Fachhochschule Nordwestschweiz

05 Priority Queues (Teil 2)

Algorithmen und Datenstrukturen 2

- 2 Vorlesungen
  - 1. Teil bis Kapitel 5.7
  - 2. Teil ab Kapitel 5.8
- Dokumente
  - Skript
  - Programmieraufgaben 1 und 2
  - Arbeitsblatt

Priority Queue - Prioritätswarteschlange

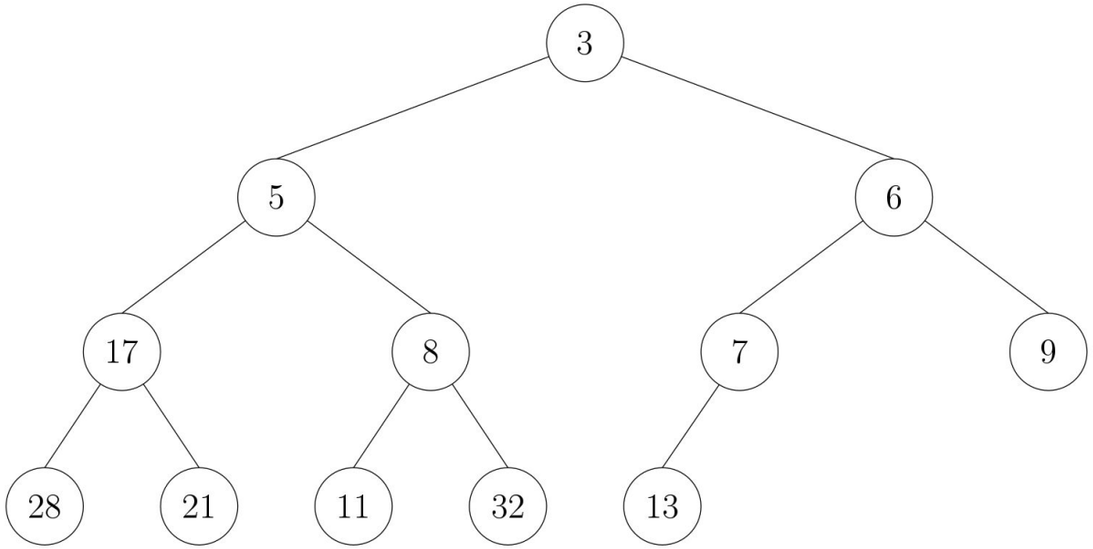
Minimum Priority Queue

min()	gibt das kleinste Element
removeMin()	gibt und entfernt das kleinste Element
add(elem)	fügt ein Element elem hinzu
size()	gibt die Anzahl Elemente in der Priority Queue

Maximum Priority Queue

max()	gibt das grösste Element
removeMax()	gibt und entfernt das grösste Element

Neue Datenstruktur – **Der Min-Heap**



Struktur-Eigenschaft:

Vollständiger Binärbaum, mit Ausnahme der untersten Stufe; dort ist er von links nach rechts aufgefüllt.

Ordnungs-Eigenschaft:

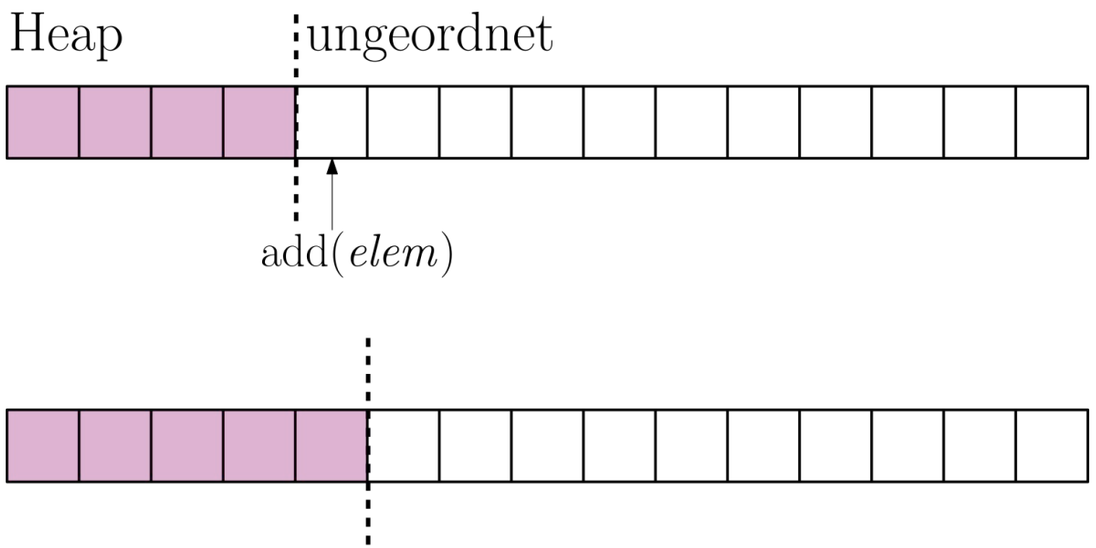
Der Schlüssel jedes Knotens ist kleiner gleich der Schlüssel seiner beider Kinder (falls vorhanden).

