DSAL - 4. Globalübung

Benjamin Kaminski, Tim Quatmann

Stoff für Präsenzübung

- Bis inkl. VL 10
- Bis inkl. ÜB 6
- Zulässige Hilfsmittel: Ein seblst beschriebenes Din A4 Blatt.

15. Mai 2018

Agenda

1. Fixpunktinduktion
2. Mergesort
3. Heapsort
4. Prioritätswarteschlangen

Fixpunktinduktion

Fixpunktinduktion

$$T(n) = 1 \quad , \quad 1 \leq n < 16$$

$$T(n) = 2\,T\left(\left\lfloor \tfrac{n}{4} \right\rfloor\right) + T\left(\left\lfloor \tfrac{n}{2} \right\rfloor\right) + n$$

$$f(n) = 1 + 2n \log_2(n)$$

$$\Phi(T)(n) = \begin{cases} 1, & 1 \leq n \leq 16 \\ 2\,T\left(\left\lfloor \tfrac{n}{4} \right\rfloor\right) + T\left(\left\lfloor \tfrac{n}{2} \right\rfloor\right) + n, & \text{sonst} \end{cases}$$

$$\Phi(f)(n) = \begin{cases} 1, & 1 \leq n < 16 \\ 2\left(1 + 2\left\lfloor \tfrac{n}{4} \right\rfloor \log_2\left(\left\lfloor \tfrac{n}{4} \right\rfloor\right)\right) + \left(1 + 2\left\lfloor \tfrac{n}{2} \right\rfloor \log_2\left(\left\lfloor \tfrac{n}{2} \right\rfloor\right)\right) + n \end{cases}$$

$$\overset{!}{\leq} 1 + 2n \log_2(n)$$

$$\Phi(f)(n) = \begin{cases} 1, & 1 \le n < 16 \\ 2\left(1+2\lfloor \frac{n}{4} \rfloor \log_2\left(\lfloor \frac{n}{4} \rfloor\right)\right) + \left(1+2\lfloor \frac{n}{2} \rfloor \log_2\left(\lfloor \frac{n}{2} \rfloor\right)\right) + n \end{cases}$$

$$\overset{!}{\le} 1 + 2n\log_2(n)$$

$$\Leftarrow \cancel{2}\left(1 + \frac{\cancel{2}n}{\cancel{4}} \log_2\left(\frac{n}{4}\right)\right) + \left(1 + \frac{\cancel{2}n}{\cancel{2}} \log_2\left(\frac{n}{2}\right)\right) + n$$

$$\le 1 + 2n\log_2(n)$$

$$\Leftrightarrow 2 + n\log_2\left(\frac{n}{4}\right) \cancel{+1} + n\log_2(n/2) + n \le \cancel{1} + 2n\log_2(n)$$

$$\Leftrightarrow 2 + n\log_2(n) - n\log_2(4) + n\log_2(n) - n\log_2(2) \overset{+n}{\le} 2n\log_2(n)$$

$$\Leftrightarrow 2 - 2n \cancel{-n+n} \le 0$$

$$\Leftrightarrow 2 - 2n \le 0 \quad \checkmark \quad \boxed{\text{Fall } n \ge 16}$$

