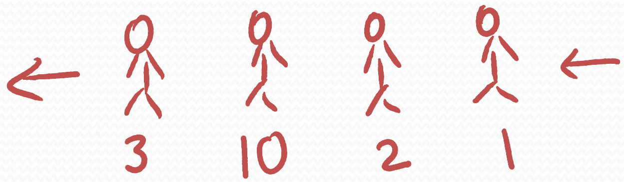
15-121
Introduction to data structures
- Binary Heaps

Ananda Gunawardena

Data Structures so far..

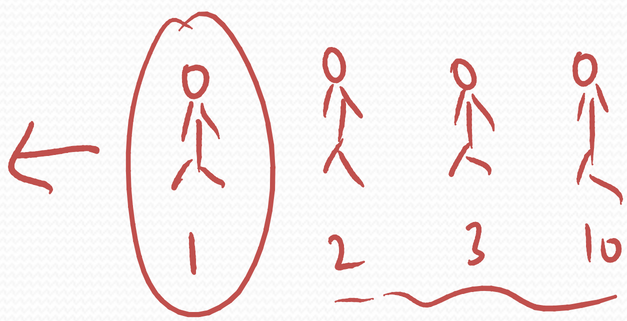
- Arrays and ArrayLists(java only)
- Linked Lists – singly, doubly, circular, multi
- Stacks and Queues
- Binary Search Trees (AVL) Find max $= O(\log n)$
- Hash tables Find max $= O(n)$
- Now to priority queues....



Ave Service time $= \dfrac{3+(3+10)+(3+10+2)+16}{4}$

$= 11.75$ min

A Data structure that is organized based on what is important "now"



Ave Service time $= \dfrac{1+3+6+16}{4}$

$= 6.5$ min

Grocery Store Example

- Suppose there are 5 people in line and each one requires a service time (in mins) 10, 4, 5, 6, 12
- What is the average service time per customer?

$$\text{Ave} = \frac{10 + 14 + 19 + 25 + 37}{5} = \boxed{\quad}$$

- Suppose we decided to service smaller times first.
- What is the average time now?

$$\text{Ave} = \frac{4 + 9 + 15 + 25 + 37}{5} = \boxed{\checkmark}$$

Hence we can make things efficient by dynamically reorganizing things

We need a data structure that can support that

What if we keep things in a random array and always serve the next min/max?

$O(n)$

What is we use a sorted array?

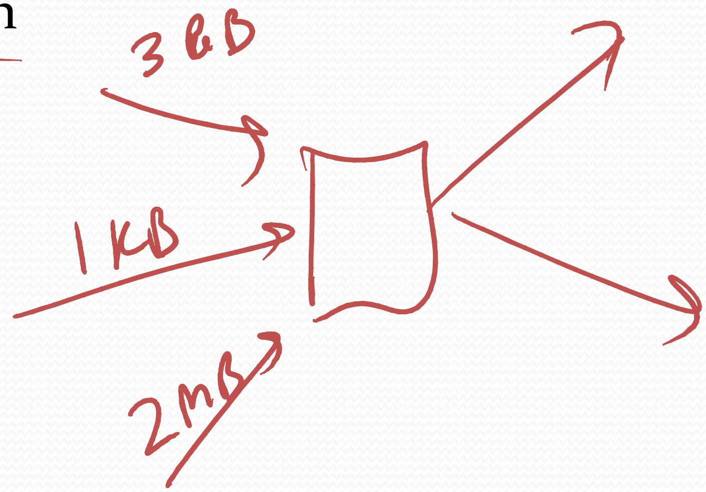
$O(1)$

$O(n \log n)$

Priority Queue
(The Binary Heap)

Priority Queue (or Heap) Data Structure

- A priority queue is a container(data structure) that supports the operations
  - insert(item, priority) — $O(\log n)$?
  - removeMin(). — $O(\log n)$
  - FindMin() — $O(1)$
  - decreaseKey(PQpointer,newPriority)
- There are many applications of Priority queues.
  - Data Compression
  - Printer queues
  - Data routing



Implementing a PQ

- How do we implement a PQ?
- Need a data structure that can support the following operations well
  - insertMin/Max, findMin/max, removeMin/Max

- Possible data structures
  - Unordered list?
    - What is the insertion complexity? Removal complexity?

      FindMin

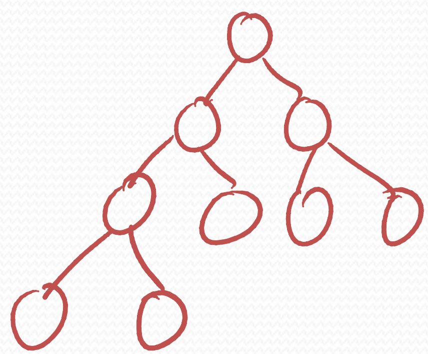
      O(1) O(n)
  - Sorted List?

    O(n) O(1)

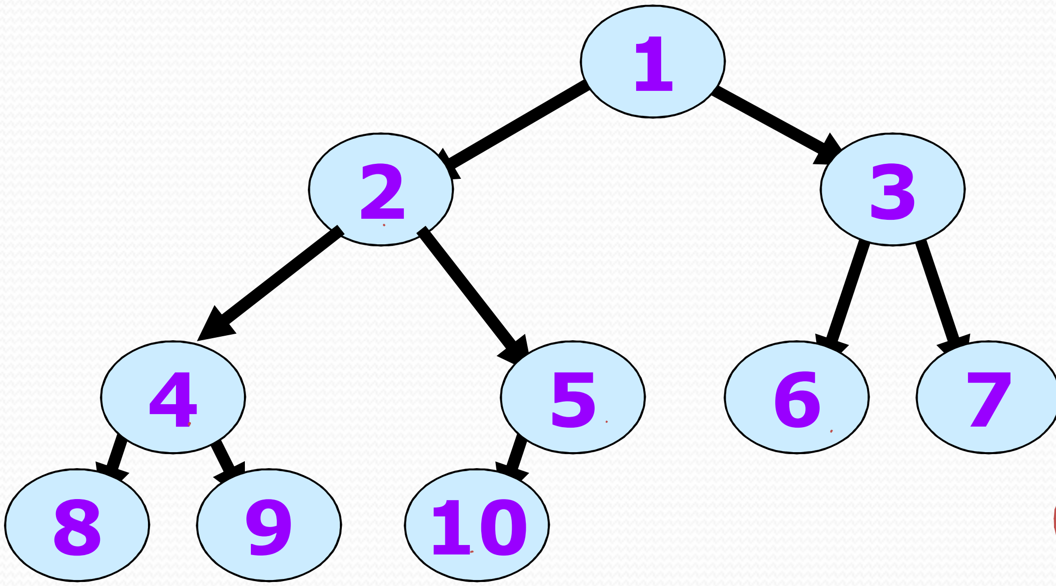
    AVL
  - What about a binary search tree?