Aufbau eines Heaps in einem gefüllten Array

Aufbau eines Heaps in einem gefüllten Array

Erste Idee

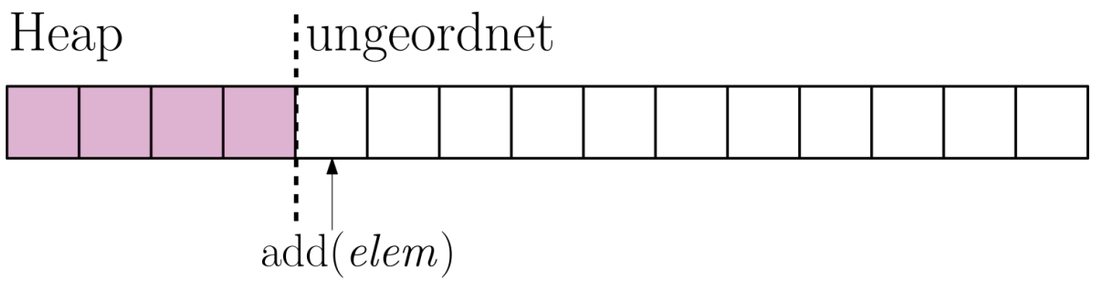
- Heap wächst *von vorne nach hinten* (im Baum Top-Down)
- Elemente nacheinander in den Heap “einfügen”



Aufbau eines Heaps in einem gefüllten Array

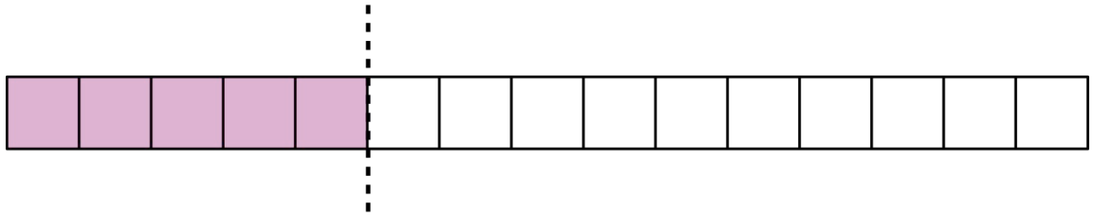
Erste Idee

- Heap wächst *von vorne nach hinten* (im Baum Top-Down)
- Elemente nacheinander in den Heap “einfügen”



Aufwand

- O(n log n), weil n Mal siftUp()
- Genauer: siftUp() ~ log i Schritte, für i = heapsize
  für i > n/2: > log n/2 Schritte
  Total: > n/2 * log n/2 Schritte = O(n log n)

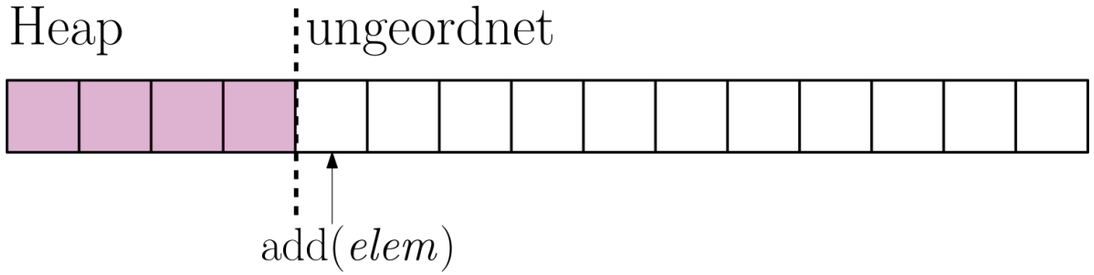


Aufbau eines Heaps in einem gefüllten Array

Mit diesem Vorgehen sparen wir Platz, aber keine Zeit!

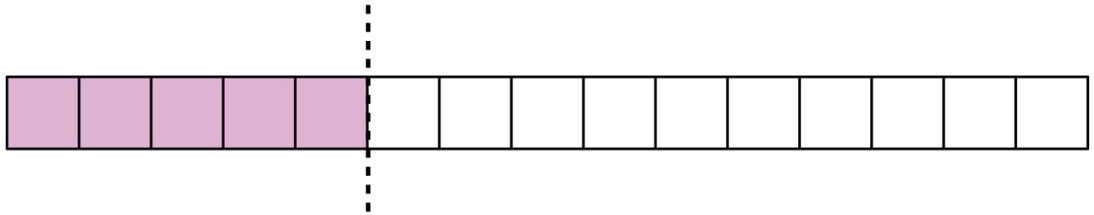
Erste Idee

- Heap wächst *von vorne nach hinten* (im Baum Top-Down)
- Elemente nacheinander in den Heap “einfügen”



