Karlsruhe Institute of Technology

# Tutorium 7: (Ganzzahliges-) Sortieren, Heaps

Matthias Schimek | 13. Juni 2017

TUTORIUM ZUR VORLESUNG ALGORITHMEN I IM SS17

# Gliederung


1. Ganzzahliges Sortieren
2. Prioritätslisten
3. Aufgabe

# Ganzzahliges Sortieren

**Satz** Vergleichsbasiertes Sortieren benötigt $\Omega(n \log n)$ Vergleiche

- Geht es anders?
- $\Rightarrow$ Ganzzahliges Sortieren

# Ganzzahliges Sortieren

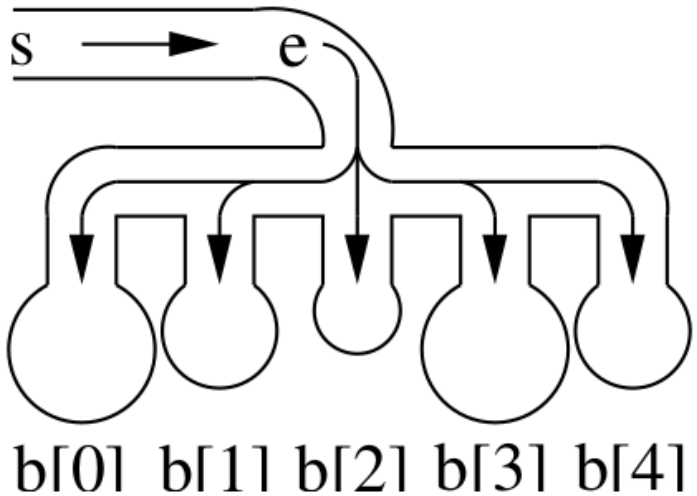
**Satz** Vergleichsbasiertes Sortieren benötigt $\Omega(n \log n)$ Vergleiche

- Geht es anders?
- $\Rightarrow$ Ganzzahliges Sortieren

# Bucketsort

**Procedure** KSort($s$ : Sequence **of** Element)
  b=$\langle\langle\rangle,\ldots,\langle\rangle\rangle$ : **Array** $[0..K-1]$ **of** Sequence **of** Element
  **foreach** $e \in s$ **do** $b[\text{key}(e)]$.pushBack($e$)
  $s :=$ concatenation of $b[0],\ldots,b[K-1]$

Zeit: $O(n+K)$




1

[4] Folien 'Algorithmen I' SS 2017, KIT

# Radixsort

*Least-Significant-Digit*-Radixsort

- Idee Bucketsort auf einzelne Stellen nacheinander anwenden
- Vorraussetzung: Stabilität von Bucketsort
- iterativ
- Laufzeit bei $d$-stelligem Schlüssel?

*Most-Significant-Digit*-Radixsort

- Idee Bucketsort auf MSB anwenden
- rekursiv weitersortieren

# Radixsort

*Least-Significant-Digit*-Radixsort

- Idee Bucketsort auf einzelne Stellen nacheinander anwenden
- Vorraussetzung: Stabilität von Bucketsort
- iterativ
- Laufzeit bei $d$-stelligem Schlüssel?

*Most-Significant-Digit*-Radixsort

- Idee Bucketsort auf MSB anwenden
- rekursiv weitersortieren

# Pancake-Sort

Gegeben sind $n$ Pancakes in unterschiedlicher Größe und gestapelt. Man hat einen Pancake-Flipper zur Verfügung, mit dem man die obersten $i$ Pancakes umdrehen kann.

- Entwickle einen schnellen Algorithmus, um die Pancakes zu sortieren.

# Prioritätsliste

## Prioritätsliste

- Verwaltet Menge $M$ von Element/Schlüssel Paaren $(k, e)$
- *insert*(): Einfügen von neuen Elementen in $M$
- *deleteMin*(): Löschen des Elements mit kleinestem Schlüssel

# Binary Heaps

## Binary Heaps

- (Binärer) Baum mit Eigenschaft $\forall v : \text{parent}(v) \leq v$
- Implizite Baumrepräsentation
- Einfügen/Löschen: Änderungen nur entlang eines Wurzel-Blatt-Pfades
- Vollständiger Binärbaum hat Höhe $\log n$