    O(log n) O(log n)

Complete Binary Trees
(implementing PQ's)



Recall - Complete binary trees



Given Child
Can parent be
uniquely determined

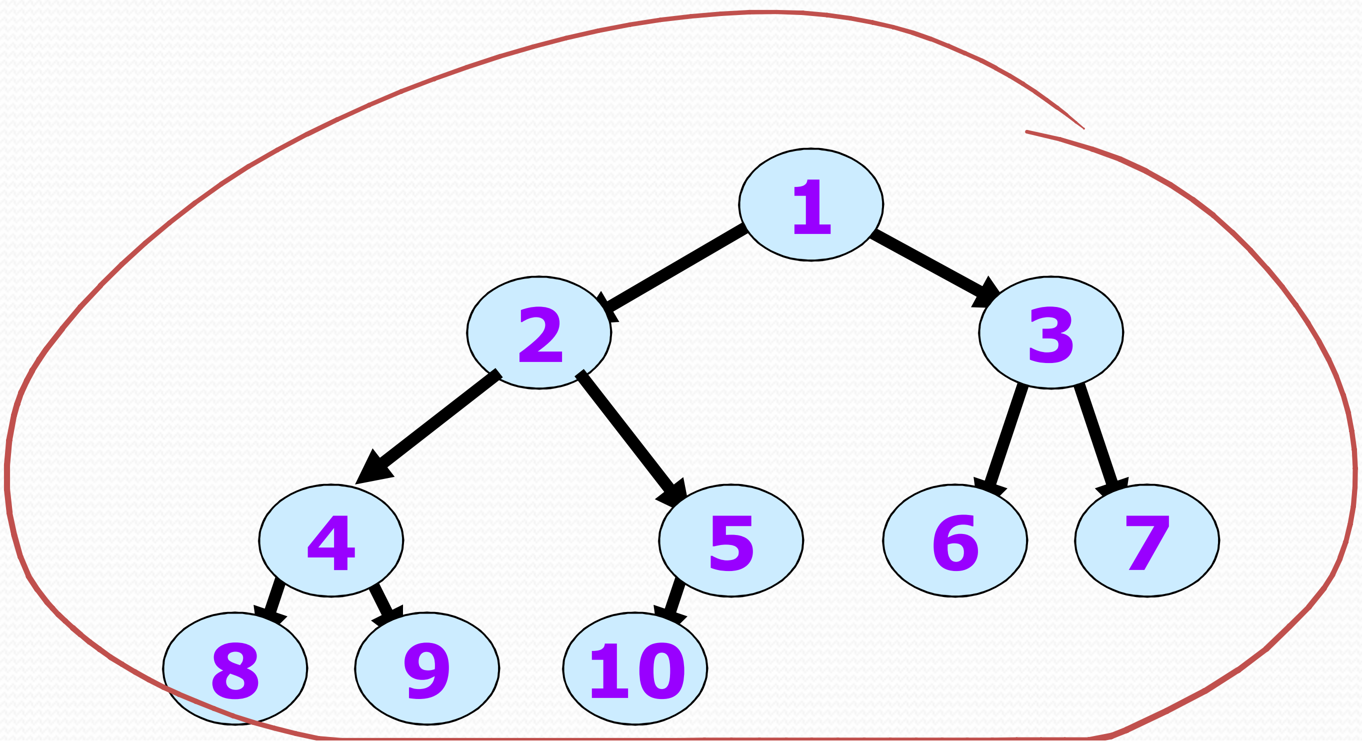
$2i/2 = i$

$(2i+1)/2 = i$

Relation	parent	left child	Right child
	i	$2i$	$2i+1$

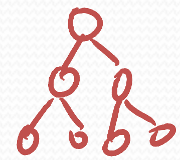
Representing complete binary trees

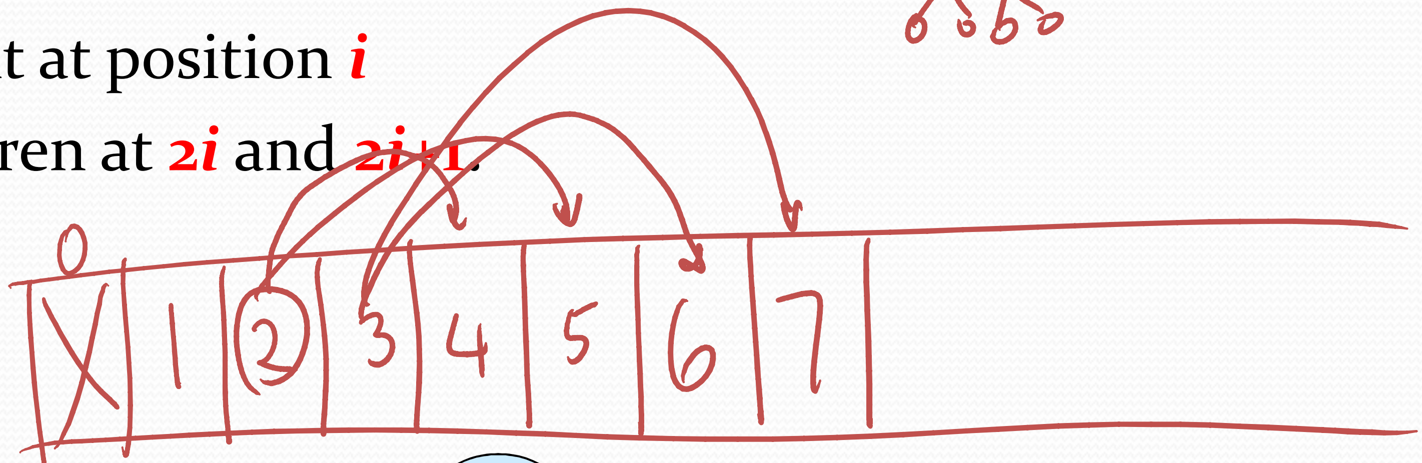
- Can be represented using
  - Linked structures? (hard)
  - Arrays! (easy)

