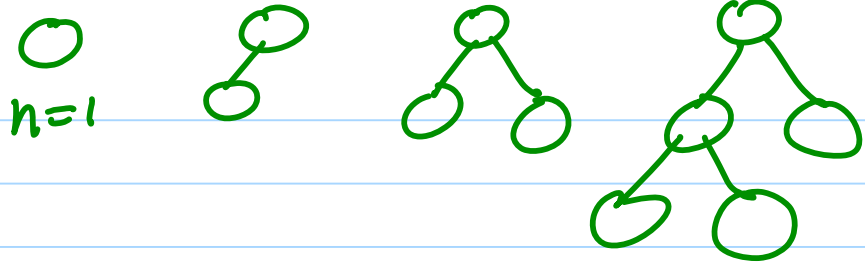


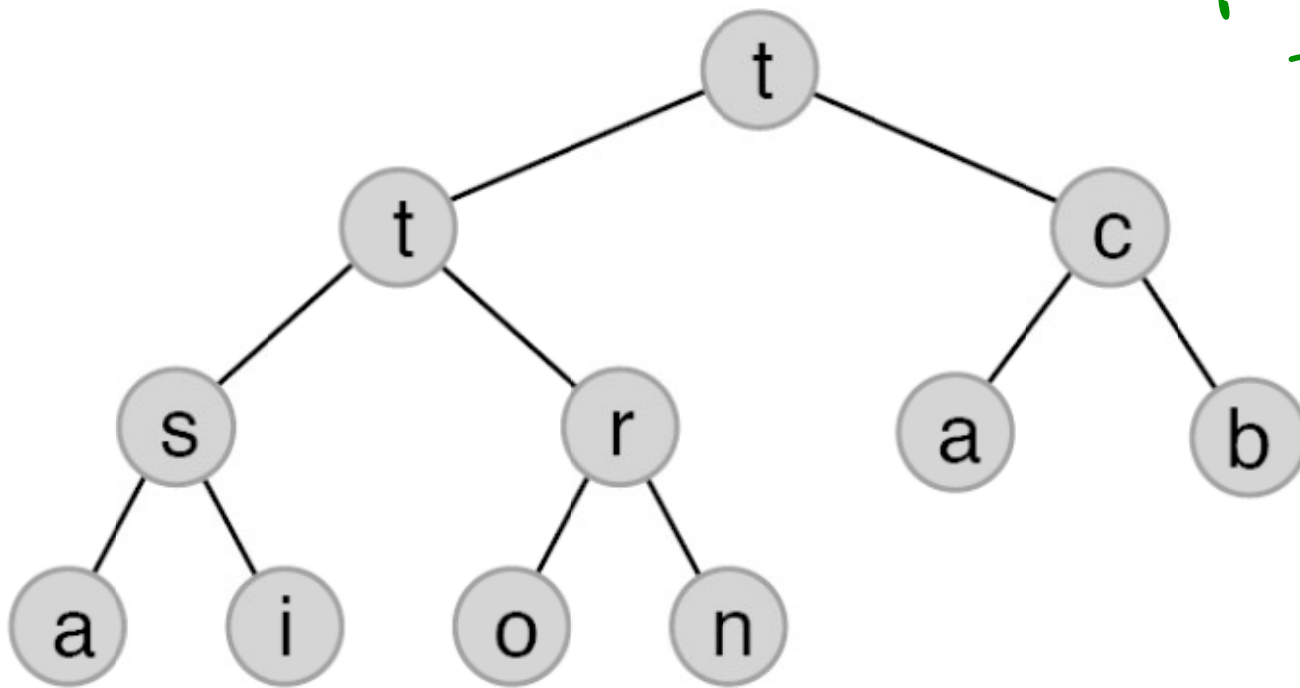


# Data Structure Binary Heap

Structure



Rep. Property

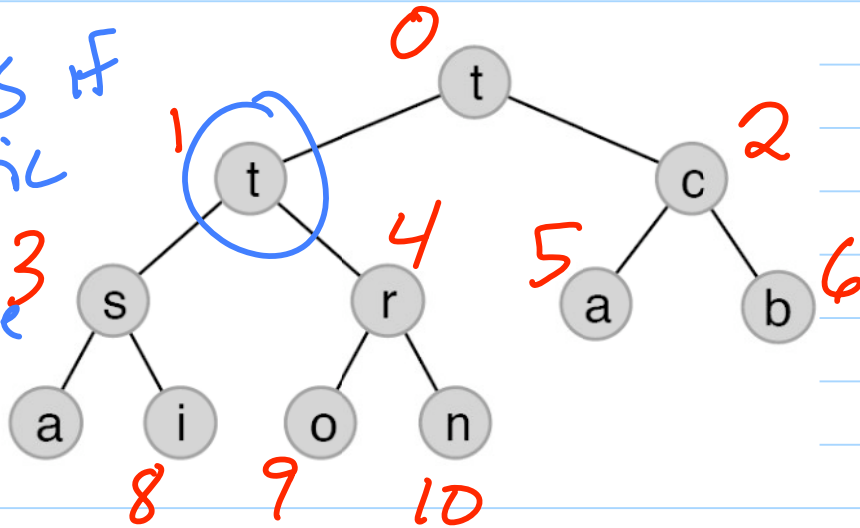


HEAP ORDERED  
the priority  
of each node  
is as great  
as that of  
its descendants

