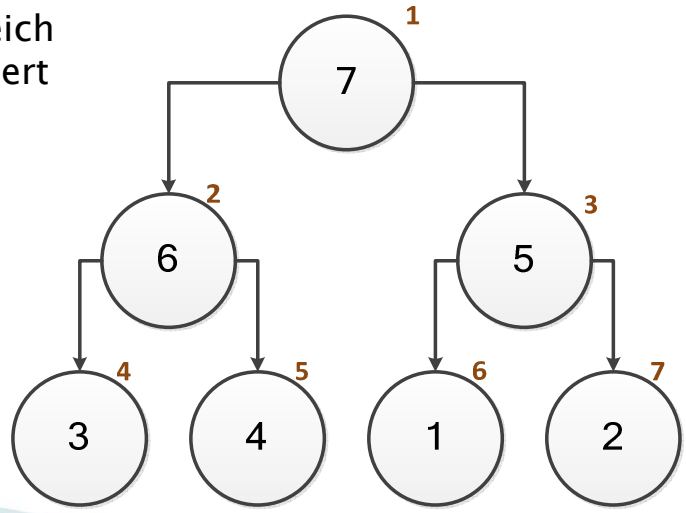
Heapsort

- Stelle Heap Ordnung wieder her



Heapsort

- Erfolgreich umsortiert

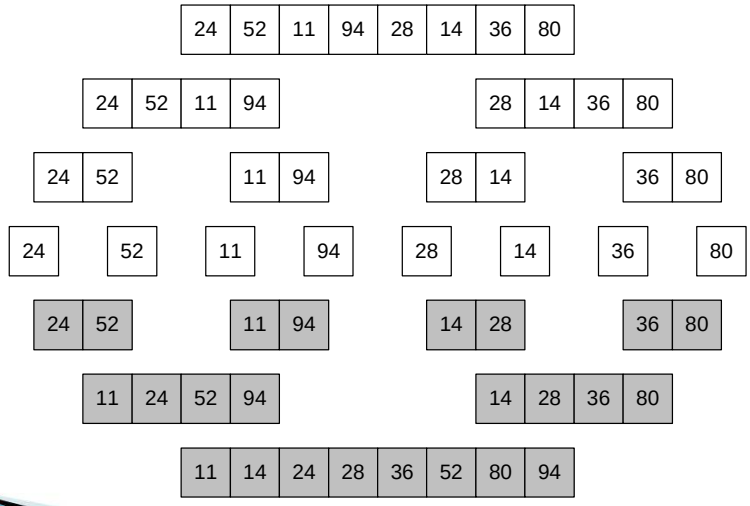


Mergesort

- Rekursiv
- Braucht 2N Speicherplatz

- Prinzip:
  - Daten werden in 2 Teile geteilt, jeder Teil wird durch rekursive Anwendung sortiert
  - Einzelne sortierte Teile werden dann zusammengeschlichtet (merged)

Mergesort



Testbench ALTestBench

- Array von Zufallszahlen, das sortiert werden soll
  - Erzeuge CArray(maxAnzahl)
  - Initialisiere CArray(maxAnzahl, maxValue)
  - StartTimer()
  - Sort()
  - StopTimer()
  - OutputArray()

  - Ausgabe
  - Jede Messung 10x wiederholen (Statistik!)

Übung 2a

- Implementiere folgende Sortierverfahren
  - Selection Sort
  - Bubble Sort
  - Insertion Sort
- Und untersuche ihr Sortierverhalten bei 100, 1000, 10000 und 100000 Werten
- Benutze dazu die Timing Klasse

Übung 2b

- Schreibe eine einfache Heap-Implementierung
- Sortiere 100 integer Werte