lecture 4

Heaps

- O(n) algorithm for building a heap
- change key (indexed priority queues)

Background

- I assume you have reviewed what M. Blanchette covered on heaps in COMP 250.
- Sedgewick Algorithms 1 - week 4
https://class.coursera.org/algs4partI-003/lecture/39
- Roughgarden Algorithms 1 - week 5
https://class.coursera.org/algo-004/lecture/62

Resources for today

- see my lecture 30 from COMP 250
(my lectures 28, 29 cover standard COMP 250 heaps)

"Queue" - add an object to end of the list, remove object from front.

"Priority Queue" - remove object with highest priority (defined by a "key")

[A priority queue is an ADT. A heap is a common implementation of a priority queue.]

terminology: possible source of confusion
(will be important later today and in AI)

The term "key" is used in two different ways.

map { (key, value) }
key ↑ name of object
value ↑ object

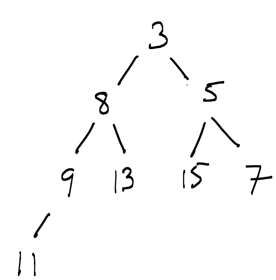
priority queue { (key, object) }
key ↑ priority
object ↑ or name of object

(binary) heap is a common implementation of the priority queue ADT. A heap is a "complete binary tree", such that each node holds a key (and an object).

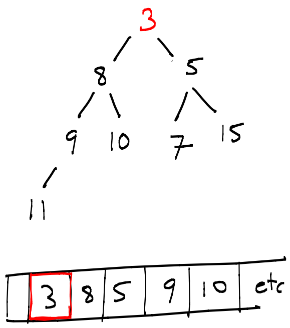
min heap - the key of a parent node is less than the keys of its children. Hence the root holds the smallest key.

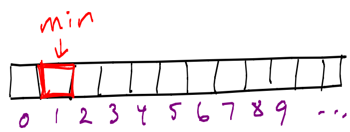
Example:

keys (priorities) are numbers. We ignore associated objects.

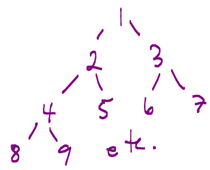


Array-based implementation

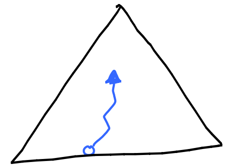




leftchild = 2 * parent
rightchild = 2 * parent + 1

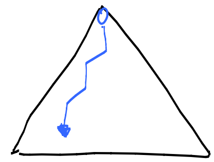


Recall from COMP 250



upHeap(i)

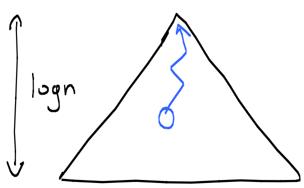
- used by add(key)



downHeap(i)

- used by remove(key)

Both are O(log n).

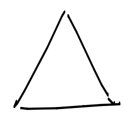


buildHeap {
  for i = 1 to n
    upHeap.(i)
}

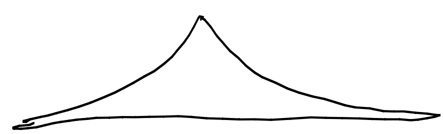
This algorithm take O(n log n) time.

Intuitively obvious (?) since tree has height ~ log n and most elements are near the leaves. But lets formalize this!

Binary trees should not be drawn like this :

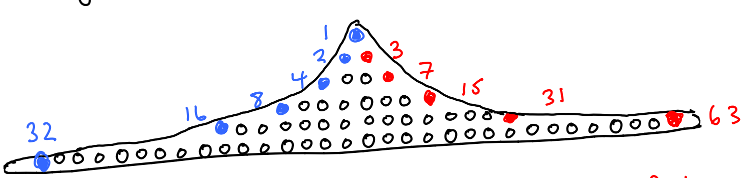


Rather, they should be drawn like this



because level ℓ has 2^ℓ nodes.

Consider a complete binary tree of height h, with all levels full.

