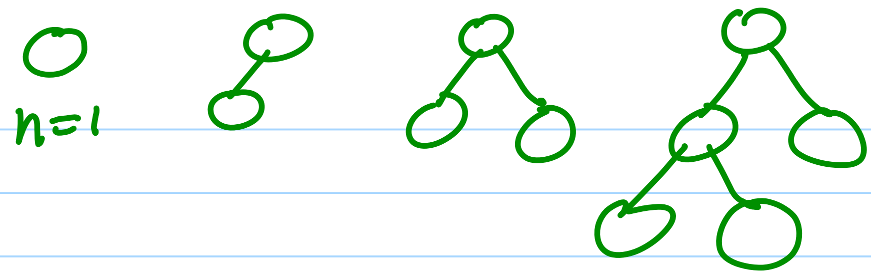
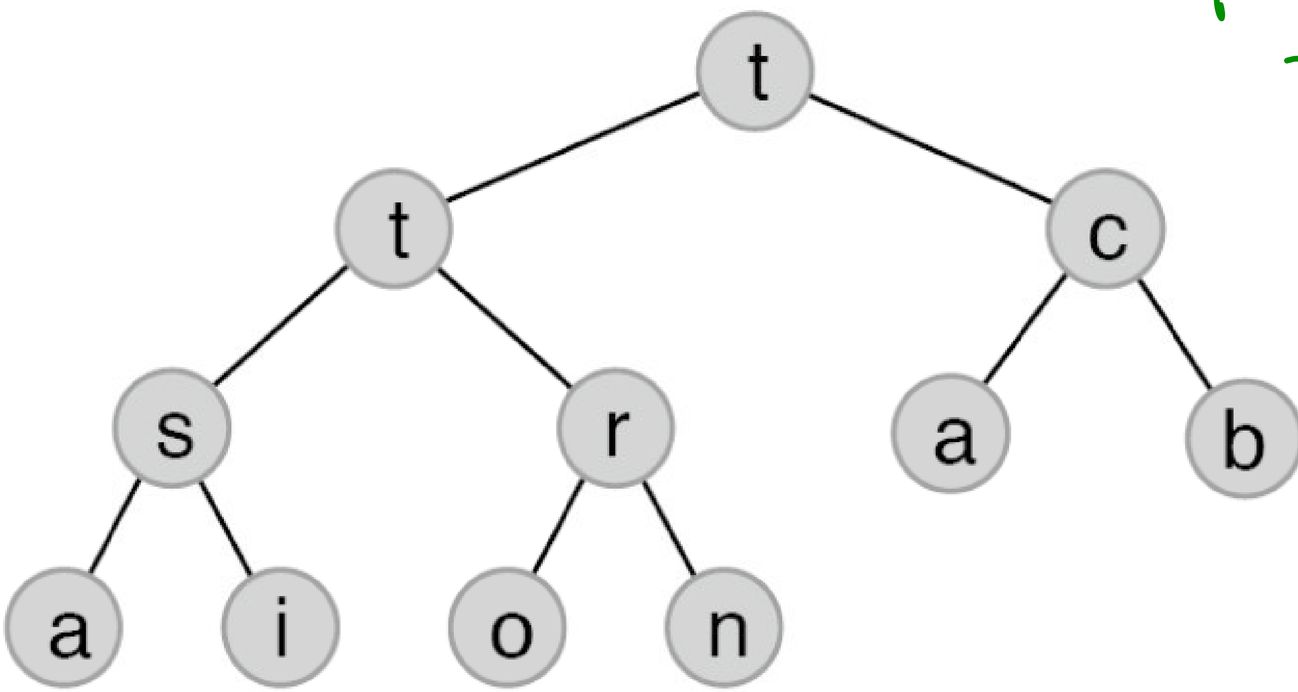
Data Structure
Binary Heap

Structure



Rep.