# Array Representation

4 refs if static

or otherwise



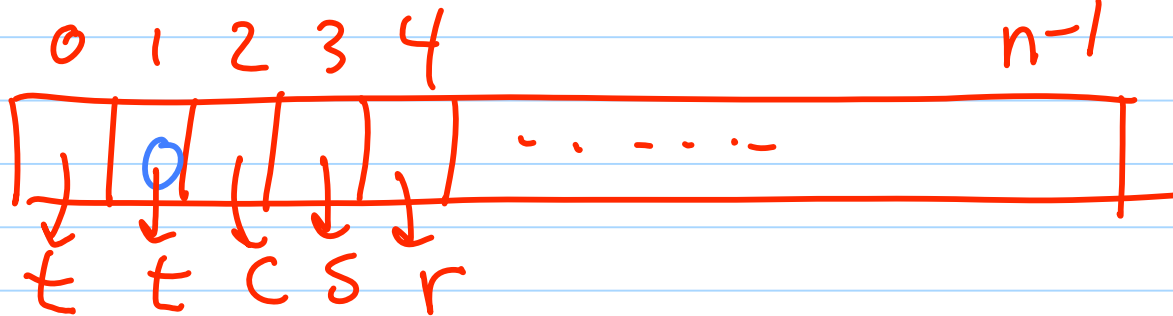
if they exist

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

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

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

