Karlsruher Institut für Technologie

Tutorium 6

Algorithmen I SS 14

Institut für Theoretische Informatik

Heaps

Heaps

(Binärer) Baum mit zusätzlicher Eigenschaft:

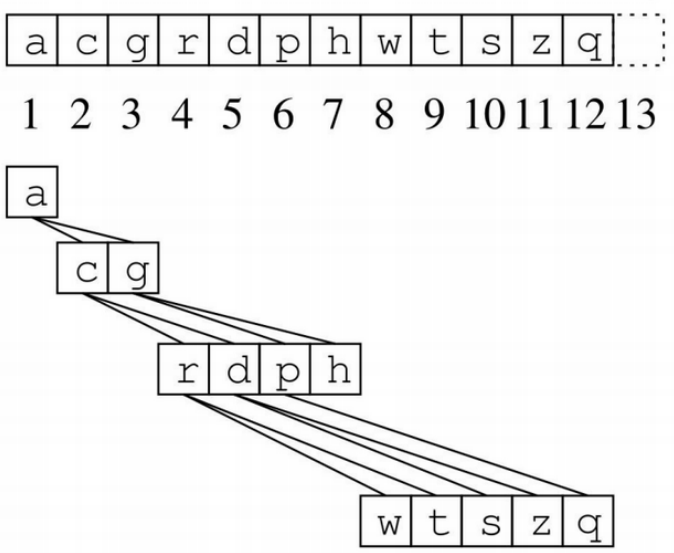
- Eltern sind kleiner gleich ihren Kindern: $\forall v : \mathit{parent}(v) \leq v$
- Auch umgekehrt möglich (Max-Heap)

Heapoperationen

- Minimum finden in $\mathcal{O}(1)$ (Wurzel betrachten)
- Einfügen in $\mathcal{O}(\log n)$
- Wurzel extrahieren in $\mathcal{O}(\log n)$
- Heap aufbauen in $\mathcal{O}(n)$

Implementierung

Im Computer effizient als Array darstellbar:



Baumnavigation in Arraydarstellung

leftChild(i): heap[$2i$]

rightChild(i): heap[$2i + 1$]

parent(i): heap[$\lfloor \frac{i}{2} \rfloor$]

- Ein Heap der Höhe h hat mindestens 2^{h-1} und maximal $2^h - 1$ Elemente
- Ein Heap mit n Elementen hat die Höhe $1 + \lfloor \log_2(n) \rfloor$

Heapoperationen: insert

1. Einfügen des neuen Elements an letzter Stelle
2. Heap-Eigenschaft ist jetzt möglicherweise dort verletzt
3. Schiebe Element so lange nach *oben*, bis Heap-Eigenschaft wiederhergestellt ist (siftUp)
4. Also: Vergleiche jeweils mit dem Elternelement und vertausche, falls es größer ist

Heapoperationen: deleteMin

1. Lösche erstes Element und ersetze es durch letztes Element
2. Heap-Eigenschaft nun möglicherweise dort verletzt
3. Schiebe Element so lange nach *unten*, bis Heap-Eigenschaft wiederhergestellt ist (siftDown)
4. Also: Vergleiche jeweils mit Kindelementen und vertausche mit dem kleineren

Heapoperationen: buildHeap

Karlsruher Institut für Technologie

1. Füge Elemente ohne Beachtung der Heap-Eigenschaft ein
2. Stelle Heap-Eigenschaft ebenenweise von unten wieder her
3. Führe siftDown auf alle Elemente, beginnend bei der vorletzten Ebene aus

Heapsort

- Benutze *buildHeap()* um Heap aufzubauen
- n mal *deleteMin()* um Minimum zu extrahieren
- insgesamt also Laufzeit $\mathcal{O}(n \log n)$
- echt in-place

Aufgabe: Heapsort

Sortiere die Folge $\langle 65, 32, 85, 37, 84, 64, 3, 31, 47 \rangle$ mit Heapsort aufsteigend.

Kreativaufgabe: Pancake-Sorting

- **gegeben:** Stapel von n Pancakes, Pancake-Flipper mit dem man die obersten k Pancakes drehen kann ($k \leq n$).
- **gesucht:** Schnelles Verfahren zum Sortieren der Pancakes.