Property

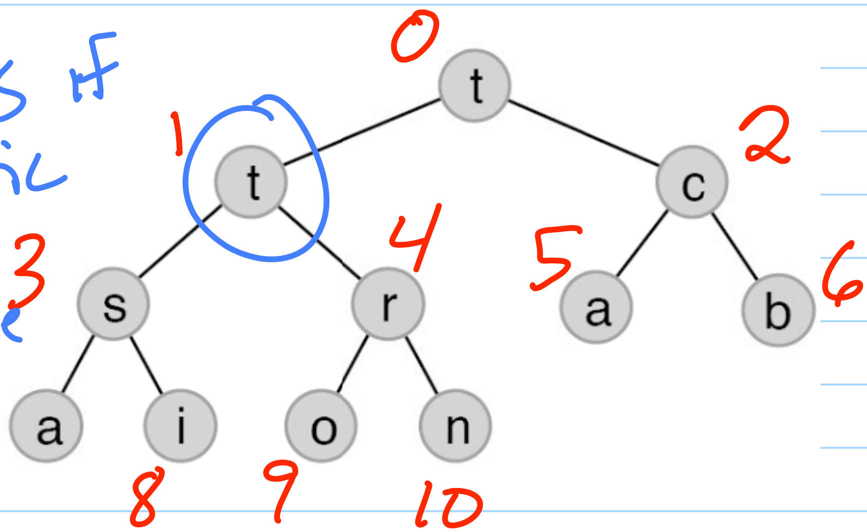
HEAP ORDERED

the priority of each node is as great as that of its descendants

Array Representation

4 refs if static

or otherwise 5



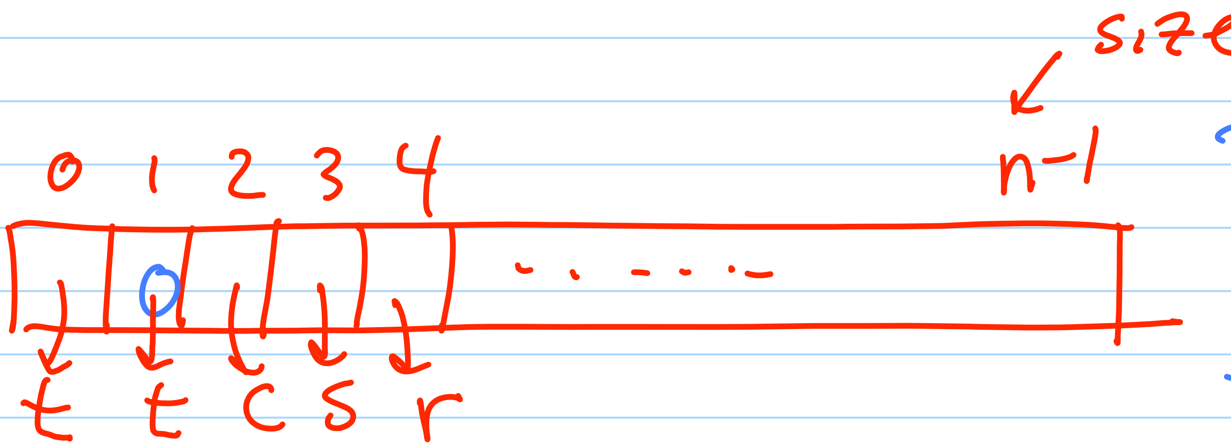
if they exist

$$\text{parent}(i) = \left\lfloor \frac{i-1}{2} \right\rfloor$$

$$\text{left}(i) = 2i+1$$