heap

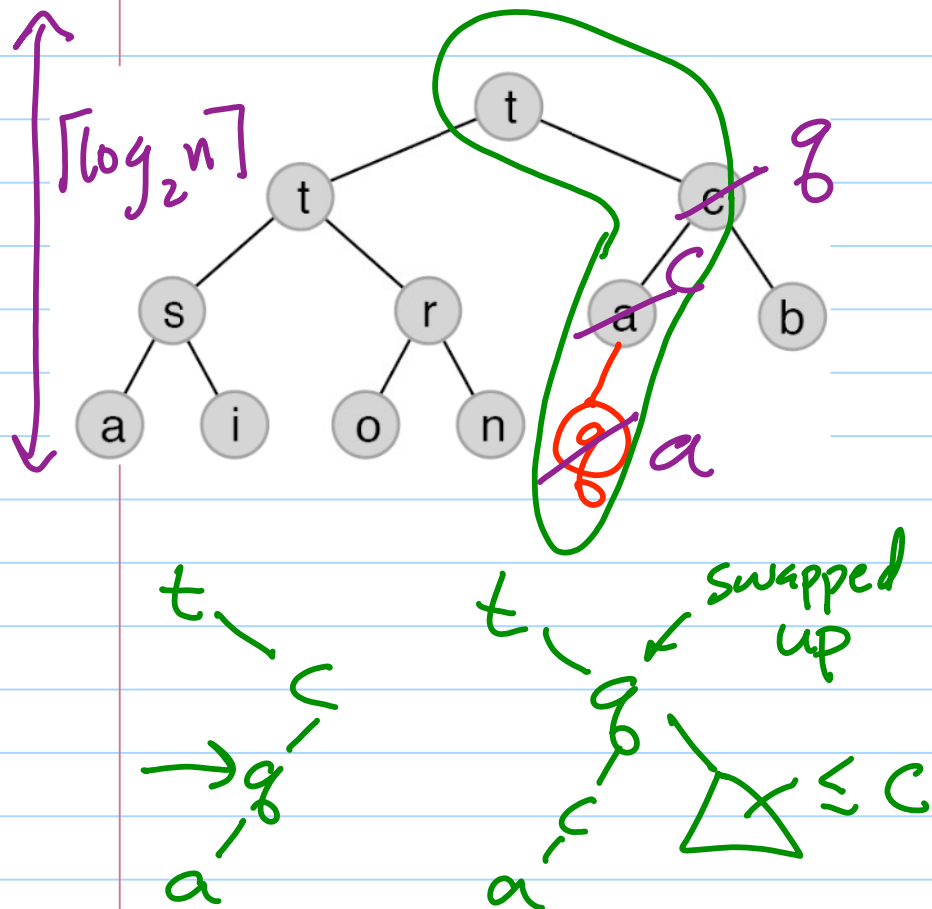


size

n-1

Space between n + 2n references

# Inserting an element

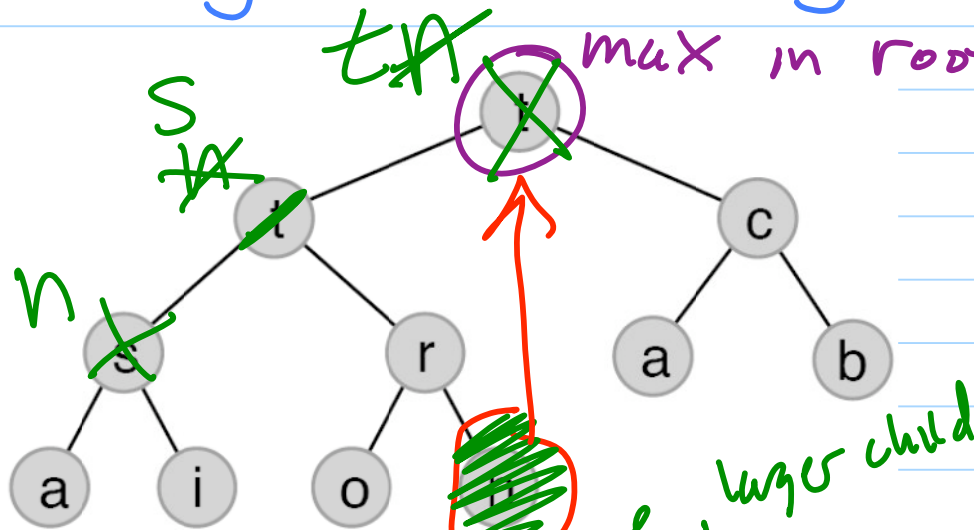


1. add new element to next open slot in the array

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)