Aufwand

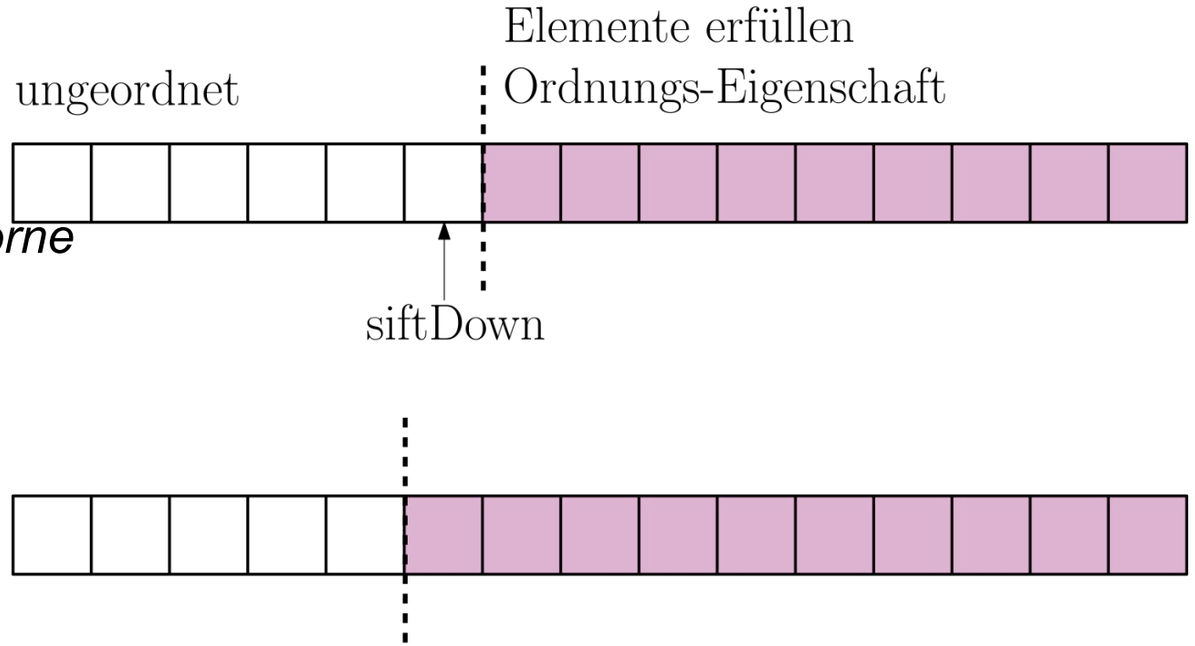
- O(n log n), weil n-1 Mal siftUp()
- Genauer: siftUp() ~ log i Schritte, für i = heapsize
  für i > n/2: > log n/2 Schritte
  Total: > n/2 * log n/2 Schritte = O(n log n)



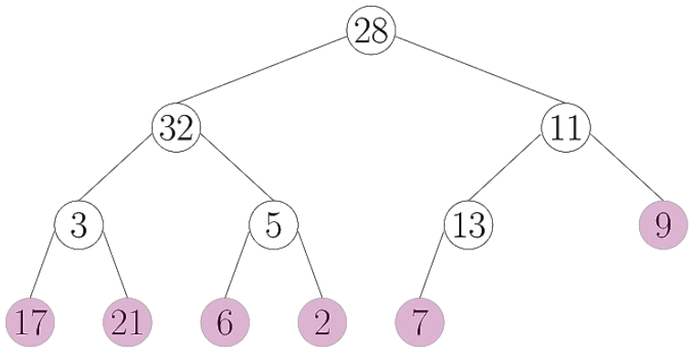
Aufbau eines Heaps in einem gefüllten Array