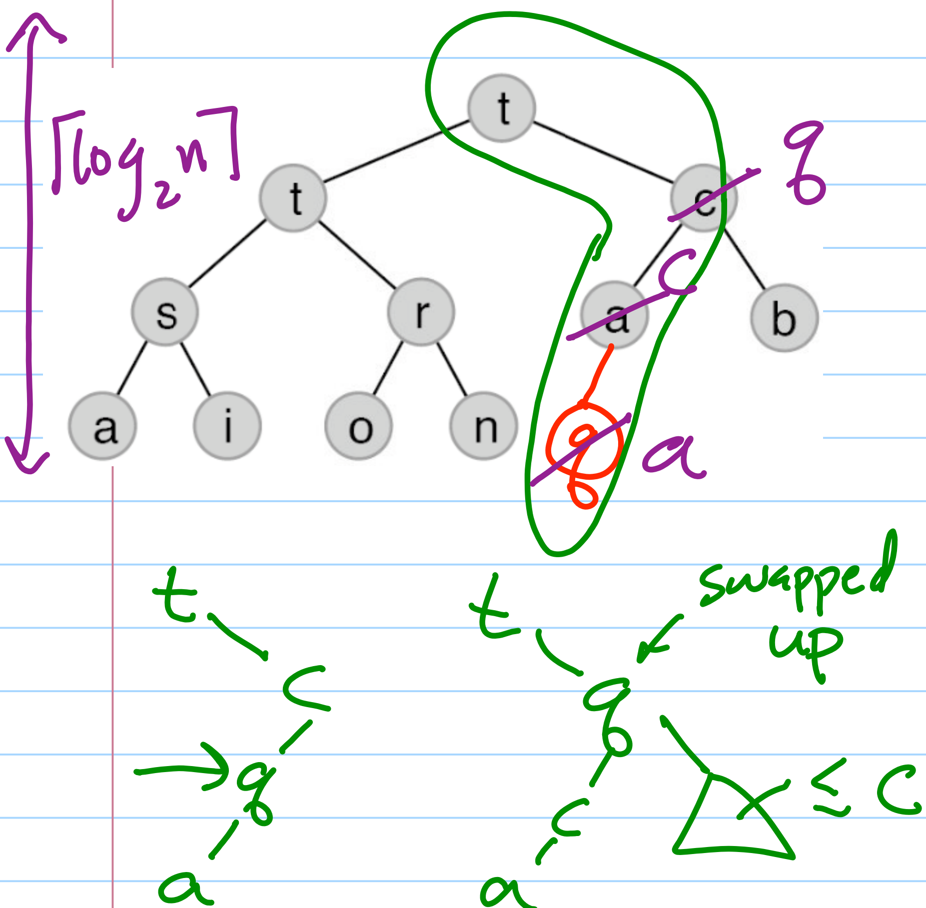
$$\text{right}(i) = 2i+2$$

heap



space between $n + 2n$ references

Inserting an element



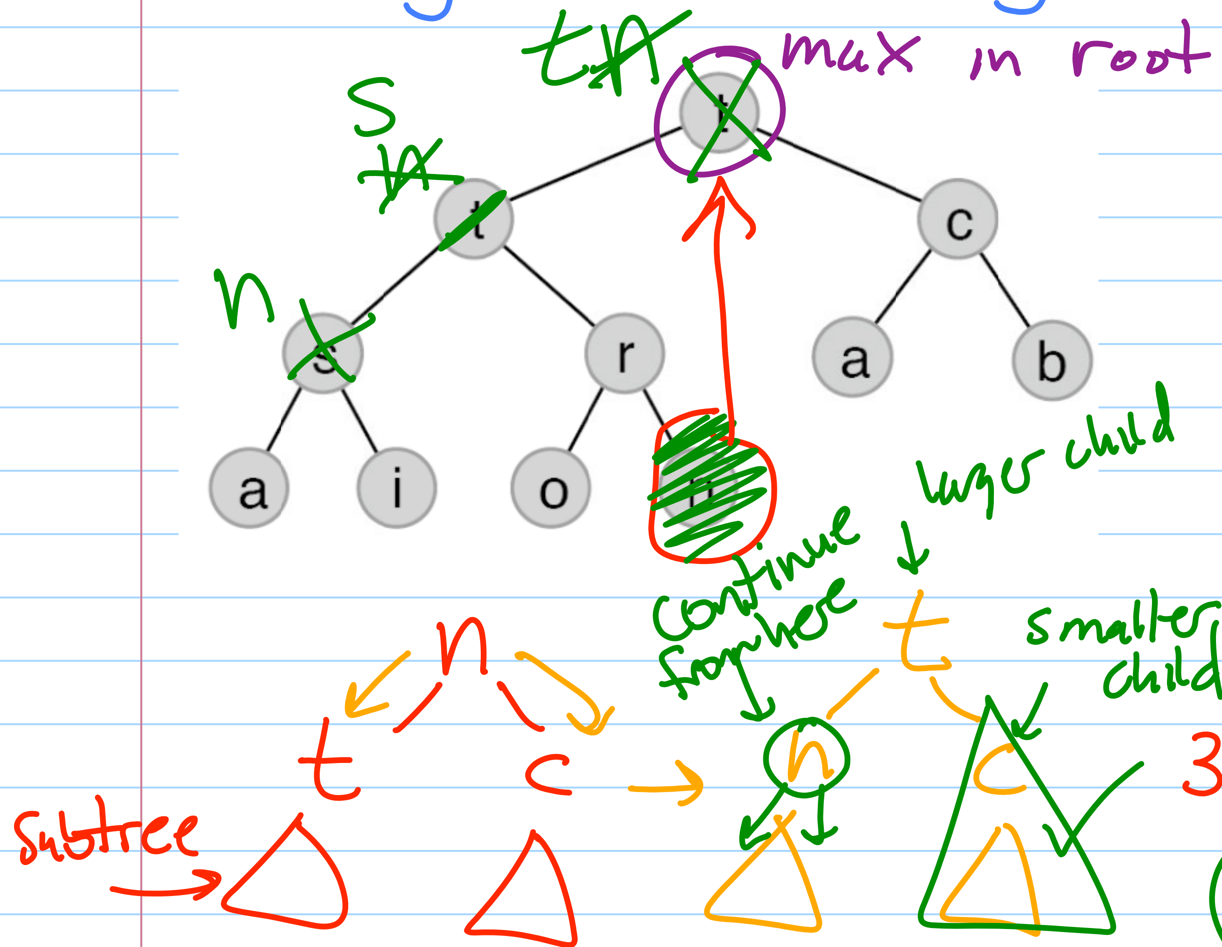
1. add new element to next open slot in the array
2. Swap new element with parent until the parent is at least as large or reach root