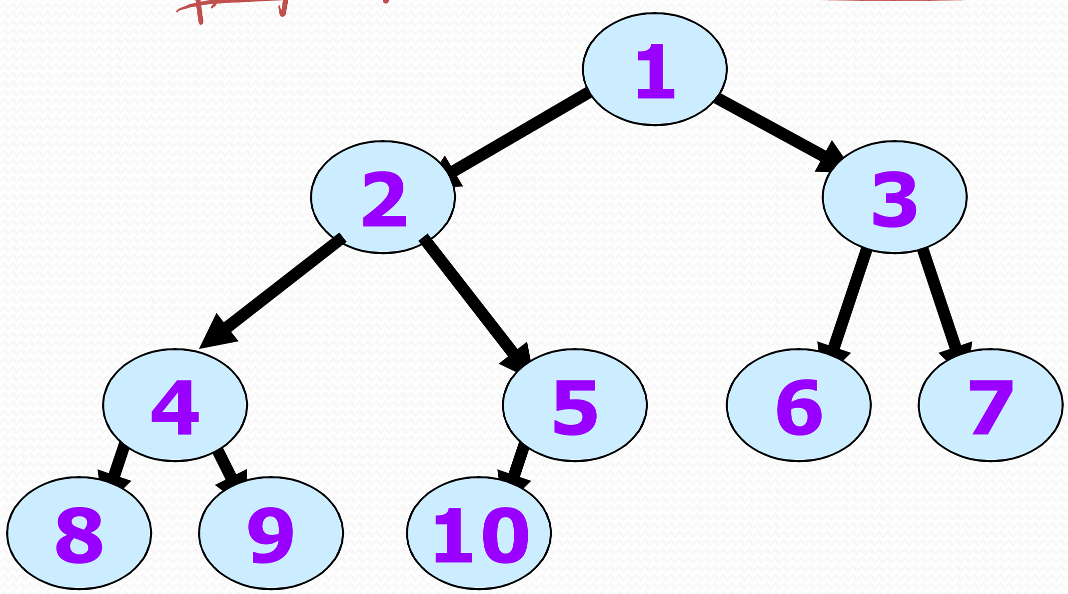


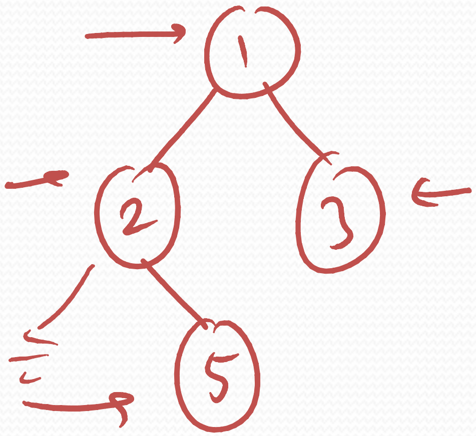
Representing complete binary trees

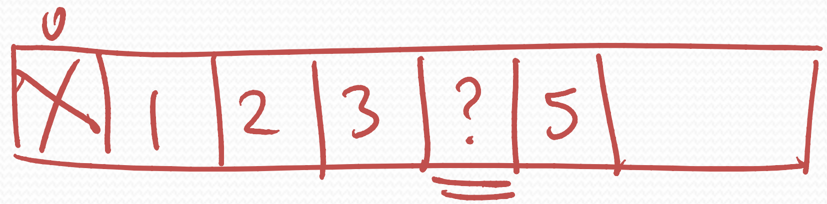
- Arrays
  - Parent at position ***i***
  - Children at **2*i*** and **2*i*+1**.