Mergesort

Mergesort

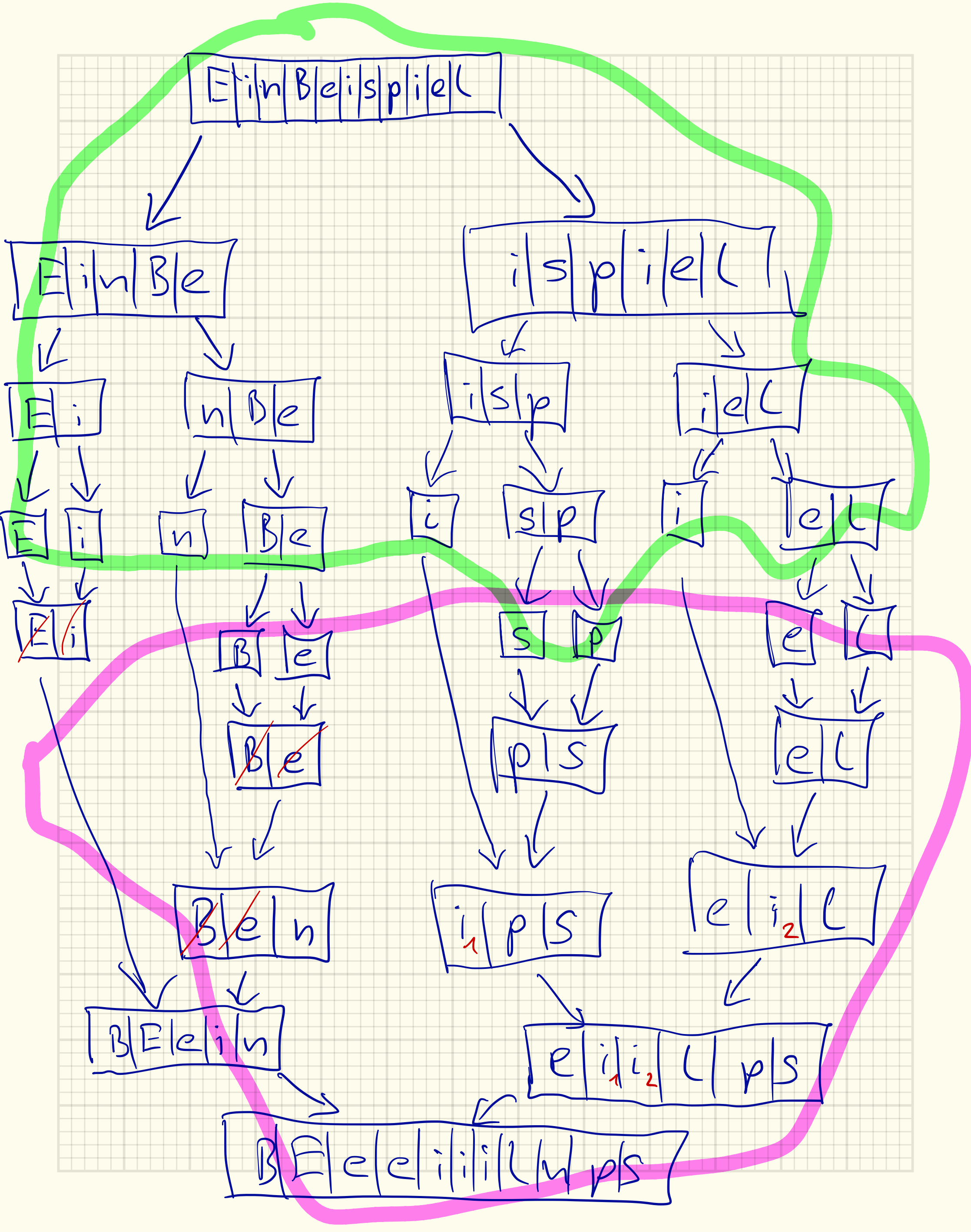
```
def sort(a):
    if len(a) > 2:
        links = sort(a[:floor(len(a)/2)])
        rechts = sort(a[floor(len(a)/2):])
    elif len(a) == 2:
        links = [a[0]]
        rechts = [a[1]]
    else:
        return a[:]
    res = []
    while len(links) > 0 and len(rechts) > 0:
        if links[0] <= rechts[0]:
            res.append(links.pop(0))
        else:
            res.append(rechts.pop(0))
    res += links + rechts
    return res
```

$\Theta(n \log(n))$

$n \sim len(a)$

$S(n) = 2S(\frac{n}{2}) + n$

$1 + 2n \log_2(n)$

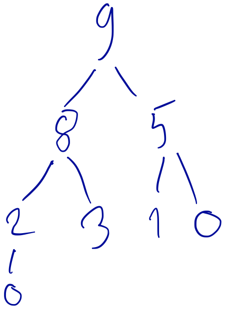

E i n B e i s p i e l
E i n B e
i s p i e l
E i
n B e
i s p
i e l
E
i
n
B e
i
s p
i
e l
E i
B
e
s
p
e
l
B e
p s
e l
B e n
i1 p s
e i2 l
B E e i n
e i1 i2 l p s
B E e e e i i i l n p s