$O(\log n)$ time

Finding and Extracting (Removing) Max

$O(\log n)$

max in root (element 0)



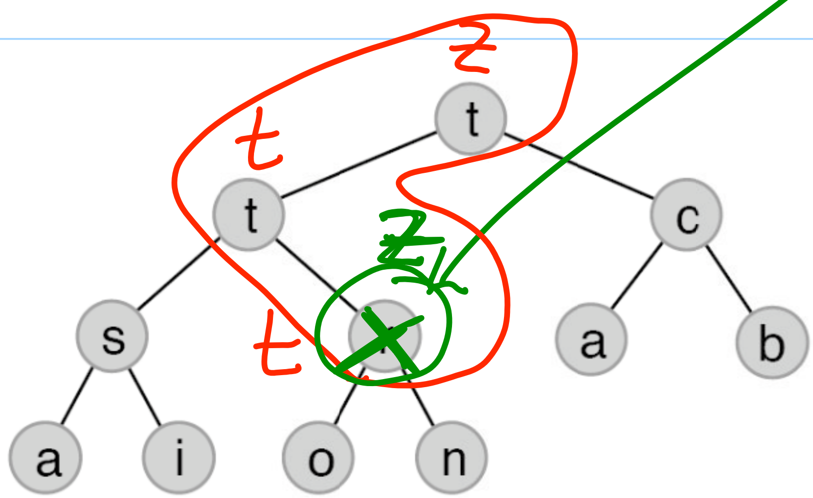
1. remember root (store in a var)
2. replace root by last element

   heap[0] = heap[n-1]

   n--;
3. Look at two children & if HEAPORDERED is violated, swap with larger child

repeat

Increase Priority



tracker t= pq.addTracked (r)

⋮

t.increasePriority (z)

