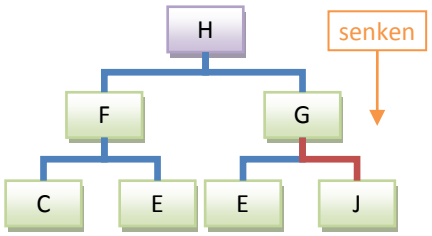
BÄUME

Halde (heap)

Ansicht als Binärbaum



Ansicht als Reihung

H	F	G	C	E	E	J

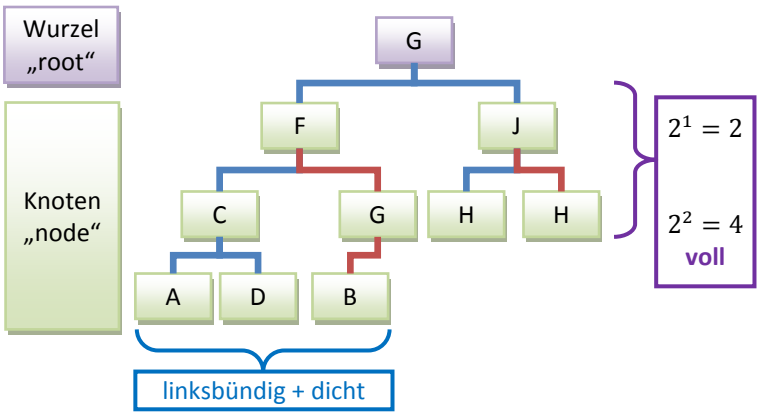
Vorgänger: $reihung[i/2]$
Nachfolger: $reihung[2i], [2i+1]$

Haldenbedingung:
Keine Komponente grösser als der Vorgänger

senken: Vorgang, der eine Halde, die an der Spitze „gestört" ist (Fast-Halde), zu einer richtigen Halde macht

Haldenbedingung-Verletzung

Bäume



Binärbaum:
Jeder Knoten besitz max. 2 Nachfolger
Jeder Knoten hat genau einen Vorgänger (ausgenommen Wurzel)

voll:	wenn jede obere Ebene k genau 2^k Knoten besitzt
komplett:	wenn er voll ist und in der untersten Ebene alle Knoten „linksbündig + dicht" sind
sortiert:	kein Knoten links hat einen grösseren Schlüssel kein Knoten rechts hat einen kleineren Schlüssel
streng sortiert:	gleiche Knoten müssen rechts sein
unsortiert	
Tiefe	= Anzahl Ebenen
Gewicht	= Anzahl Knoten

Übersicht

Suchen in sortierten Bäumen

worst	average	best
$O(n)$	$O(\log n)$	

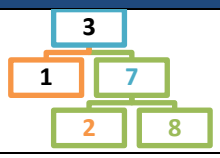
average	Suchen	Einfügen
sortiertes Array	$O(\log n)$	$O(n)$
sortierte verkettete liste	$O(n)$	$O(1)$
streng sortierter Binärbaum	$O(\log n)$	$O(1)$

Sortieren mit Bäumen

Darstellungsmöglichkeiten

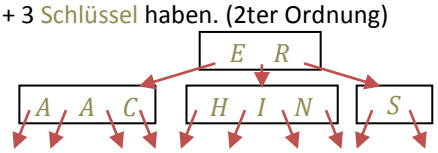
mit Referenzen	**als Halde (heap)**
`class Node{ Node left, right; Comparable data;}`	
Vorteil: Einfach zum Vorstellen	Vorteil: kein Speicher für Referenzen

Baumsort

Durchwanderung			
inorder	links, operation, rechts	1 3 2,7,8	
preorder	operation, links, rechts	3 1 7,2,8	
postorder	links, rechts, operation	1 2,8,7 3	

2-3-4 Bäume

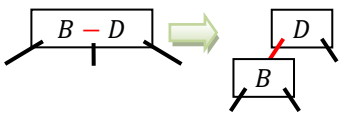
Knoten können bis zu 4 Referenzen + 3 Schlüssel haben. (2ter Ordnung)



Gutes Beibehalten der Ausgeglichenheit
komplexe Implementation

Rot-Schwarz-Bäume

Idee: 2-3-4 Bäume als Binärbaum



Rot: Spezialkanten
Schwarz: Originalkanten

längster Ast ist maximal doppelt so lang wie der kürzeste

B-Bäume

beliebig viele Schlüssel
n-ter Ordnung hat m Schlüssel:
$n \leq m \leq 2n$