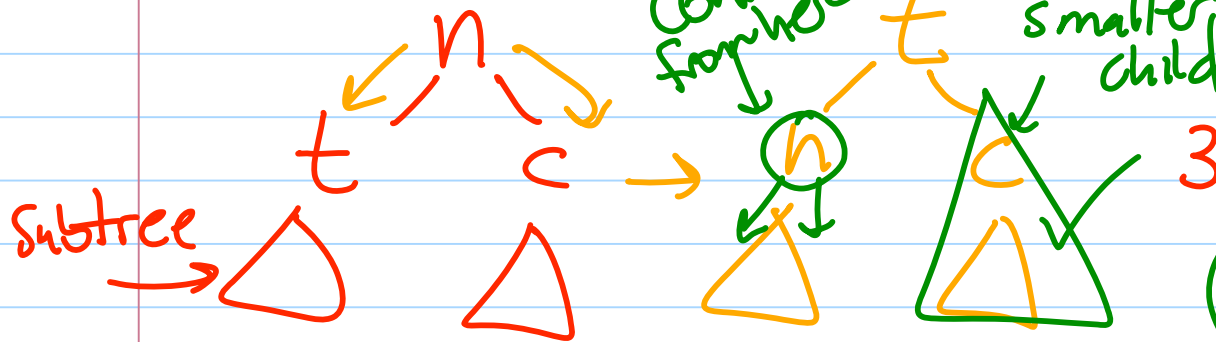
1. remember root (store in a var)

2. replace root by last element

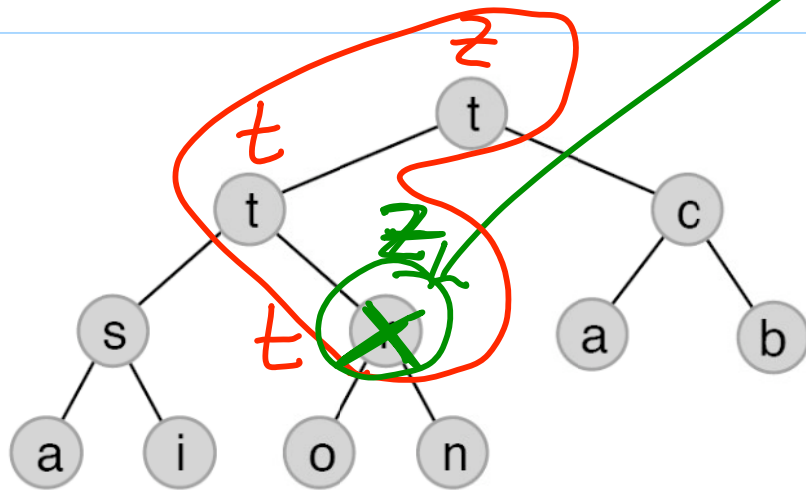
heap[0] = heap[n-1]  
n--;

3. Look at two children

if HEAPOORDERED is violated, swap with larger child  
repeat



# Increase Priority



Tracker t = pg.addTracked(r)

⋮

t.increasePriority(z)