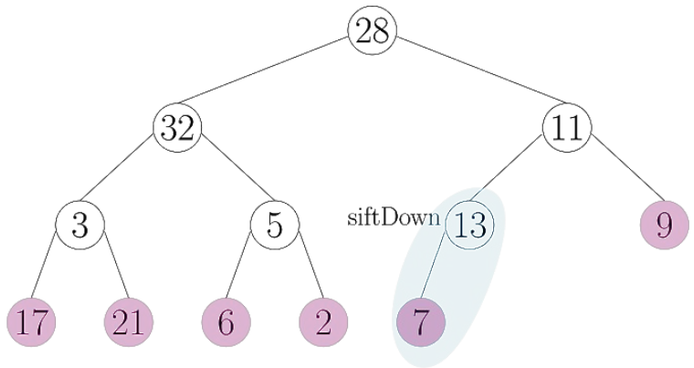
Algorithmus von Floyd

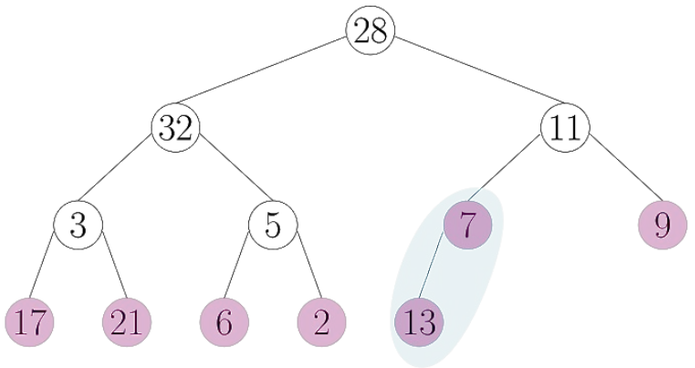
- Heap wächst *von hinten nach vorne* (im Baum Bottom-Up)
- Elemente nacheinander “versickern”



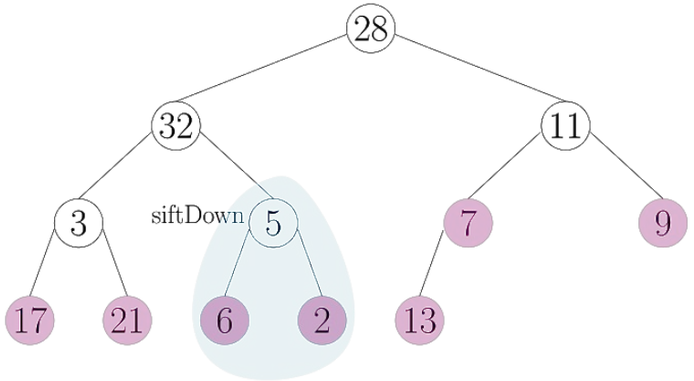
Aufbau eines Heaps in einem gefüllten Array

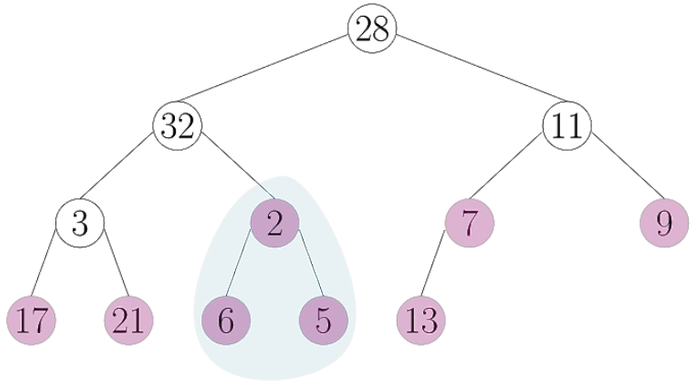




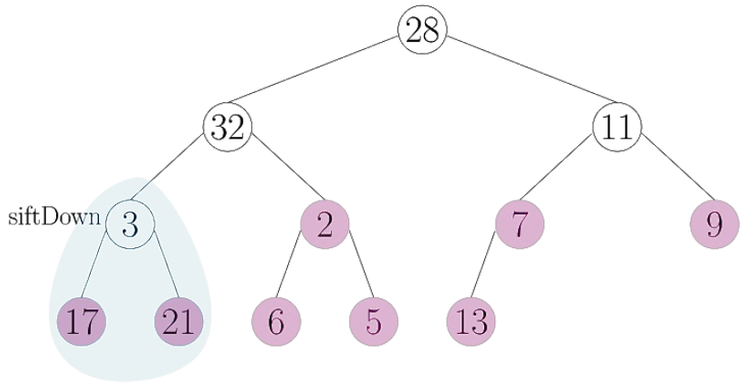


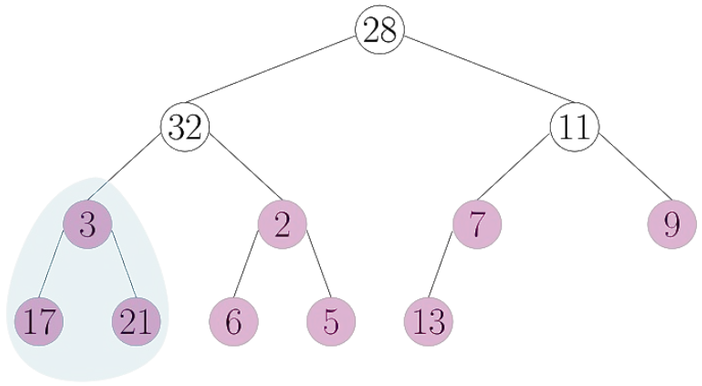
Aufbau eines Heaps in einem gefüllten Array