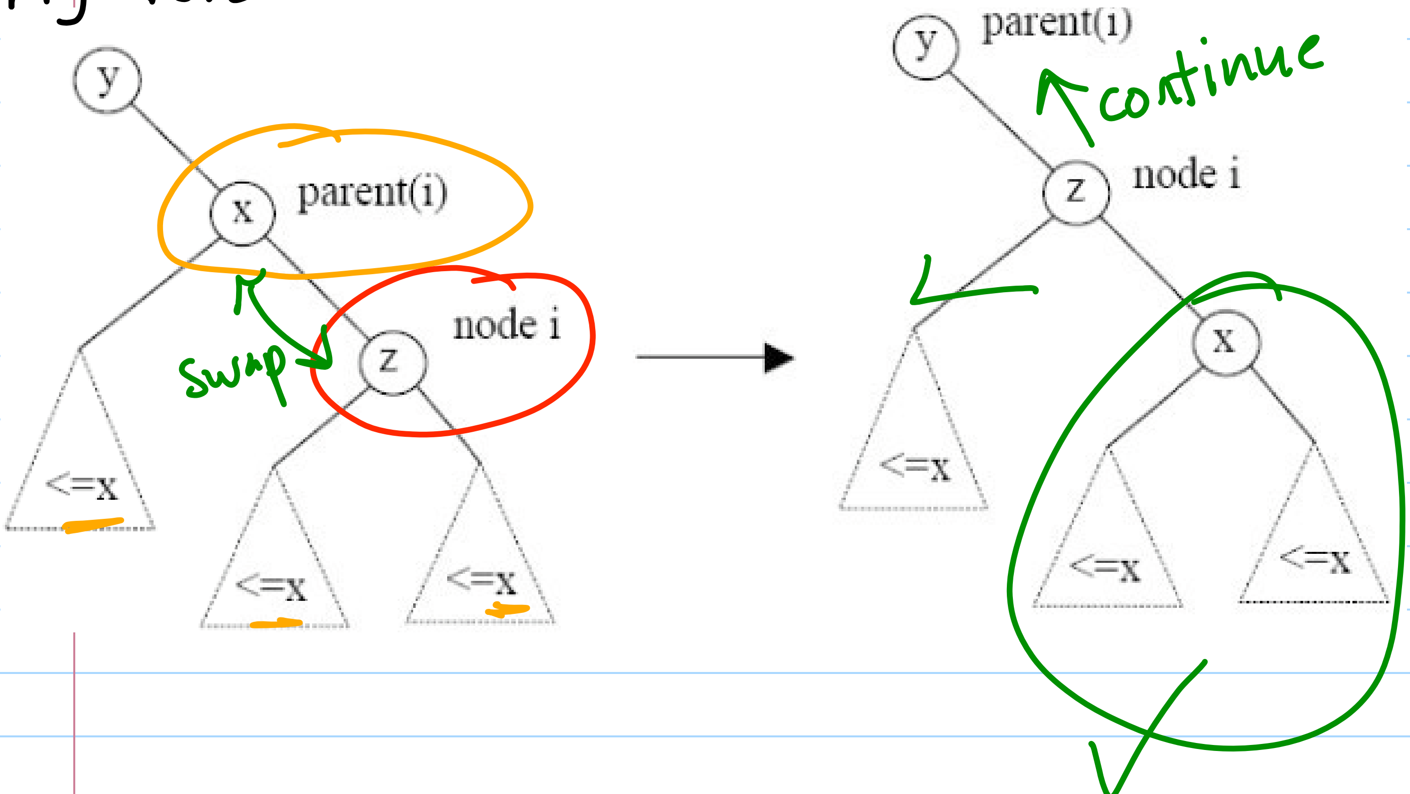
change element

swap with parent until its HEAPORDERED is restored

$O(\log n)$

FixUpward(i)

Fig 25.3

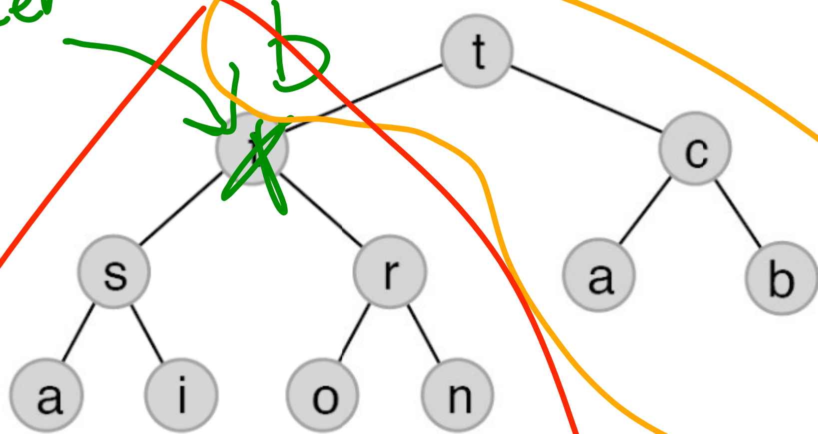


Requires:
Only possible violation of HeapOrdered is between i + its parent

Decrease Priority

tracker
t

b



tracker
t.decreasePriority(b)