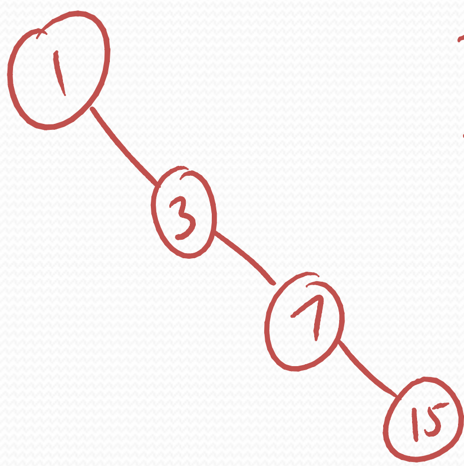


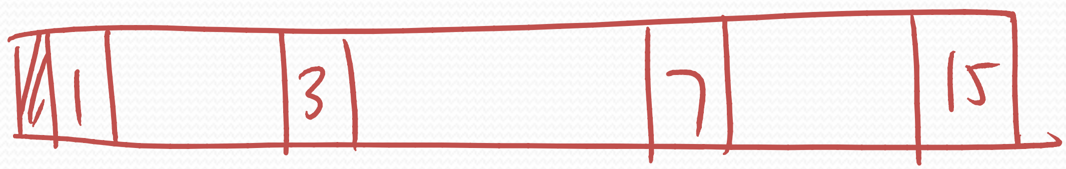





1
2
3
5

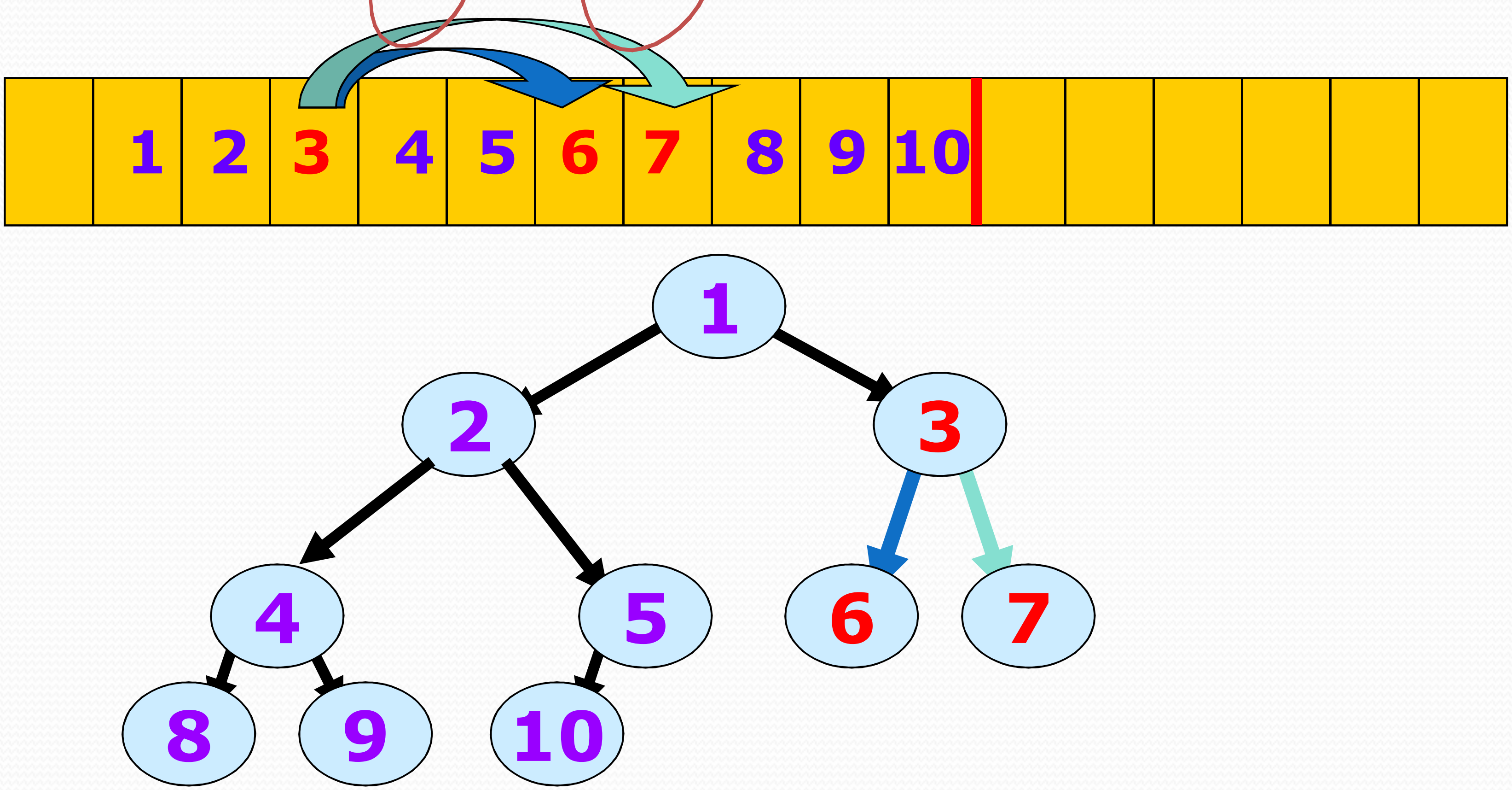

0
1
2
3
?
5


1
3
7
15


1
3
7
15

Representing complete binary trees

- Arrays (1-based)
  - Parent at position i
  - Children at $2i$ and $2i+1$.



A heap can be represented using a complete binary tree

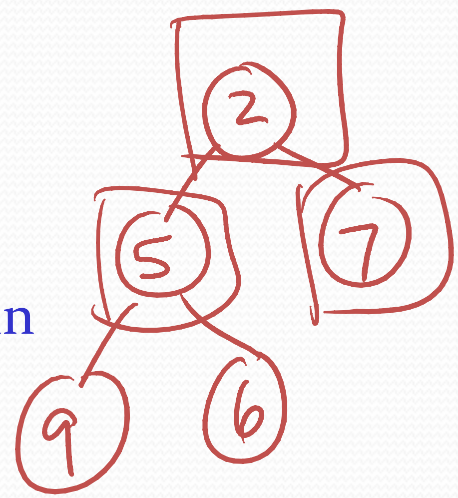
packed array

Binary Heap properties

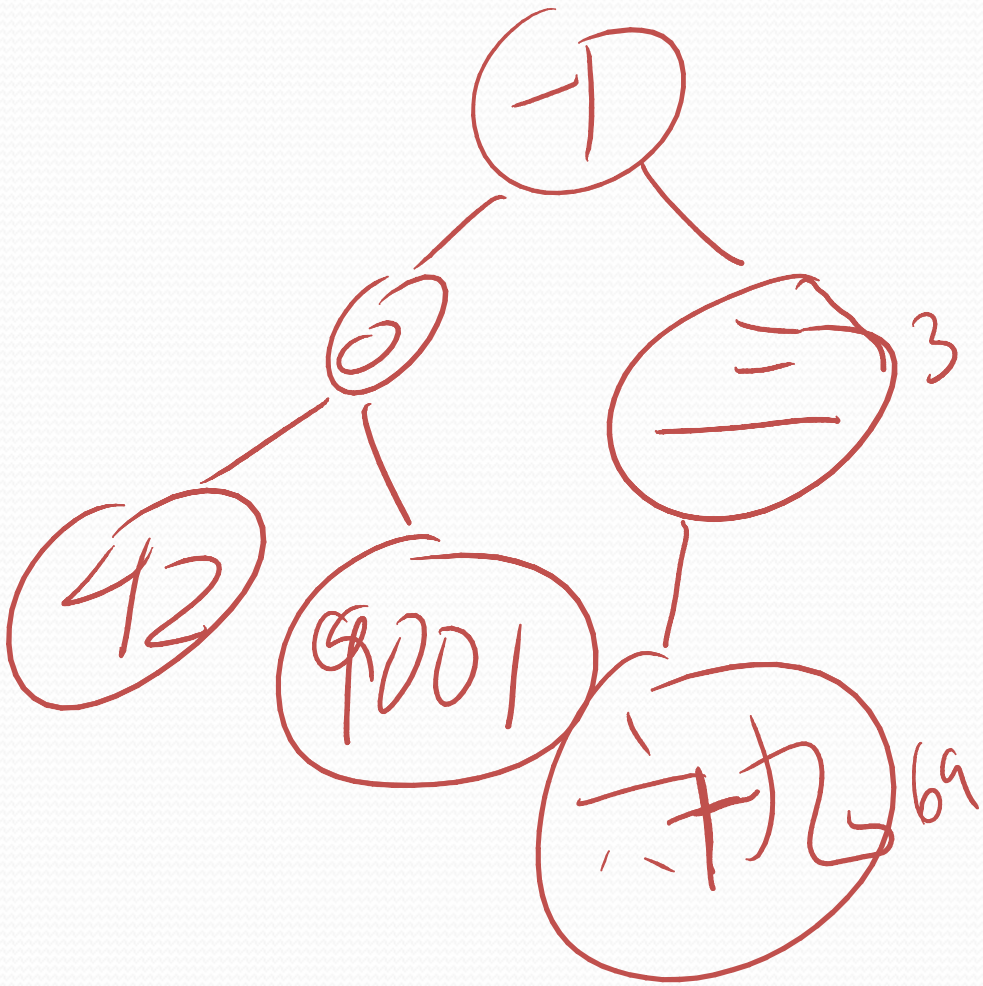
Must satisfy two properties

1. Structure property
    - **Complete binary tree**
    - *Hence*: efficient compact representation

2. Heap order property
    - **Parent keys less than children keys**
    - *Hence*: rapid insert, findMin, and deleteMin
        - $O(\log(N))$ for insert and deleteMin
        - $O(1)$ for findMin



An Example of a binary Heap



Priority Queue operations using a binary heap

- How to code a PQ operations using a Heap?
- findMin() –

  - The code

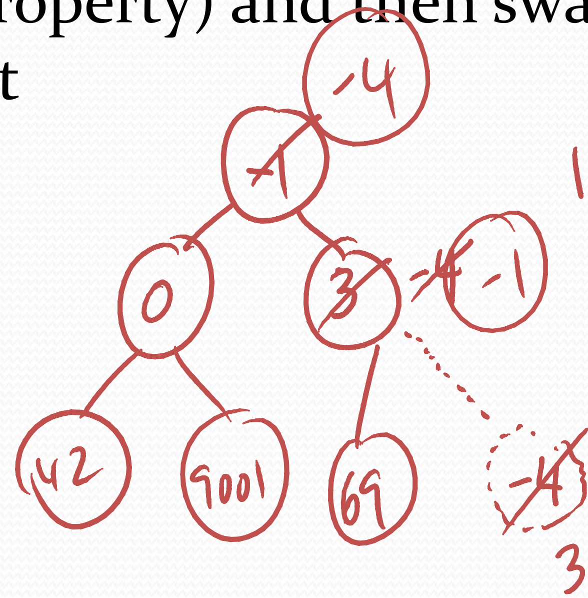
```
public boolean isEmpty() {
    return size == 0;
   }
public Comparable findMin() {
   if(isEmpty()) return null;
   return heap[1];
   }
```

- FindMin() does not change the tree
  - Trivially preserves the invariant

Insert Operation

- Insert(x) -
  - put the new element into next leaf position (to maintain complete tree property) and then swap it up as long as it's <= its parent

- More formally...



insert(-4)

$\rightarrow O(\log n)$

insert (Comparable x)

- Process

1. Create a "*hole*" at the next tree cell for x.

   next Available location

   `heap[size+1]`

   This preserves the ***completeness*** of the tree *assuming it was complete to begin with*.

