Lecture 6, Thursday 4/19/01

Data Structures - Heaps

- We will be developing data structures that support **queries** and **updates**.
- Example where a data structure will help:
  - Event driven system
  - event: (start of call *i*, time when *i* starts)
    (end of call *j*, time when *j* 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**.

Several possible approaches

- Keep all events in a **list**.
  (What is the problem with using array ??
  the number of events is unknown !)
  - 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)

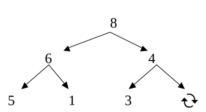
  Tradeoff

Heaps

- Nearly complete binary tree with:

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

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



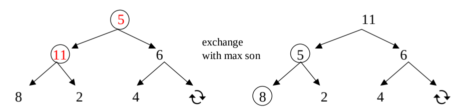

```
Data:  8 6 4 5 1 3
Index: 1 2 3 4 5 6
```

- Pointers are not the most efficient solution.

  Instead, **parent(i)** is stored in $\left\lfloor \frac{i}{2} \right\rfloor$
  Example: parent of the 5th element is at 2.

Fixing a broken heap

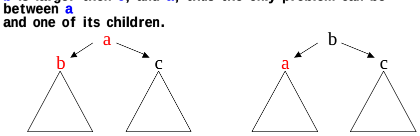
- Assume problem is only at the root:



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

Correctness of fixing the heap

- **b** is larger then **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(A,i,n)***.
  Makes subtree rooted at ***A(i)*** into a heap.
- Time: **O(lg n)**. (Why ??)

Extract max

- ```
  max=A(1)
  A(1)=A(last)
  last --
  Heapify(A,1,last)
  ```
  O(lg n)
- How to build a heap initially ?
  - for i=n down to 1
    Heapify(*A*,*i*,*n*)
    end

    n loops, O(log n) each,
    Total: O(n log n)
  - 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.

    Total: $1\cdot 2^{k-1} + 2\cdot 2^{k-2} + 3\cdot 2^{k-3} + \cdots + k\cdot 2^0$

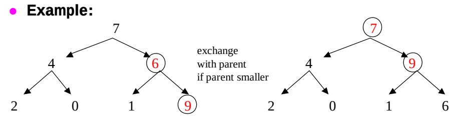
    $$= \sum_{i=1}^{k} i 2^{k-i} = 2^k \sum_{1}^{k} \frac{i}{2^i} \leq 2^k \sum_{1}^{\infty} \frac{i}{2^i} = 2^k \frac{1/2}{(1-1/2)^2} = 2^{k+1} = O(n)$$
  ```

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 ??**