fine

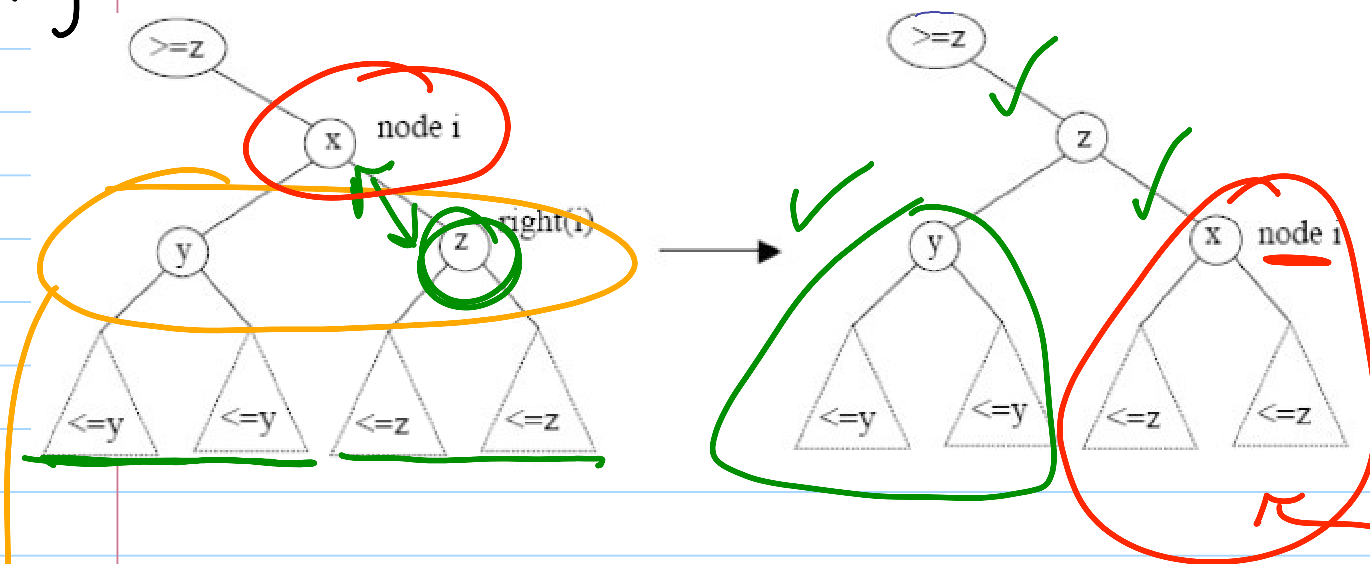
like extractMax on the subtree root at the node being changed

O(log n)

often called heapify

Fix Downward(i)

Fig 25.4



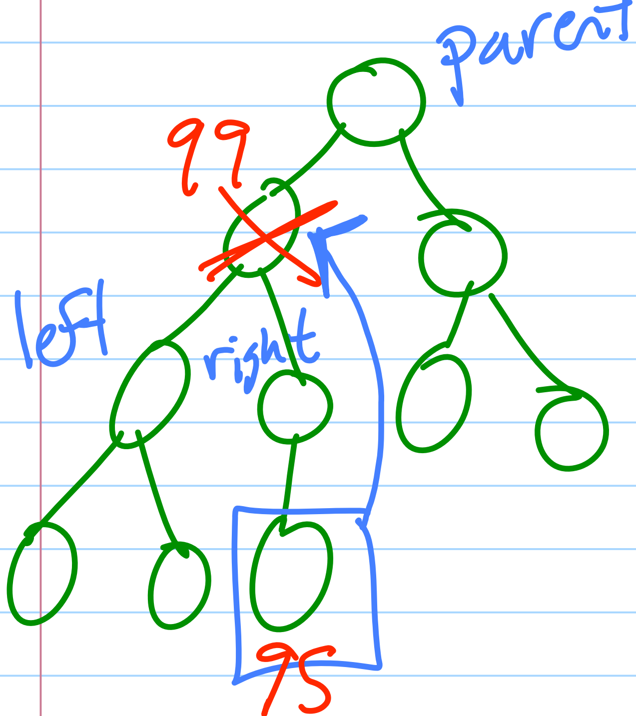
Requires:
Only violation of HeapOrdered between node i + its children