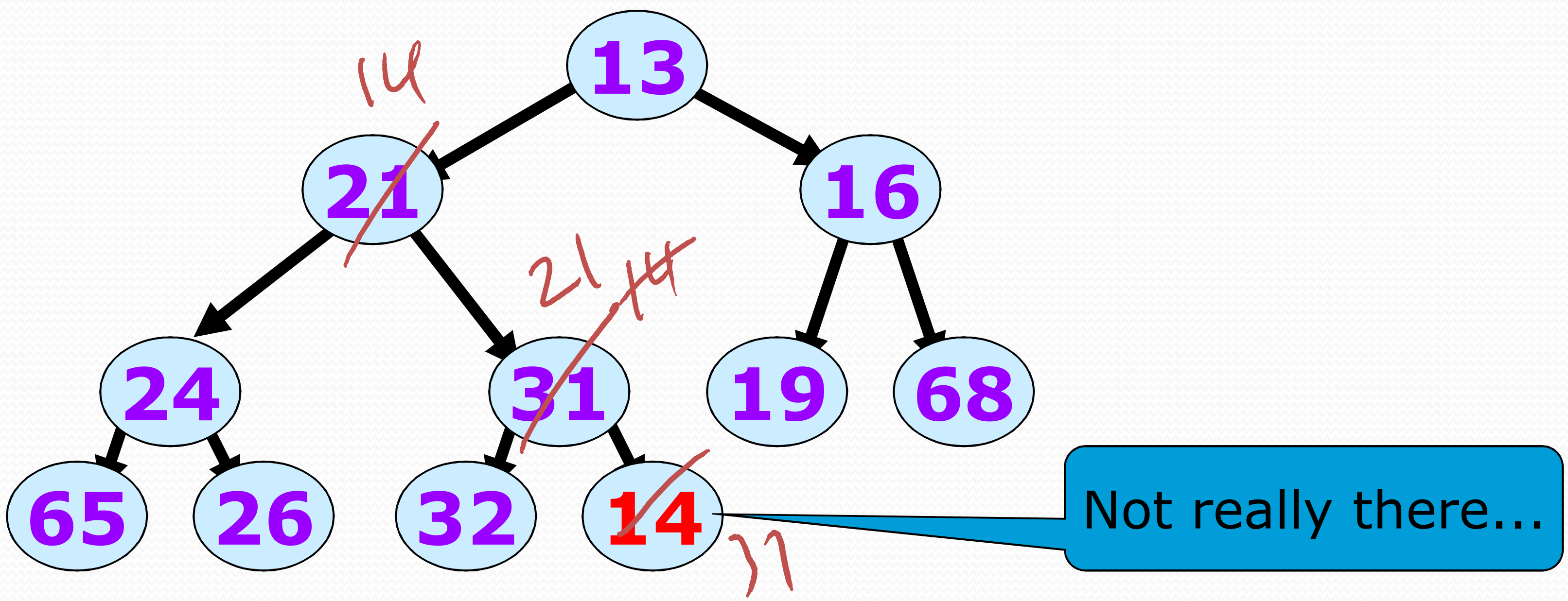
2. *Percolate* the hole *up* the tree until the heap order property is satisfied.

   This assures the ***heap order property*** is satisfied *assuming it held at the outset*.

Percolation up

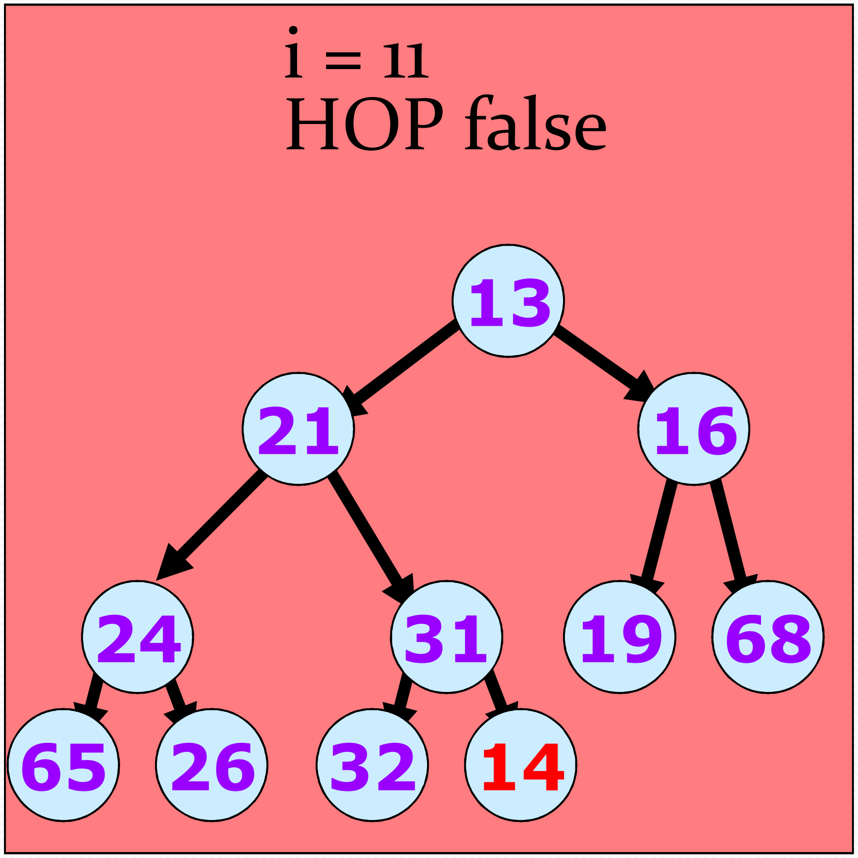
- Bubble the hole ***up the tree*** until the heap order property(HOP) is satisfied.