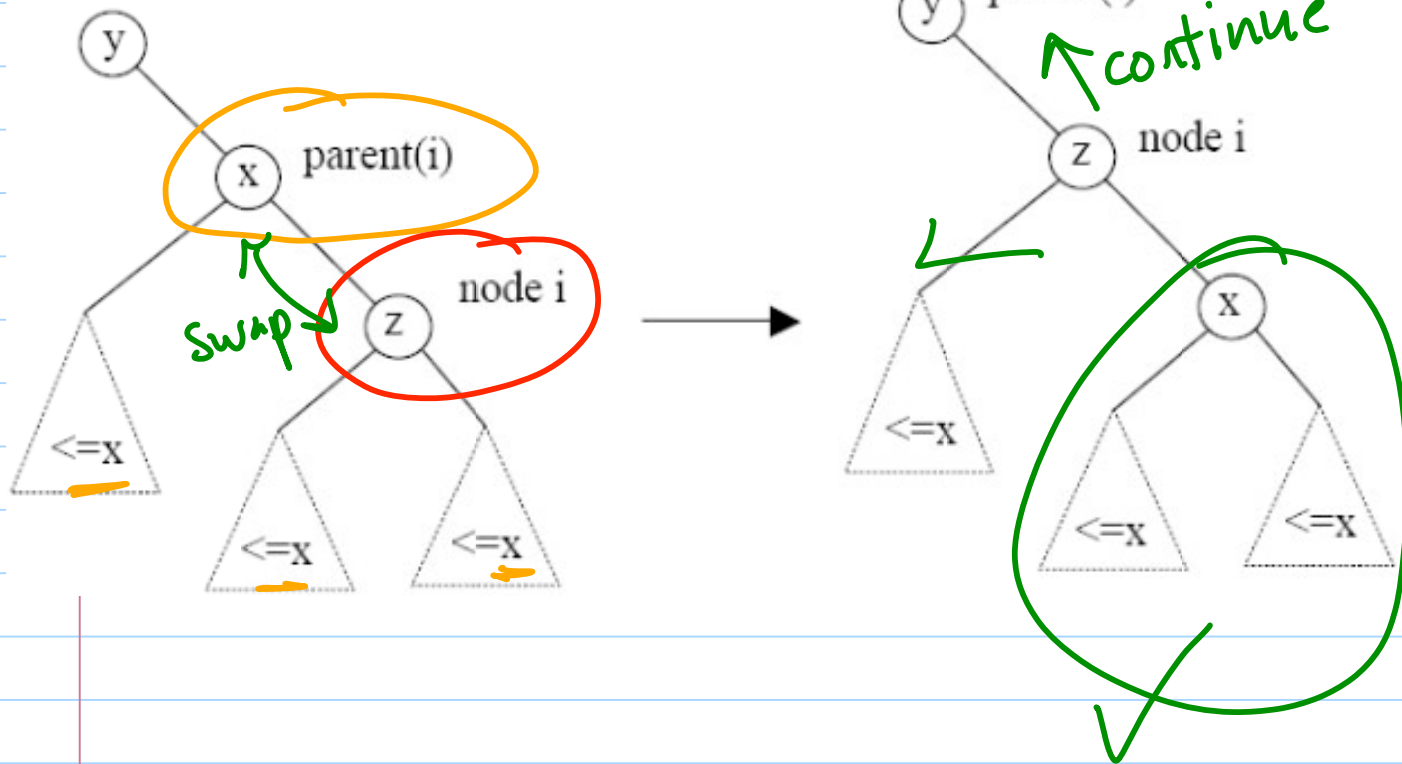
change element

swap with parent  
until its HEAPOORDERED  
is restored

$O(\log n)$

# Fix Upward (i)

Fig 25.3

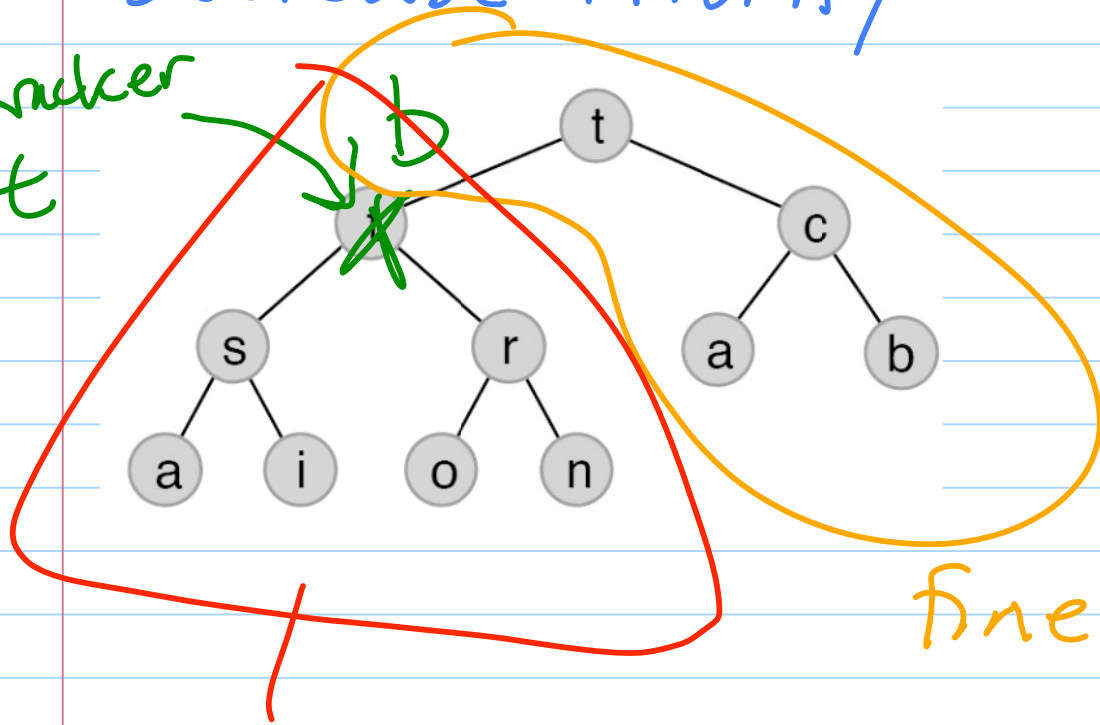


Requires:

Only possible violation of Heap Ordered is between  $i$  + its parent