continue here

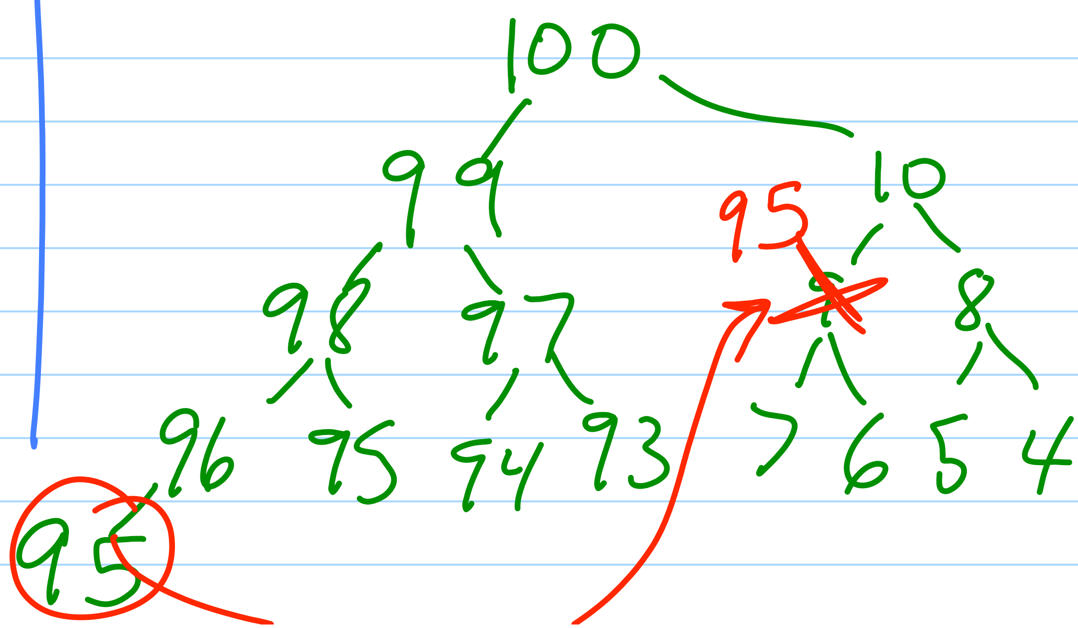
possible violations

select larger child + swap i with that child heap 100, 99, 10, 98, 97, 9, 8, 96, 95, 94, 93, 7, 6....

Deletion



Ex where ~~heap[n-1]~~ replacement is larger than what it replaces



if replacement is larger
fix upward
otherwise fix Downward

Overview of binary heap

Advantage - very space efficient
very simple with low constants
hidden in asymptotic notation

Drawbacks

merging two binary heaps takes
linear time

increasing priority (through tracker) takes
logarithmic time

Merging two priority queues.

Give an arbitrary array $a[0] \ldots a[n-1]$.

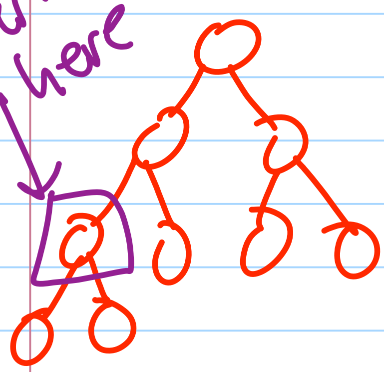
Convert to a binary heap in linear time

could optimize to start at first non leaf

For (int $i = n-1$; $i \geq 0$; i--)

FixDownward(i)

start here



$O(n)$

Successive inserts $O(n \log n)$