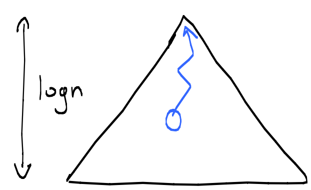


level ℓ has 2^ℓ nodes : $2^\ell, \ldots, 2^{\ell+1} - 1$

number of nodes : $n = 2^{h+1} - 1$

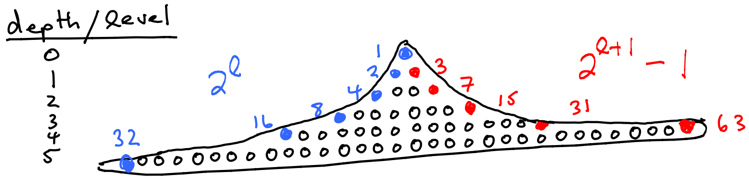
height of tree : $h = \log(n+1) - 1$

buildHeap {
  for i = 1 to n
    upHeap(i)
}



Worst case : each element "bubbles up" all the way to the root (element at depth ℓ is swapped ℓ times).

$\Rightarrow$ total # swaps = sum of depths e.g. tree height $h = 5$



Sum of depths $= \sum_{i=1}^{n} \lfloor \log i \rfloor$ floor

(suppose $n = 2^{\ell+1} - 1$) $\quad = \sum_{\ell=0}^{h} \ell \, 2^\ell$

$$\sum_{\ell=0}^{h} \ell \, 2^\ell = ?$$

Use a trick (calculus)

$$\ell \, x^{\ell-1} = \frac{d}{dx} x^\ell$$

$$\begin{aligned}
\sum_{\ell=0}^{h} \ell \, 2^\ell &= 2 \sum_{\ell=0}^{h} \ell \, 2^{\ell-1} \\
&= 2 \sum_{\ell=0}^{h} \ell \, x^{\ell-1}, \quad x = 2 \\
&= 2 \sum_{\ell=0}^{h} \frac{d}{dx} x^\ell \\
&= 2 \frac{d}{dx} \sum_{\ell=0}^{h} x^\ell \\
&= 2 \frac{d}{dx} \left(\frac{x^{h+1} - 1}{x - 1} \right) \\
&= \ldots \text{ use quotient rule from calculus}
\end{aligned}$$

... and substitute $x = 2$

Check for yourself:

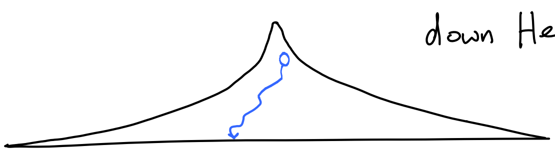
Sum of depths

$$\begin{aligned}
&= \sum_{\ell=0}^{h} \ell \, 2^\ell \\
&= (h-1) \, 2^{h+1} + 2 \\
&= \underbrace{(\log(n+1) - 2)(n+1) + 2}_{\Theta(n \log n)}
\end{aligned}$$

faster way to build a heap

for $i = n/2$ down to 1

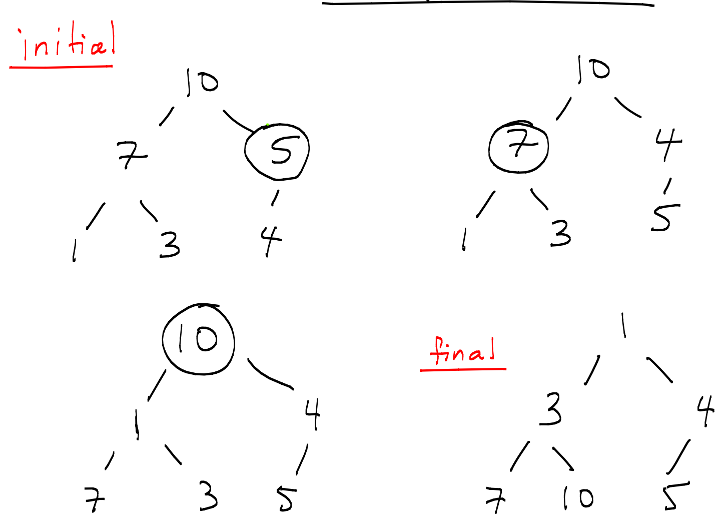
downHeap(i) see my COMP 250 lecture 30



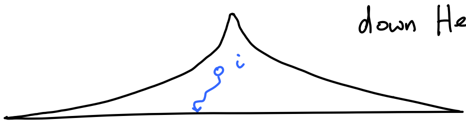
I will show this algorithm is $O(n)$.