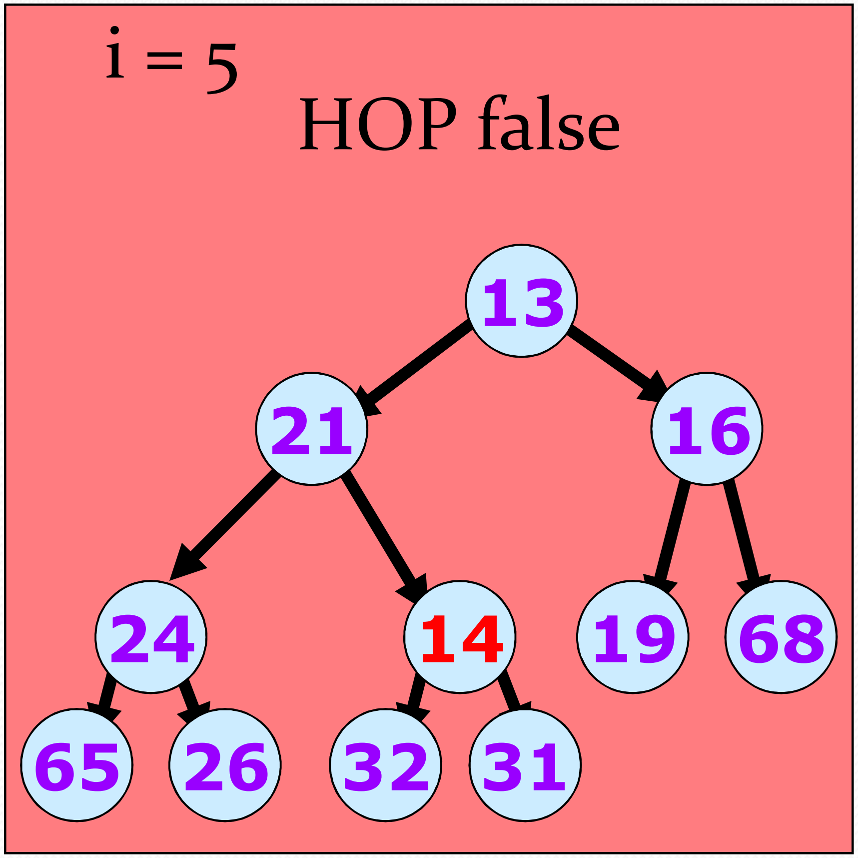
i = 11
HOP false



Percolation up

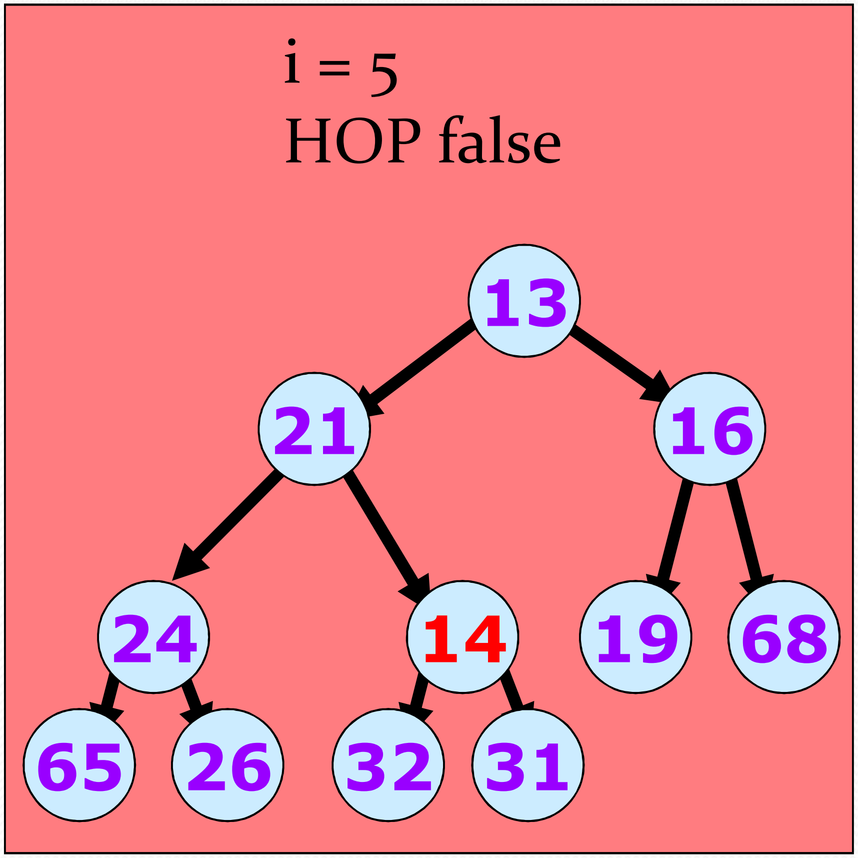
- Bubble the hole up the tree until the heap order property is satisfied.

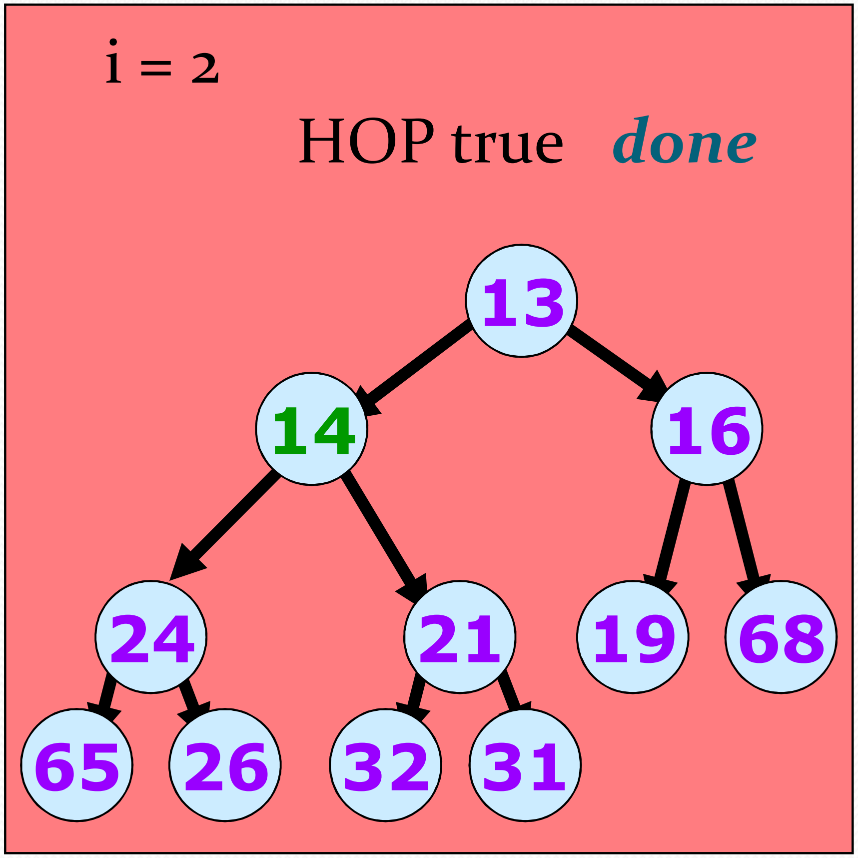




Percolation up

- Bubble the hole up the tree until the heap order property is satisfied.