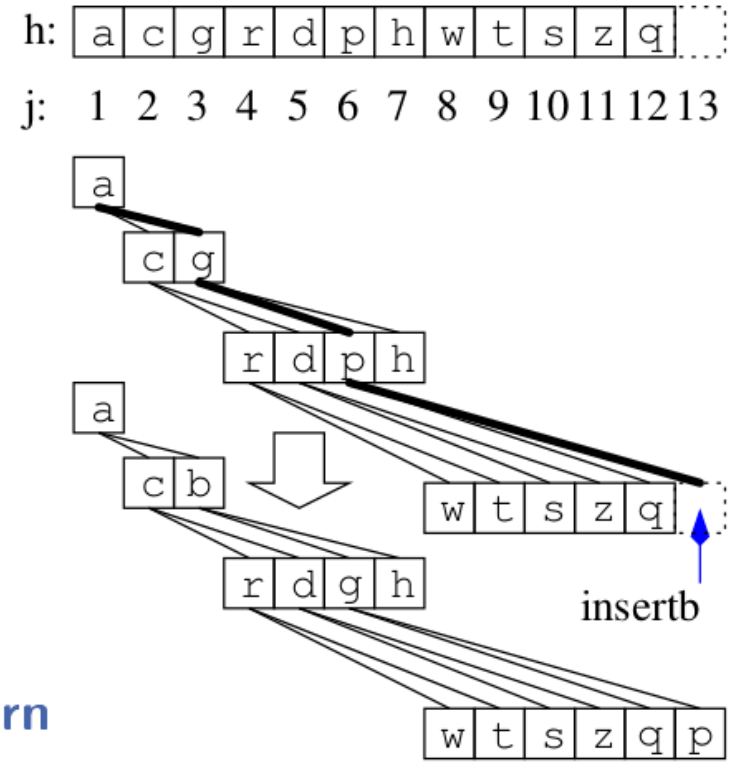
  $\Rightarrow$ *insert*/*delete* in $\mathcal{O}(\log n)$

## Binary Heaps

- (Binärer) Baum mit Eigenschaft $\forall v : \ \text{parent}(v) \leq v$
- Implizite Baumrepräsentation
- Einfügen/Löschen: Änderungen nur entlang eines Wurzel-Blatt-Pfades
- Vollständiger Binärbaum hat Höhe $\log n$

  $\Rightarrow$ *insert*/*delete* in $\mathcal{O}(\log n)$

# Heap - Insertion




**Procedure** insert($e$ : *Element*)
  **assert** $n < w$
  $n{+}{+}$ ; $h[n] := e$
  siftUp($n$)

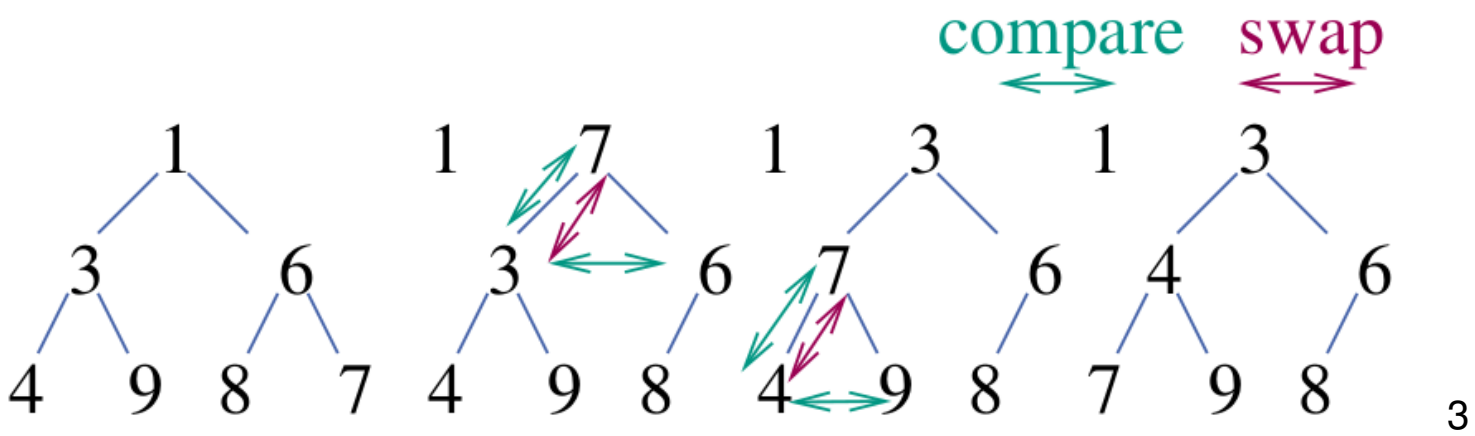
**Procedure** siftUp($i : \mathbb{N}$)
  **assert** the heap property holds
    except maybe at position $i$
  **if** $i = 1 \vee h[\lfloor i/2 \rfloor] \leq h[i]$ **then return**
  swap($h[i], h[\lfloor i/2 \rfloor]$)
  siftUp($\lfloor i/2 \rfloor$)