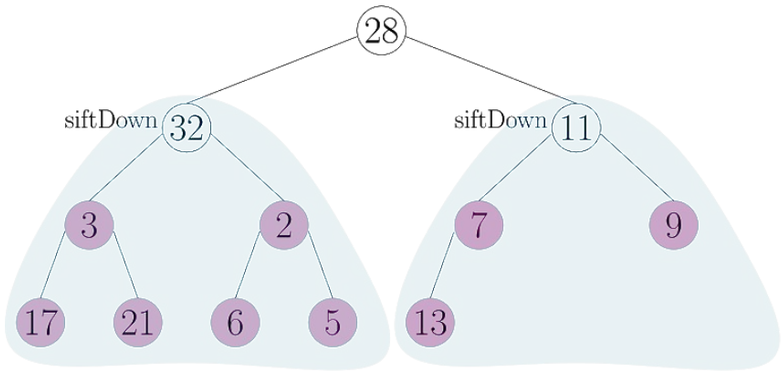


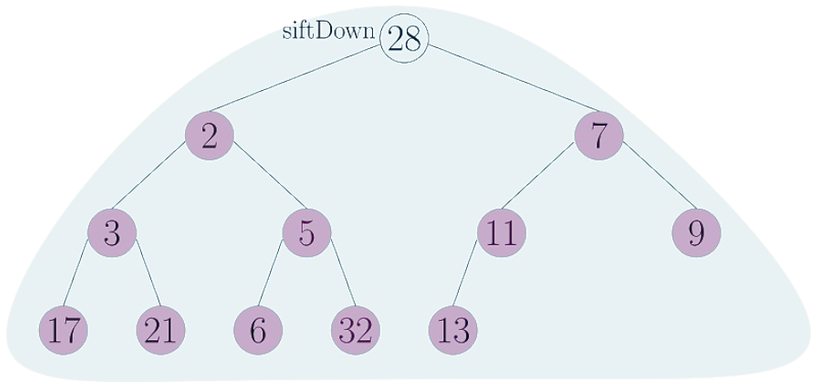
Aufbau eines Heaps in einem gefüllten Array