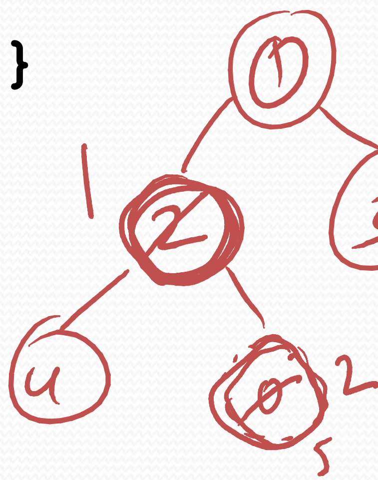


Percolation up

```
public void insert(Comparable x) throws
  Overflow
   {
     if(isFull()) throw new Overflow();
     for(int i = ++size;
         i>1 && x.compareTo(heap[i/2])<0;
          i/=2)
       {heap[i] = heap[i/2];}
     heap[i] = x;
   }
```

$x < heap[i/2]$

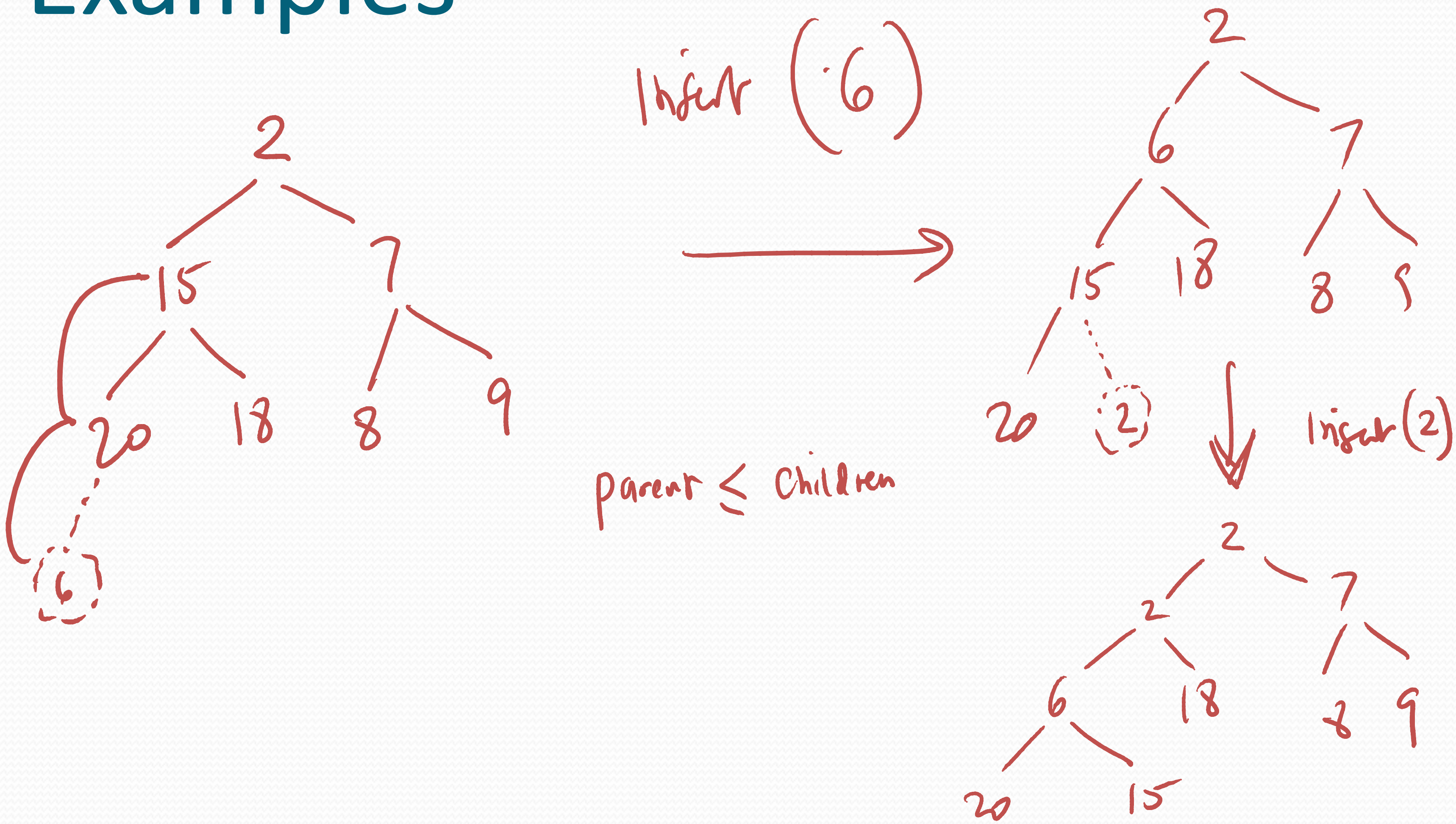
$size = \boxed{4}$



