

Data Structures - Heaps

- We will be developing data structures that support **queries** and **updates**.
- Example where a (priority queue) data structure will help:
 - » Event driven system- simulating phone calls of known length
 - » event: (start of call , time when starts)
(end of call , time when ends)
 - » Processing a new call introduces 2 events - **start** and **end**.
 - » Simulator: pick **next event**, process it, maybe update event queue.
 - » How to maintain events ?
Need support for fast:
 - enter **new event**
 - pick "next event", i.e. event with **smallest time key**.

79

Several possible approaches

- Keep all events in a **list**.
 - » Easy to insert - $O(1)$
 - » hard to extract - $\Omega(n)$ } Explain why !
- **Sorted list**:
 - » Easy to extract - $O(1)$
 - » Hard to insert - $\Omega(n)$ } Explain why !
- We would like something like:
 - » insert $O(\lg n)$
 - » extract $O(\lg n)$ } Tradeoff

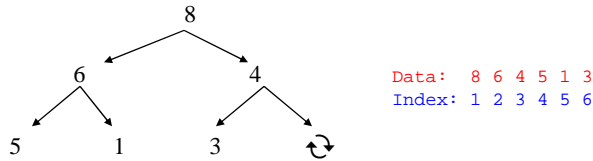
80

Heaps

- Nearly complete binary tree with:

Heap property: $A[\text{parent}(i)] \geq A[i]$

- Claim: max is at the root (by induction on the size of the heap)

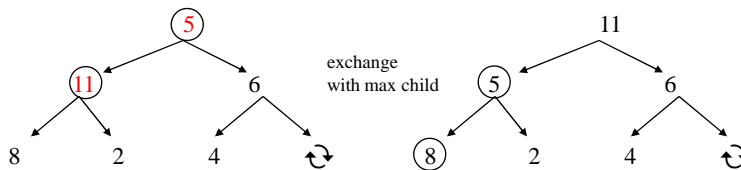


- Pointers are not the most efficient solution.
Instead, $\text{parent}(i)$ is stored in location $\lfloor \frac{i}{2} \rfloor$
Example: parent of the 5th element is at index 2.
- How can we determine that a node has no children?
[Hint: use n , the size of the heap.]

81

Fixing a broken heap

- Assume problem is only at the root:

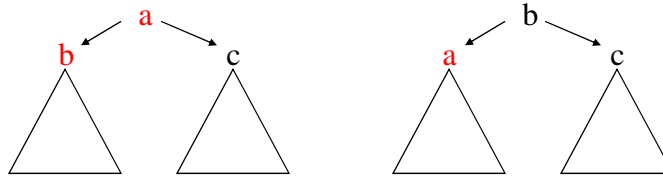


- Now the problem "moved" down, into left tree.
Recursion in this tree, exchanging 5 and 8, its largest child.

82

Correctness of fixing the heap

- b is larger than c , and a , thus the only problem can be between a and one of its children.



- Formal proof - by induction on the height of a .
- This procedure will be called **Heapify**. Makes subtree rooted at a into a heap.
- Time: $O(\lg n)$. (Why ??)

83

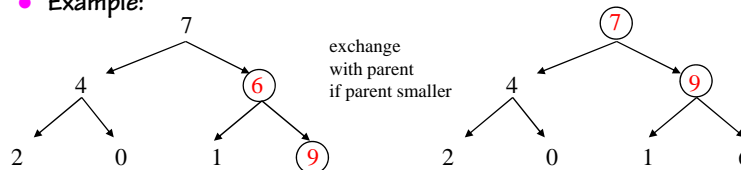
Inserting new element

- Similar to Heapify:


```

last++
A[last]=new element
i=last
while parent(i) != null
  if A(i) < A(parent(i)) return
  else exch. A(i), A(parent(i))
      i=parent(i)
end
end
      
```

- Example:



- Propagate up, $O(\lg n)$. Correctness ??
(Main idea: we start from legal heap, the only problematic element is the new one. Old elements "move down", which maintains heap property)

84

Extract max

- $\left. \begin{array}{l} \text{max}=\text{A}(1) \\ \text{A}(1)=\text{A}(\text{last}) \\ \text{last} -- \\ \text{Heapify}(\text{A},1,\text{last}) \end{array} \right\} O(\lg n)$
- How to build a heap initially?
 - $\left. \begin{array}{l} \text{for } i=n \text{ down to } 1 \\ \text{Heapify}(\quad) \\ \text{end} \end{array} \right\} \begin{array}{l} n \text{ loops, } O(\log n) \text{ each,} \\ \text{Total: } O(n \log n) \end{array}$
 - But "bottom loops" take less time, since height is smaller!
 - Observation: cost of Heapify prop. to the height, i.e. # visited levels.

1st level: height 1, 2^{k-1} nodes,
 2nd level: height 2, 2^{k-2} nodes, etc.

$$\begin{aligned} \text{Total: } & 1 \cdot 2^{k-1} + 2 \cdot 2^{k-2} + 3 \cdot 2^{k-3} + \dots + k \cdot 2^0 \\ & = \sum_{i=1}^k i 2^{k-i} = 2^k \sum_{i=1}^k \frac{i}{2^i} \leq 2^k \sum_{i=1}^{\infty} \frac{i}{2^i} = 2^k \frac{1/2}{(1-1/2)^2} = 2^{k+1} = O(n) \end{aligned}$$

85

Sorting using heaps

- We can first build heap, then **repeat: remove max.**
- In place:


```
BuildHeap
for i=n down to 2
  exch A(1), A(i)
  Heapify(A,1,i-1)
```
- Essentially same as the first approach.

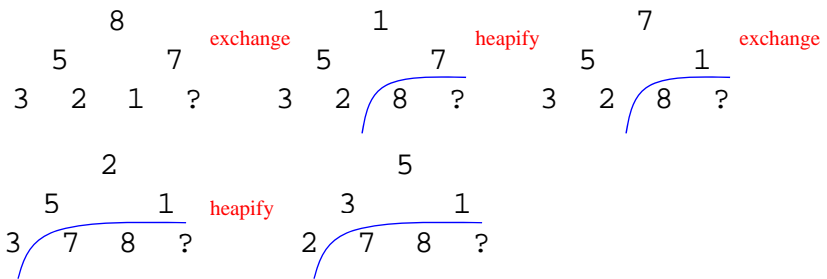
We use the fact that:

 - heap becomes smaller after "remove max",
 - last array entry becomes free.

86

Example

- Sort:



87

Variations on Heaps

- Min instead of max.
- K-ary instead of binary

88