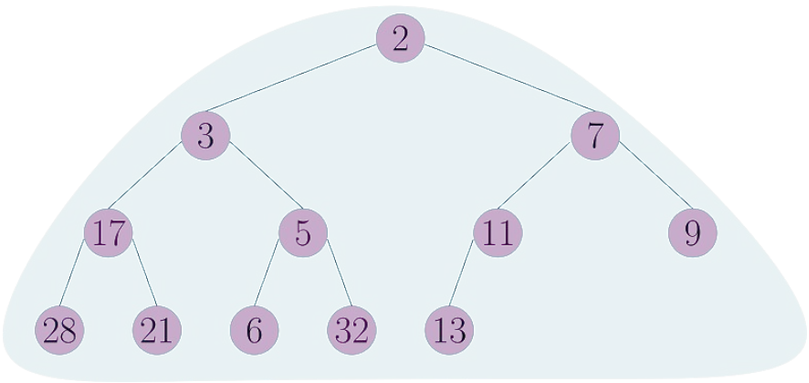




Aufbau eines Heaps in einem gefüllten Array



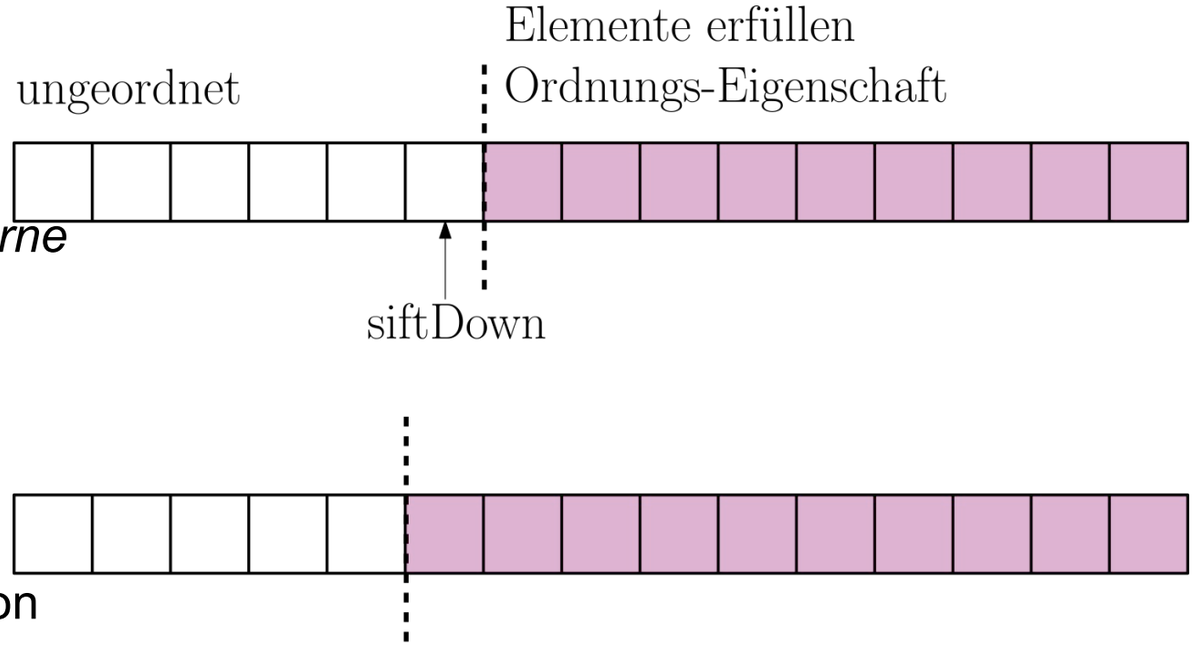




Aufbau eines Heaps in einem gefüllten Array

Algorithmus von Floyd

- Heap wächst *von hinten nach vorne* (im Baum Bottom-Up)
- Elemente nacheinander versickern



Aufwand

Im Gegensatz zur ersten Idee, wo viele Elemente von weit unten im Baum bis zur Wurzel hinaufwandern, lässt der Algorithmus von Floyd **viele Elemente in kleinen Teilbäumen versickern** und ruft nur für **wenige Elemente ein siftDown von grosser Höhe** auf. Dies führt zu einer linearen Laufzeit O(n).

Fachhochschule
Nordwestschweiz

Aufbau eines Heaps in einem gefüllten Array

Arbeitsblatt - Teil 2:

Lösen Sie Aufgabe 7 zur Laufzeit des Algorithmus von Floyd.

Aufbau eines Heaps in einem gefüllten Array

```
class Heap<K> implements PriorityQueue<K> {
    private HeapNode<K>[] heap;
    private int size;

    // Konstruktor nach Algorithmus von Floyd
    Heap(HeapNode<K>[] elems) {




    }
}
```

Aufbau eines Heaps in einem gefüllten Array

```
class Heap<K> implements PriorityQueue<K> {
    private HeapNode<K>[] heap; // Array to store the heap elements
    private int size; // Number of elements currently stored in heap

    // Konstruktor nach Algorithmus von Floyd
    Heap(HeapNode<K>[] elems) {
        this.heap = elems;
        this.size = elems.length;
        for(int i=size/2; i>= 0; i--){
            siftDown(i);
        }
    }
}
```

Fachhochschule
Nordwestschweiz

Sortieren mit einem Heap

Sortieren mit einem Heap

Erste Idee

1. Alle Elemente aus dem Input-Array in einen Min-Heap einfügen
2. Elemente in geordneter Reihenfolge aus dem Heap herauslesen und in Input-Array zurückschreiben

```
public int[] HeapSort(int[] values ) {
  int[] sorted = new int[values.length] ;
  MinHeap q = new MinHeap(values);
  for (int i = 0; i < values.length; i++) {
      sorted[i] = q.deleteMin();
  }
  return sorted;
}
```

Sortieren mit einem Heap

Erste Idee

1. Alle Elemente aus dem Input-Array in einen Min-Heap einfügen
2. Elemente in geordneter Reihenfolge aus dem Heap herauslesen und in Input-Array zurückschreiben

```
public int[] HeapSort(int[] values ) {
  int[] sorted = new int[values.length] ;
  MinHeap q = new MinHeap(values);
  for (int i = 0; i < values.length; i++) {
      sorted[i] = q.deleteMin();
  }
  return sorted;
}
```

Zeit
- O(n log n)
- O(log n) pro deleteMin()