Intuition: most nodes are already deep.

Example ($n = 6$)



for $i = n/2$ down to 1
downHeap(n, i)

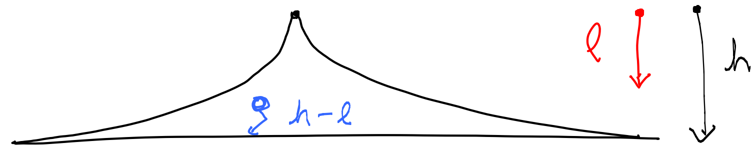


Exercise:

Does it always build a heap?

Claim: buildHeap takes time $O(n)$

Proof:



Suppose the heap has height h.
DownHeap-ing a node at level ℓ requires at most $h-\ell$ swaps
($h-\ell$ is the height of the node.)

worst case for total number of swaps
= sum of node heights

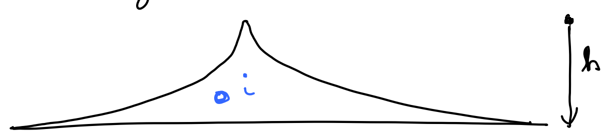
$$= \sum_{\ell=0}^{h} (h-\ell)\, 2^{\ell}$$

$$= h \sum_{\ell=0}^{h} 2^{\ell} - \sum_{\ell=0}^{h} \ell\, 2^{\ell}$$

check for yourself

$$= n - \log(n+1)$$

Summary:



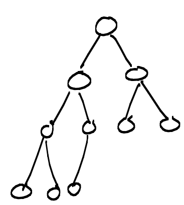
Most nodes are near level h

for $i = 1$ to n upHeap(i)	for $i = n/2$ to 1 downHeap(i)
$O(n \log n)$	$O(n)$

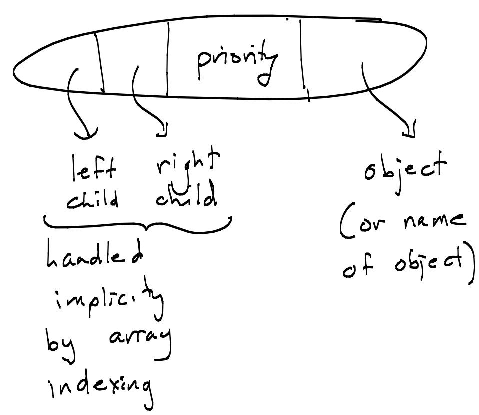
Heaps

- $O(n)$ algorithm for building a heap
- changePriority, indexed priority queues

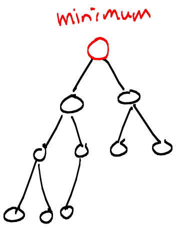
Heap



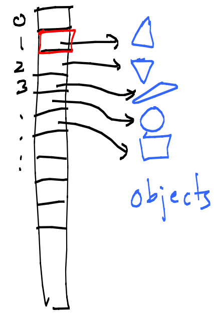
Each node is



class PriorityQueue<E> in Java



presumably implemented using an array-based heap



Objects of class E are ordered according to E.compareTo( )

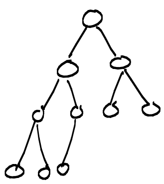
Problem: how to change the priority of some object?

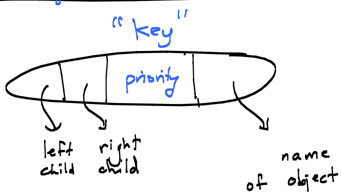
The heaps we have seen (including Java PriorityQueue <E>) do not support changePriority(object, newpriority).

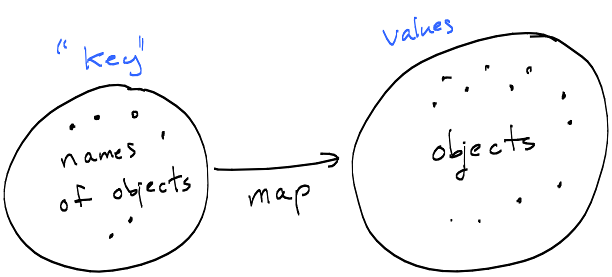
Indeed, find(object) is O(n).