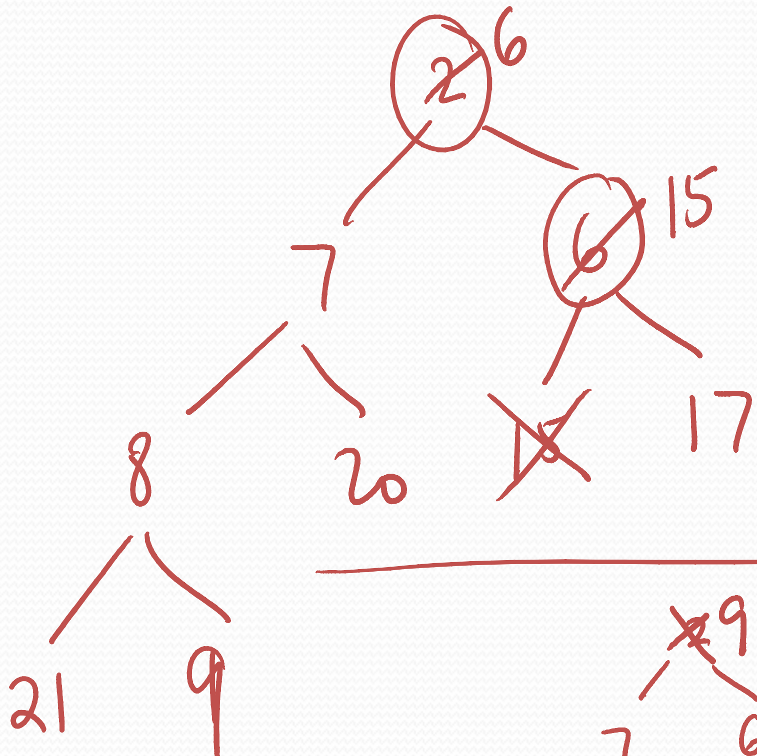
$\boxed{0}$

i: 5, 2, 1

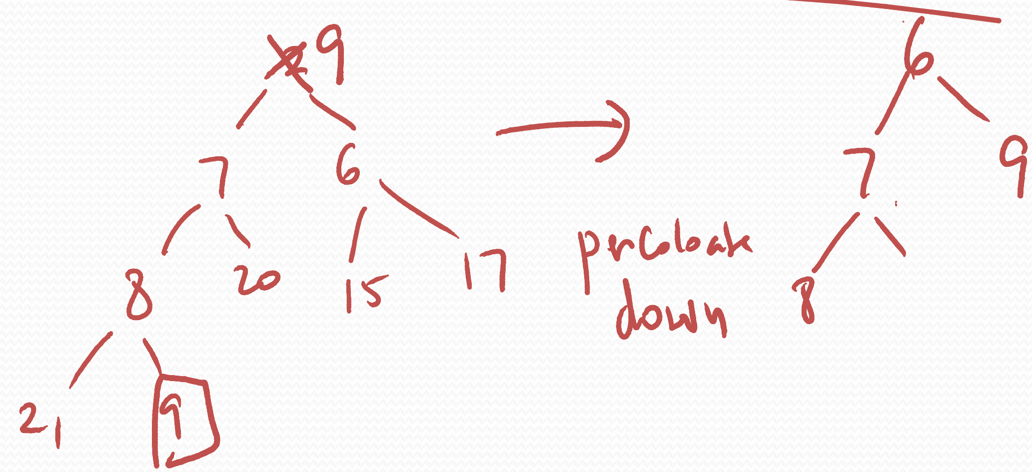
Examples



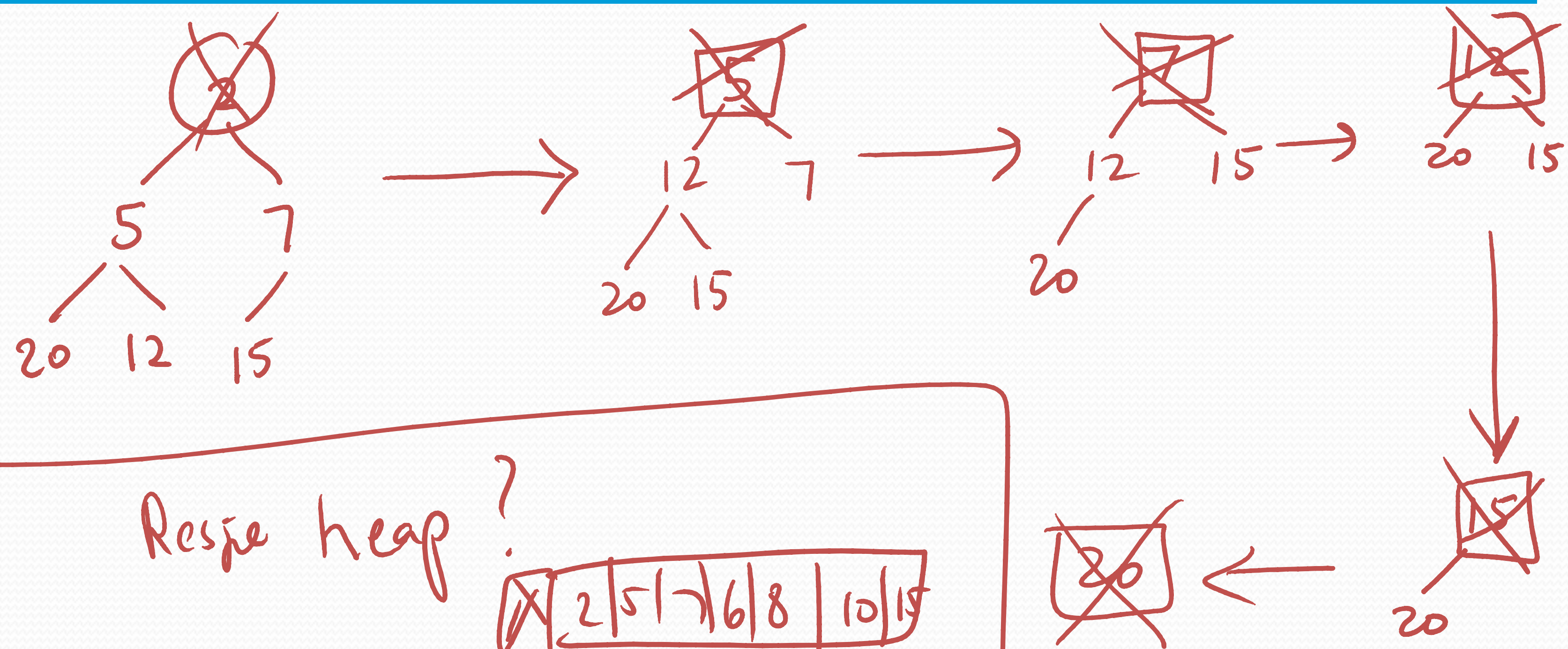
delete min



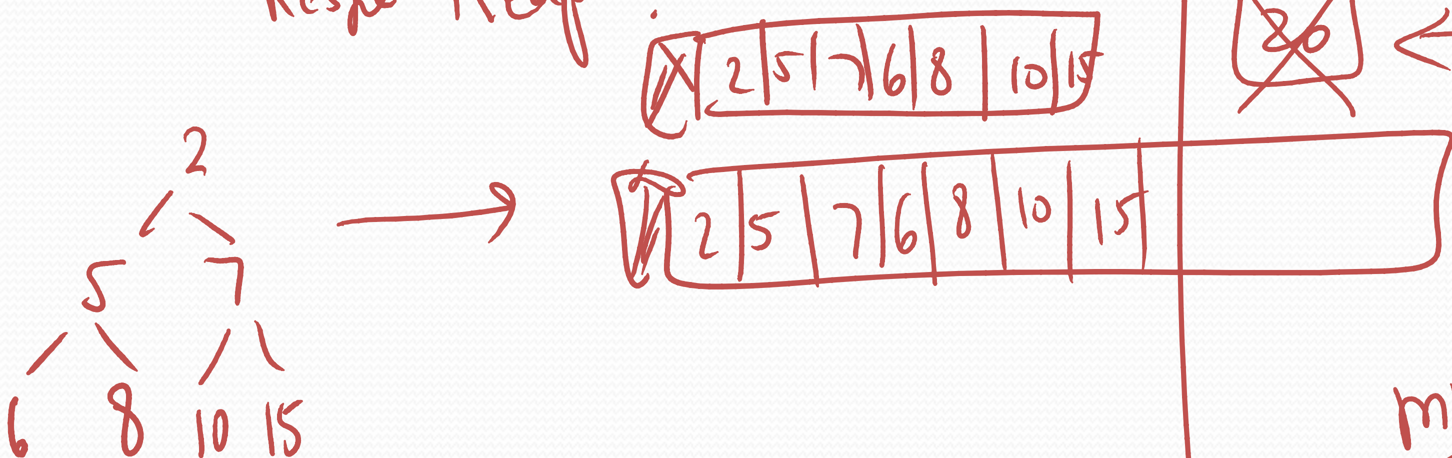
does not work



delete min until empty



Reuse heap ?



min heap

max heap

Is it possible to create a data structure that finds median in $O(1)$ and inserts can be done in $O(\log n)$