Heaps

Goldsmiths Computing

Motivation

- interesting non-trivial data structure
- asymptotically efficient support for many operations:
  - comparison sort
  - priority queues
- component of efficient algorithms for
  - graph traversal
  - selection of k^{th} largest element

Operations

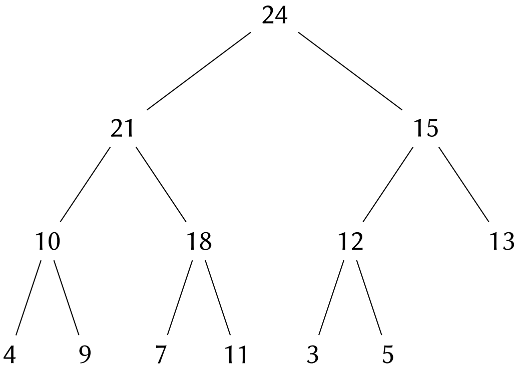
maximum	return the maximum element
extract-max!	remove and return the maximum element
insert![o]	insert the object o into the heap
size	how many elements are currently stored?

Insert

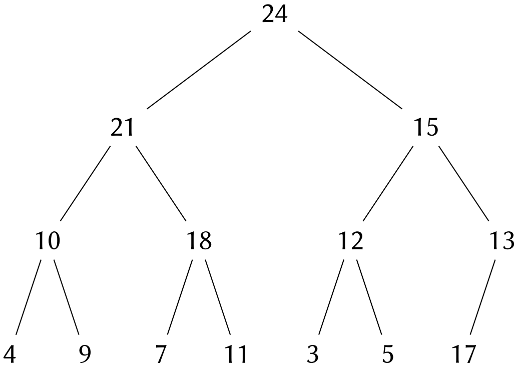
Require: heap :: Heap
  function INSERT!(heap,object)
    s $\leftarrow$ NEXT(heap)
    p $\leftarrow$ PARENT(s)
    while p $\neq$ NIL $\wedge$ p.key $<$ object **do**
      s.key $\leftarrow$ p.key
      s $\leftarrow$ p; p $\leftarrow$ PARENT(p)
    end while
    s.key $\leftarrow$ object
  end function

insert!

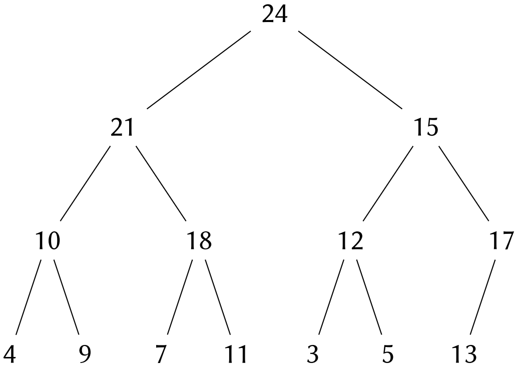
Inserting 17 to:



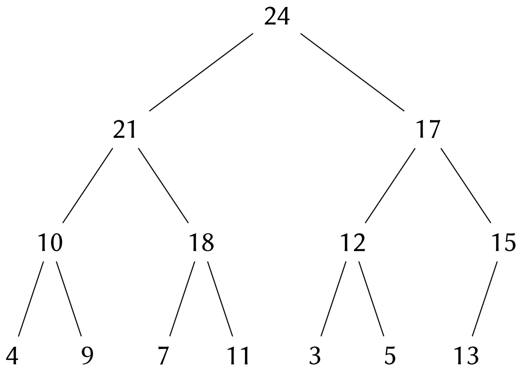
insert!



insert!



insert!



Complexity analysis

insert!

- new element goes at the bottom of the tree
- in principle could be moved up h times, with constant work each time

$$\implies \Theta(h) = \Theta(\log(N))$$

Constructing a heap incrementally

```
function MAKE-HEAP(S)
    H ← new Heap()
    for 0 ≤ i < LENGTH(S) do
        INSERT!(H,S[i])
    end for
    return H
end function
```

Complexity analysis

to build a heap with N elements, incrementally:

- each incremental addition takes $\Omega(h)$ time (h is the *current* height of the tree)
- in the worst case, there are $\frac{N}{2}$ nodes with height $\log(N)$

$$\Longrightarrow \Omega(N \log(N)), \text{ and in fact } \Theta(N \log(N)))$$

Other operations

maximum	trivial
extract-max!	see next term