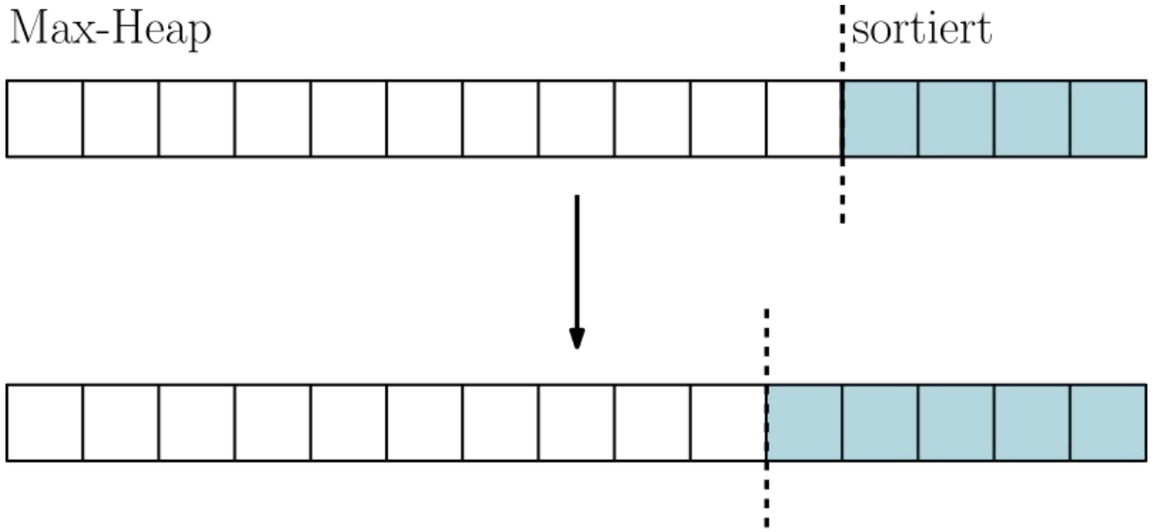
Platz
- O(n) zusätzlich

Wie können wir Platz sparen?

Sortieren mit einem Heap

Effizienter

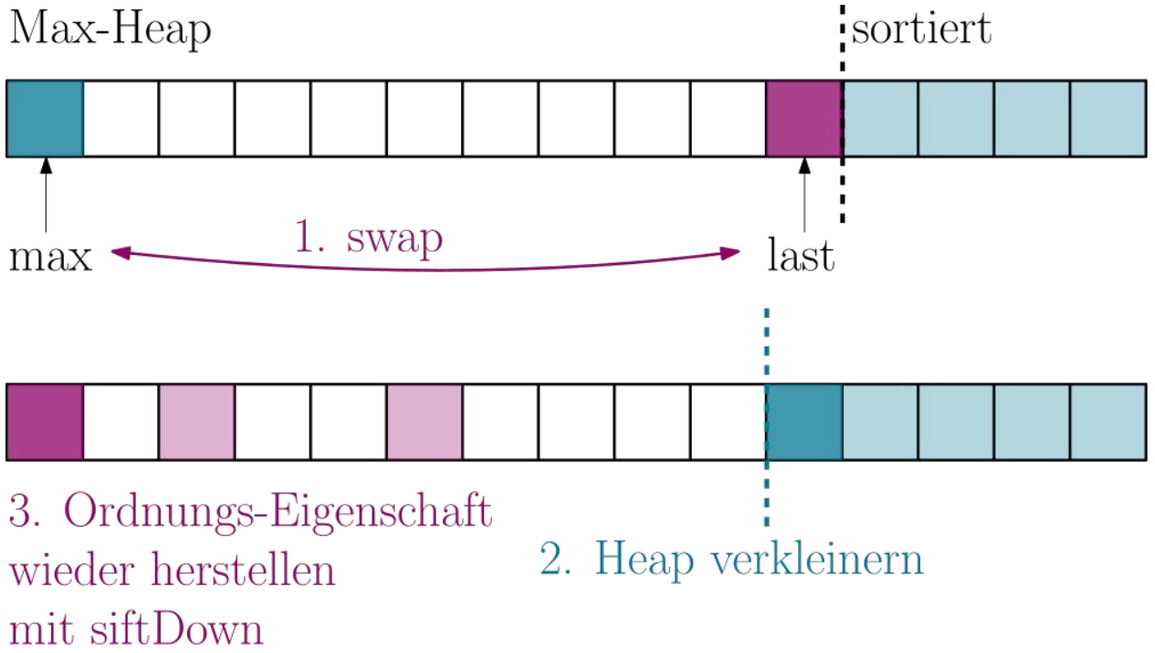
1. Max-Heap direkt im Input-Array bauen
2. Elemente in geordneter Reihenfolge aus dem Heap herauslesen und hinten im Input-Array speichern



Sortieren mit einem Heap

Effizienter

1. Max-Heap direkt im Input-Array bauen
2. Elemente in geordneter Reihenfolge aus dem Heap herauslesen und hinten im Input-Array speichern