# Decrease Priority

tracker  
t



tracker  
t.decreasePriority(b)

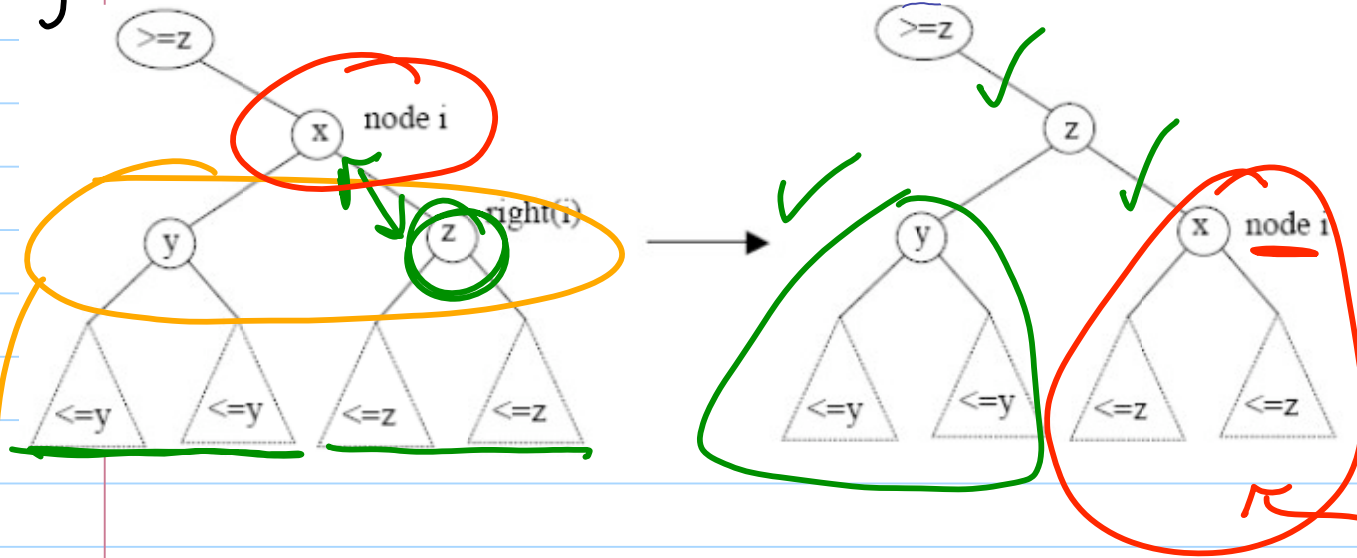
like extractMax on the subtree root  
at the node being changed

$O(\log n)$

often called heapify

Fix Downward( $i$ )

Fig 25.4



Requires:

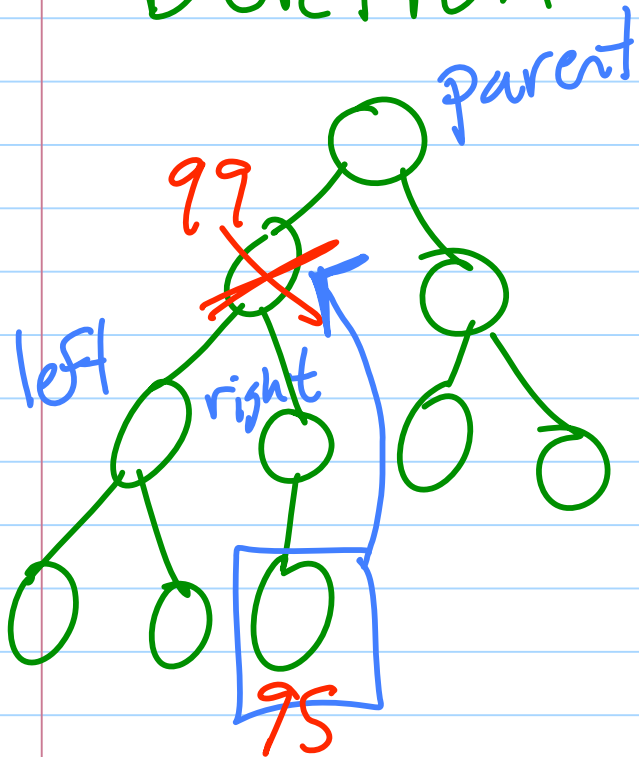
Only violation of Heap Ordered between node  $i$  + its children

Continue here

possible violations

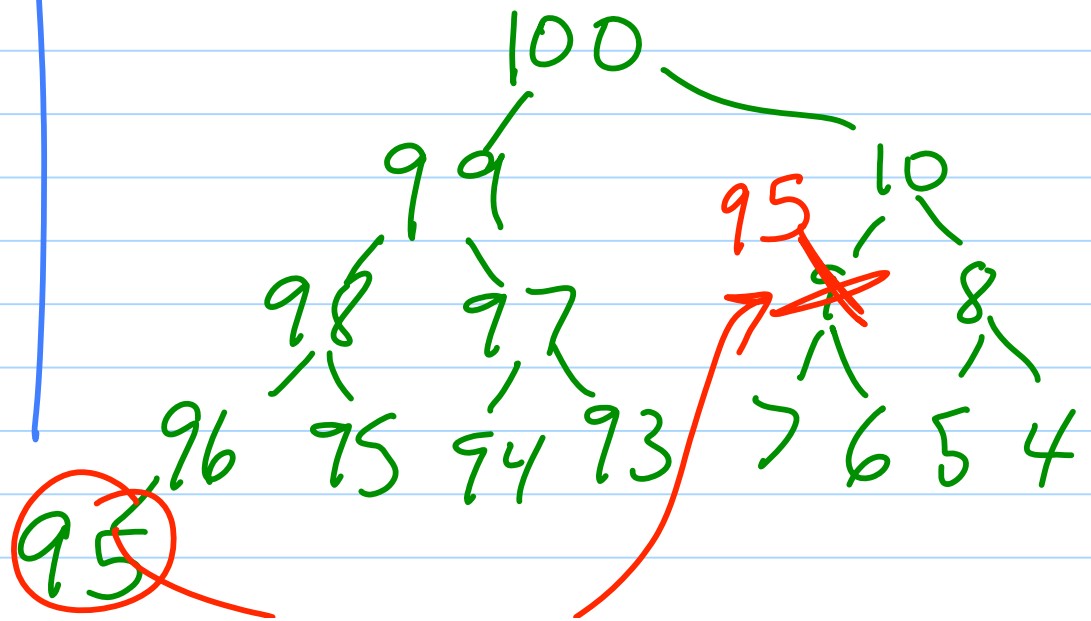
select larger child + swap  $i$  with that child

# Deletion



heap 100, 99, 10, 98, 97, 9, 8, 96, 95, 94, 93, 7, 6, ...

Ex where replacement heap[n-1] 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.