Heapsort

Heaps

Max-Heaps

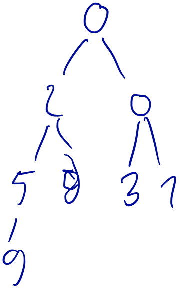
- Schlüssel eines Knotens ist **größer** als die Schlüssel seiner Kinder (oder gleich)
- Alle Ebenen (abgesehen von evtl. der untersten) sind komplett gefüllt
- Die Blätter auf der untersten Ebene sind linksbündig angeordnet



9	8	5	2	3	1	0	0

Min-Heaps

- Schlüssel eines Knotens ist **kleiner** als die Schlüssel seiner Kinder (oder gleich)
- Alle Ebenen (abgesehen von evtl. der untersten) sind komplett gefüllt
- Die Blätter auf der untersten Ebene sind linksbündig angeordnet



0	2	0	5	8	3	7	9

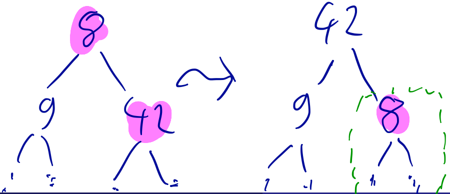
auch Min-Heaps lassen sich als Array repräsentieren

Heapify auf Max-Heaps

Heapgröße

Knotenindex der die Max-Heapeigenschaft verletzt

```
void heapify (int E[], int n, int pos) {
  int next = 2 * pos + 1;
  while (next < n) {
    if (next + 1 < n && E[next + 1] > E[next]) {
      next = next + 1;
    }
    if (E[pos] >= E[next]) {
      break;
    }
    swap(E[pos], E[next]);
    pos = next;
    next = 2 * pos + 1;
  }
}
```



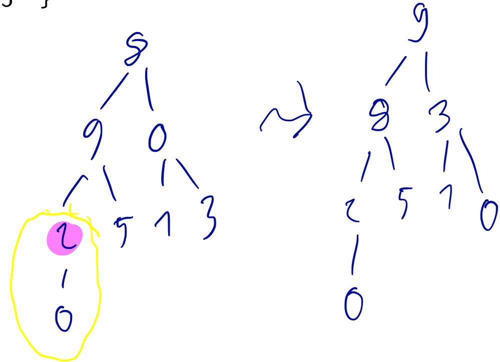
Heapify auf Min-Heaps

```
void heapify (int E[], int n, int pos) {
  int next = 2 * pos + 1;
  while (next < n) {
    if (next + 1 < n && E[next + 1] < E[next]) {
      next = next + 1;
    }
    if (E[pos] <= E[next]) {
      break;
    }
    swap(E[pos], E[next]);
    pos = next;
    next = 2 * pos + 1;
  }
}
```

Heapaufbau auf Max-Heaps

Index vom "letzten" inneren Knoten

```
void buildHeap (int E[]) {
  for (int i = E.length / 2 - 1; i >= 0; i--) {
    heapify(E, E.length, i)
  }
}
```

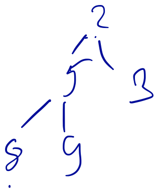


Heapaufbau auf Max-Heaps und Min-Heaps

```
void buildHeap (int E[]) {
  for (int i = E.length / 2 - 1; i >= 0; i--) {
    heapify(E, E.length, i)
  }
}
```

Heapsort

```
void heapSort (int E[]) {
  buildHeap(E);
  for (int i = E.length - 1; i > 0; i--) {
    swap(E[0], E[i]);
    heapify(E, i, 0);
  }
}
```



Max Heaps → Aufst. Sortierung

Min Heaps → Abst. Sortierung

9	8	5	3	2	1	0	0

Prioritätswarteschlangen

Erinnerung: Die Prioritätswarteschlange (I)

(siehe VL)

- Betrachte Elemente, die mit einem Schlüssel (key) versehen sind.
- Jeder Schlüssel sei höchstens an ein Element vergeben.
- Schlüssel werden als Priorität betrachtet.
- Die Elemente werden nach ihrer Priorität sortiert.