Recall issue mentioned at start of lecture: "key" can mean two things

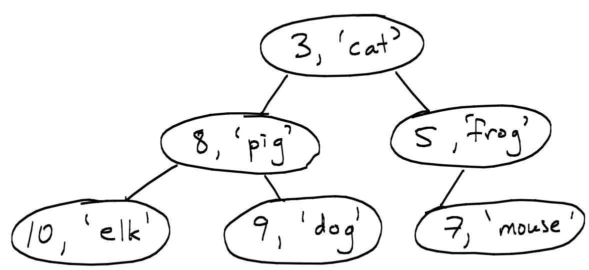
Heap



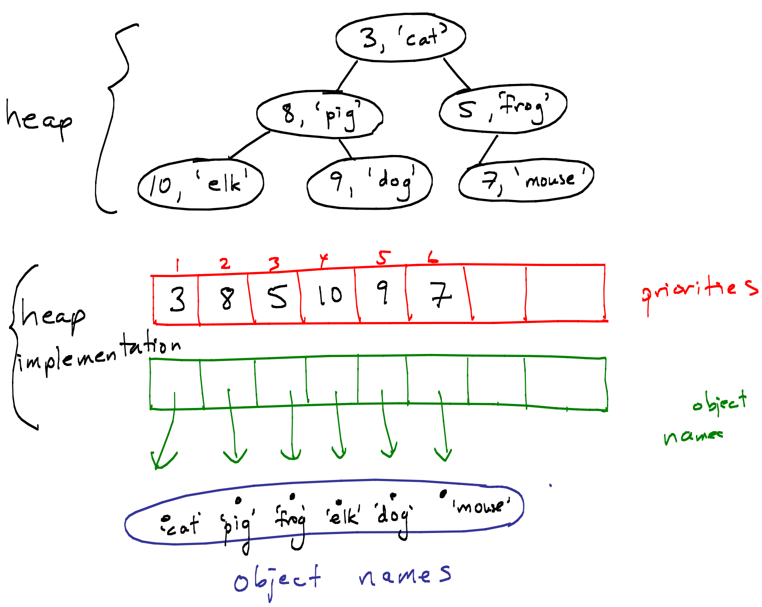




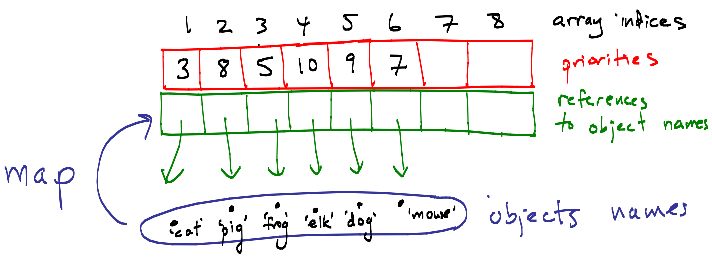
To change the priority of some object, we need to know where that object is in the heap.



e.g. changePriority(4, 'dog')?



What do we need to add to previous slide to have an indexed heap?

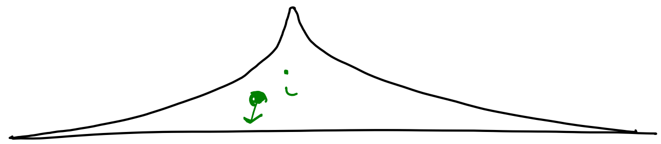


indexToNames : $\{0, \ldots n-1\} \to \{$object names$\}$

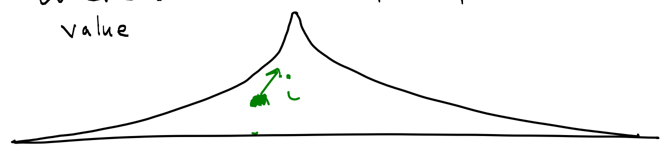
nameToIndex : $\{$object names$\} \to \{$indices $0, \ldots n-1\}$

What if we want to change a priority ?

increase value $\Rightarrow$ downHeap(i)



decrease value $\Rightarrow$ upHeap(i)



Assignment 1 postd today.

due Sunday Jan. 26

(in 10 days) .