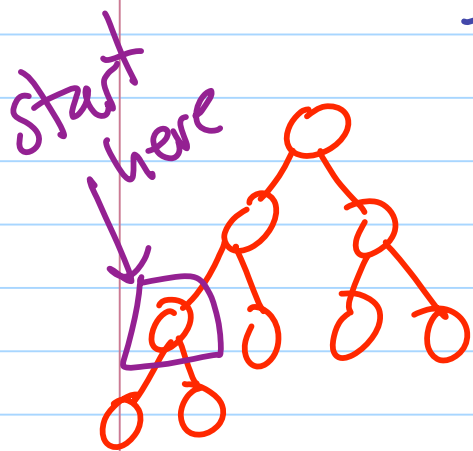
Give an arbitrary array  $a[0] \dots a[n-1]$ .

Convert to a binary heap in linear time

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

FixDownward( $i$ )

could optimize to start at first non leaf



$O(n)$

Successive inserts  $O(n \log n)$

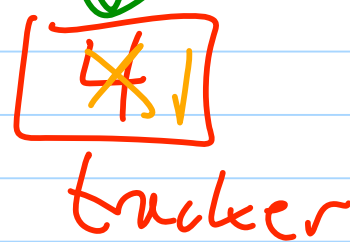
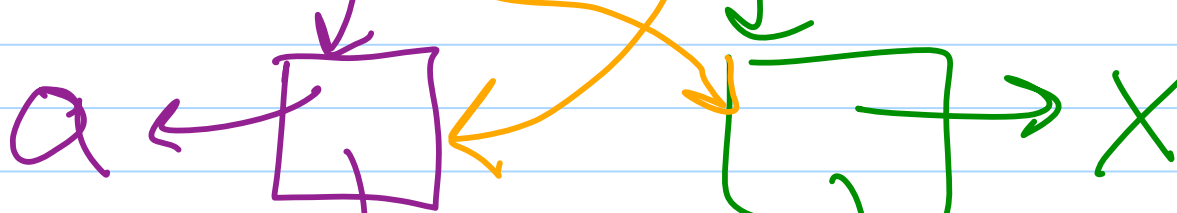
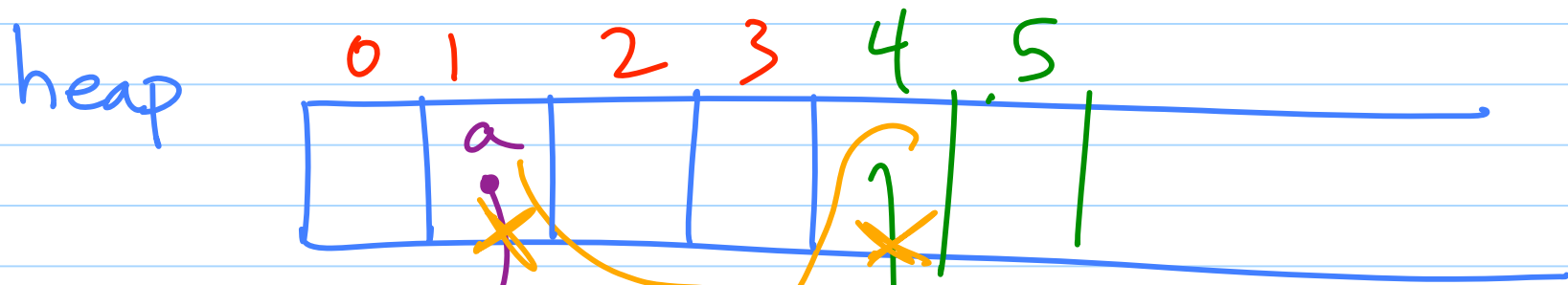
# Overview of Priority Queue Data Structures

	Binary Heap	Leftist Heap	Pairing Heap	Fibonacci Heap	
Contains	●	●	●	●	<u>key</u>
extractMax	○	○	○	○	○
max	○	○	○	○	constant
add	○	○	○	○ ☆	
merge	●	○	○	○ ☆	○
remove	○	○	○	○	logarithmic
increase priority	○	○	○	○ ☆	●
decrease priority	○	○	○	○	linear

through tracker
← amortized →

# Tracked Binary Heap

Swap  
 $a \leftrightarrow X$



application  
 $t_a$