2

[1] Folien 'Algorithmen I' SS 2017, KIT

# Heap - Deletion

```
Function deleteMin : Element
    result=h[1]  : Element
    h[1]:= h[n];   n−−
    siftDown(1)
    return result
```

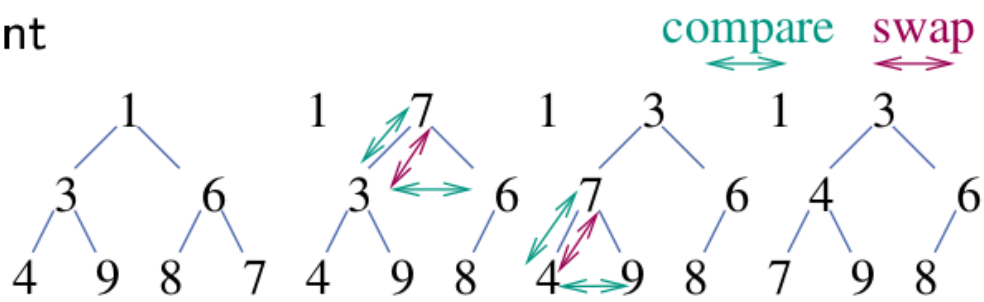



3

[2] Folien 'Algorithmen I' SS 2017, KIT

# Heap - Deletion

```
Function deleteMin : Element
    result=h[1]  : Element
    h[1]:= h[n];   n−−
    siftDown(1)
    return result
```




```
Procedure siftDown(i : ℕ)
    assert heap property except, possibly at j = 2i and j = 2i + 1
    if 2i ≤ n then                                          // i is not a leaf
        if 2i + 1 > n ∨ h[2i] ≤ h[2i + 1] then m:= 2i else m:= 2i + 1
        assert ∄sibling(m) ∨ h[sibling(m)] ≥ h[m]
        if h[i] > h[m] then                                 // heap property violated
            swap(h[i], h[m])
            siftDown(m)
    assert the heap property holds for the subtree rooted at i
```

4

[3] Folien 'Algorithmen I' SS 2017, KIT

# Heap - Build

- Idee?

[5] Folien 'Algorithmen I' SS 2017, KIT

# Heap - Build

- Idee?
- $n$ Aufrufe von $insert()$ $\Rightarrow \mathcal{O}(n \log n)$

[5] Folien 'Algorithmen I' SS 2017, KIT

# Heap - Build

- Idee?

- $n$ Aufrufe von $insert()$ $\Rightarrow \mathcal{O}(n \log n)$
- Geht es besser?

[5] Folien 'Algorithmen I' SS 2017, KIT

# Heap - Build

- Idee?
- $n$ Aufrufe von $insert()$ $\Rightarrow \mathcal{O}(n \log n)$
- Geht es besser?

**Procedure** buildHeapBackwards
    **for** $i := \lfloor n/2 \rfloor$ **downto** 1 **do** siftDown($i$)

[5]

- Mittels genauer Analyse $\Rightarrow$ Konstruktion in $\mathcal{O}(n)$

[5] Folien 'Algorithmen I' SS 2017, KIT

# Aufgaben

- Bilde alle möglichen binären Min-Heaps mit den Zahlen $< 9, 5, 3, 4 >$
- Stelle die Min-Heaps in der implizten Array-Schreibweise dar

# Aufgaben

- Zeichne den (Min-) Heap der nach folgenden Operationen entsteht:
  *insert*(4), *insert*(55), *insert*(1),
  *insert*(9), *insert*(23), *deleteMin*,
  *insert*(3), *insert*(92)

# Aufgabe

Entwerfe einen effizienten Sortieralgorithmus, der einen binären Heap verwendet.

# Aufgabe - Bulk Insert

Gegeben sei ein binärer Heap der $n = 2^m - 1$ Elemente enthält.
Es sollen nun $k$ Elemente auf einmal eingefügt werden.

- Gebt ein Verfahren an, mit dem man das Einfügen in $\mathcal{O}\min\{k \log k + \log n, k + \log n \log k\}$ Schritten erledigen kann.

# Aufgabe - Bulk Insert

Gegeben sei ein binärer Heap der $n = 2^m - 1$ Elemente enthält.
Es sollen nun $k$ Elemente auf einmal eingefügt werden.

- Gebt ein Verfahren an, mit dem man das Einfügen in $\mathcal{O}\min\{k \log k + \log n, k + \log n \log k\}$ Schritten erledigen kann.