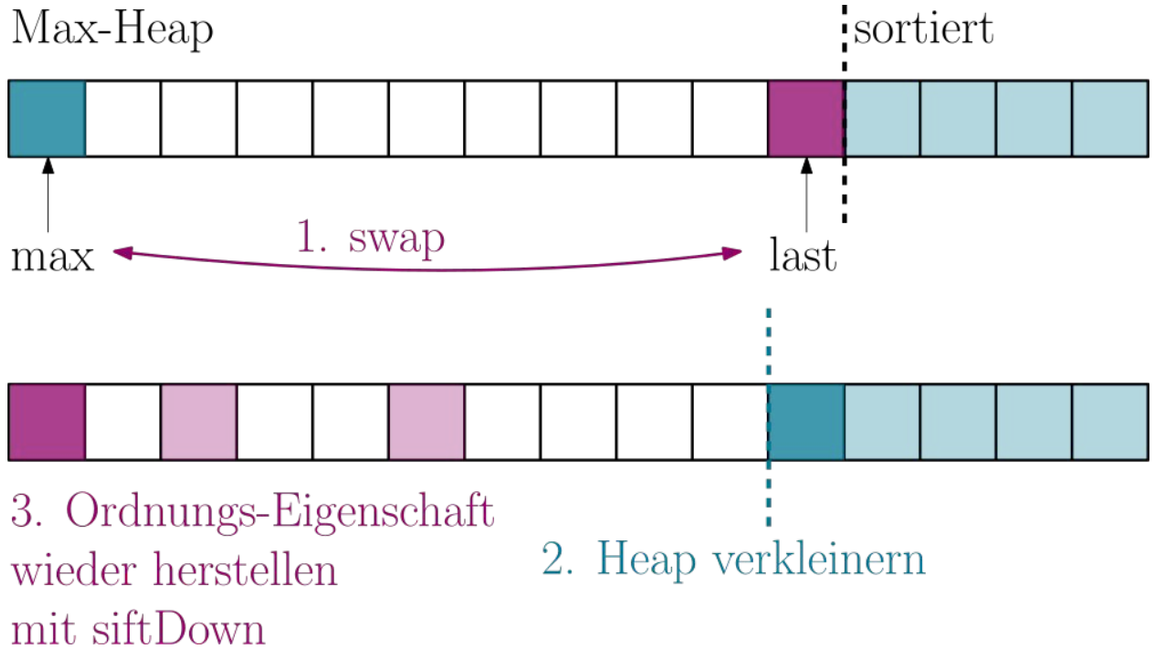
Sortieren mit einem Heap

Effizienter

1. Max-Heap direkt im Input-Array bauen
2. Elemente in geordneter Reihenfolge aus dem Heap herauslesen und hinten im Input-Array speichern

Zeit

- O(n log n)
- swap: O(1)
- verkleinern: O(1)
- siftDown: O(log n)



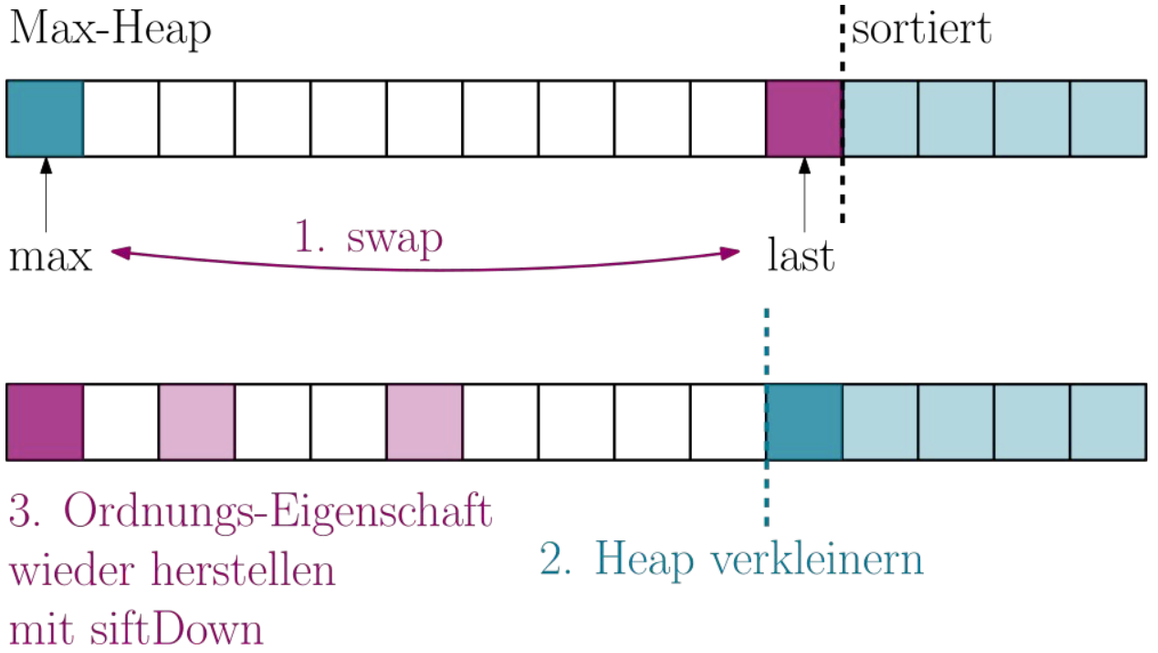
Sortieren mit einem Heap

Effizienter

1. Max-Heap direkt im Input-Array bauen
2. Elemente in geordneter Reihenfolge aus dem Heap herauslesen und hinten im Input-Array speichern

Zeit

- O(n log n)
- swap: O(1)
- verkleinern: O(1)
- siftDown: O(log n)



Platz

- O(1) zusätzlich
- in place / in situ

Hausaufgaben

Arbeitsblatt - Teil 2

- Aufgabe 8

Programmieren

- Programmieraufgabe 2 (HeapSort)