Erinnerung: Die Prioritätswarteschlange (II)

Prioritätswarteschlange (priority queue)

- `void insert(PriorityQueue pq, Element e, int k)` fügt das Element `e` mit dem Schlüssel `k` in `pq` ein.
- `Element getMin(PriorityQueue pq)` gibt das Element mit dem kleinsten Schlüssel zurück; benötigt nicht-leere `pq`.
- `void delMin(PriorityQueue pq)` entfernt das Element mit dem kleinsten Schlüssel; benötigt nicht-leere `pq`.
- `Element getElt(PriorityQueue pq, int k)` gibt das Element `e` mit dem Schlüssel `k` aus `pq` zurück; `k` muss in `pq` enthalten sein.
- `void decrKey(PriorityQueue pq, Element e, int k)` setzt den Schlüssel von Element `e` auf `k`; `e` muss in `pq` enthalten sein. `k` muss außerdem kleiner als der bisherige Schlüssel von `e` sein.

Mit Heaps ist eine effiziente Implementierung möglich.

Drei Prioritätswarteschlangenimplementierungen

	Implementierung		
Operation	unsortiertes Array	sortiertes Array	Heap
`isEmpty(pq)`	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$
`insert(pq,e,k)`	$\Theta(1)$	$\Theta(n)^*$	$\Theta(\log(n))$
`getMin(pq)`	$\Theta(n)$	$\Theta(1)$	$\Theta(1)$
`delMin(pq)`	$\Theta(n)^*$	$\Theta(1)$	$\Theta(\log(n))$
`getElt(pq,k)`	$\Theta(n)$	$\Theta(\log(n))^\dagger$	$\Theta(n)$
`decrKey(pq,e,k)`	$\Theta(1)$	$\Theta(n)^*$	$\Theta(\log(n))$

*Beinhaltet das Verschieben aller Elemente „rechts“ von `k`.

†Mittels binärer Suche.

Prioritätswarteschlangen mit Min-Heaps

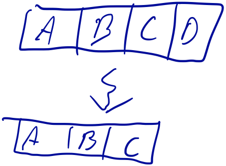
Annahmen:

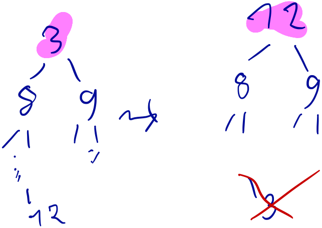
- Die Elemente in der Warteschlange sind Integer
- Element $\widehat{=}$ Schlüssel

- Prioritätswarteschlange wird als Integer-Array `E` repräsentiert
  - `E` entspricht einem Min-Heap.

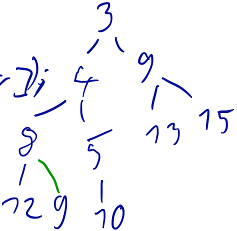
```
int getMin (int E[]) { // Gebe das kleinste Element zurueck
    return E[0];
}
```

```
void delMin (int E[]) { // Loesche das kleinste Element
    int n = E.length - 1;
    swap(E[0], E[n]);
    E.resize(n);
    heapify(E, n, 0);
}
```





```
void insert (int E[], int k) { // Fuege k ein
int pos = E.length;  E.resize(pos+1); E[pos] = k;
while (pos > 0){
   int prev = ⌊(pos-1)/2⌋ // Elternknoten
   if(E[pos] >= E[prev])){
       break;
   }
   swap(E[pos], E[prev]);
   pos = prev;
 }
}
```



```
void insert (int E[], int k) { // Fuege k ein
  int pos = E.length;
  E.resize(pos + 1);
  E[pos] = k;
  while (pos > 0) {
    int prev = floor(pos/2)
    if (E[pos] >= E[prev]) {
      break;
    }
    swap(E[pos], E[prev]);
    pos = prev;
  }
}
```

Nächster Termin

Nächste Vorlesung

Freitag, 18. Mai, 13:15 (H01).

Nächste Globalübung

Dienstag **29.** Mai, 14:15 (Aula 1).