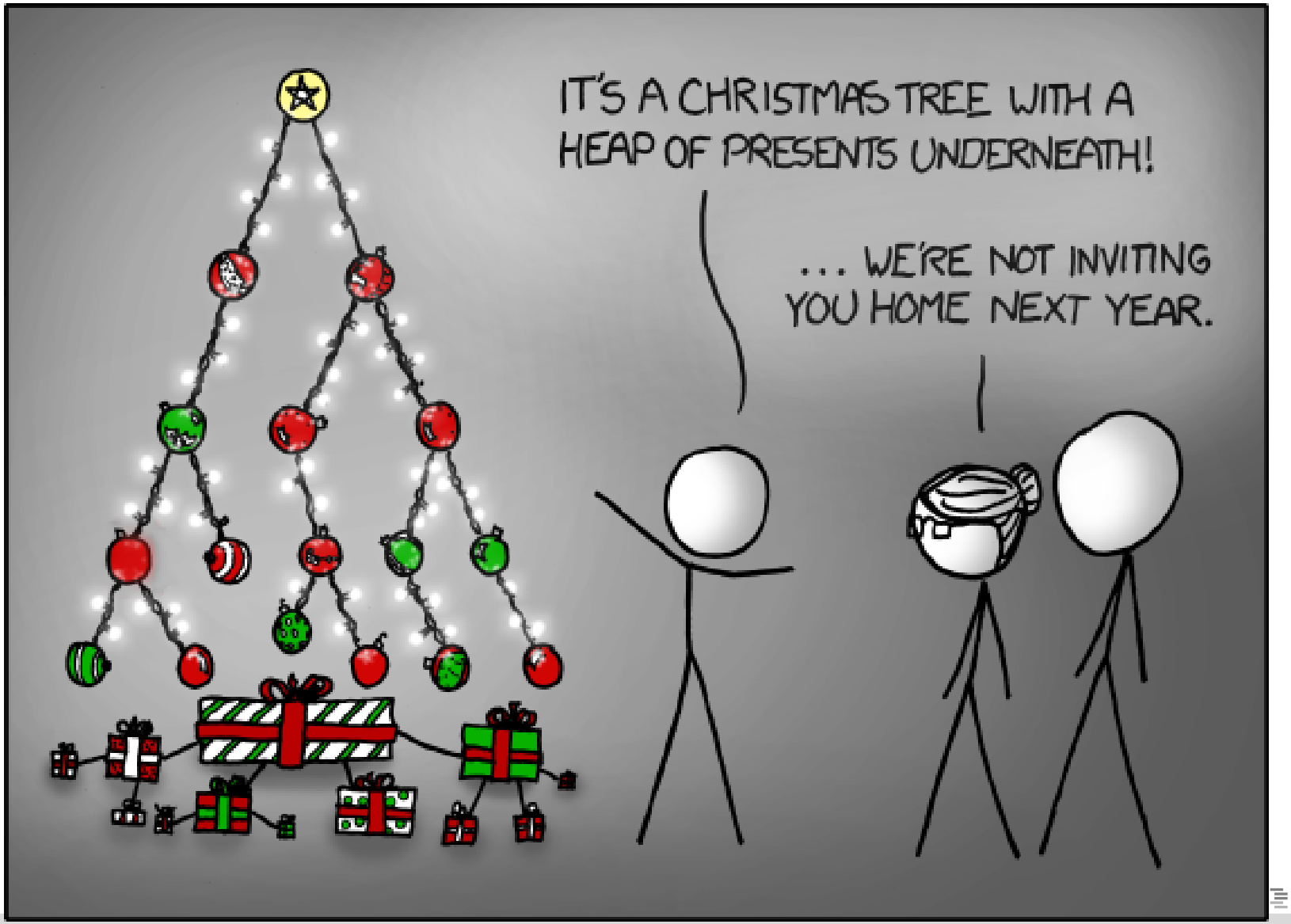
Lösung: Pancake-Sorting

- Größten Pancake nach oben flippen und dann Stapel komplett wenden. Anschließend genauso für die kleineren Panecakes. $\Rightarrow$ Laufzeit: $2n$
- Man kann allerdings nach dem Zweitkleinsten aufhören. $\Rightarrow$ Laufzeit: $2(n-1)$
- Hat man den drittgrößten Pancake erreicht, gibt es für die kleinsten nur 2 Möglichkeiten, d.h. muss man maximal einmal drehen. $\Rightarrow$ Laufzeit: $2(n-1)-1 = 2n-3$

Kreativaufgabe: Spaghetti-Sort

- **gegeben:** Liste mit n Elementen $\in \mathbb{N}$, eine Packung Spaghetti mit mindestens n Spaghetti.
- **gesucht:** Algorithmus zum Sortieren der Liste mit Hilfe der Spaghetti in $\mathcal{O}(n)$.

Tree


IT'S A CHRISTMAS TREE WITH A HEAP OF PRESENTS UNDERNEATH!
... WE'RE NOT INVITING YOU HOME NEXT YEAR.