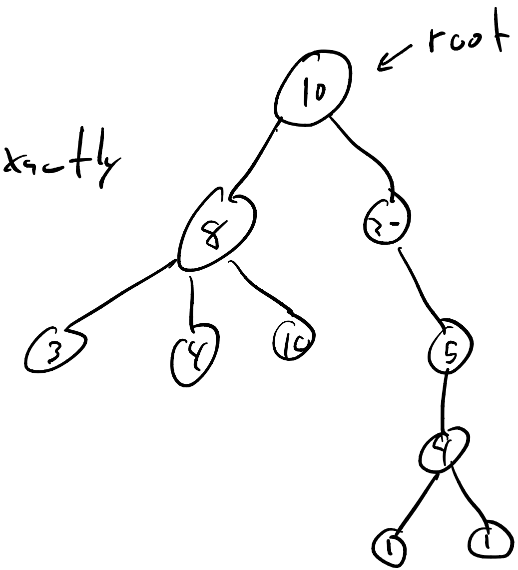
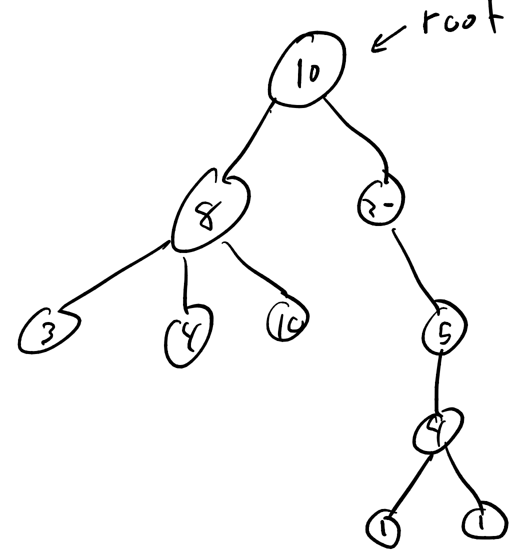
Trees

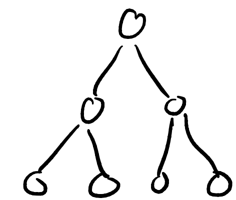
- Hierarchical data structure
- Composed of nodes (or vertices) which have children, and parents are connected to children by edges (or links)
- Nodes may have values
- Every node except the root must have exactly one parent
- Nodes with children are internal nodes
- Nodes w/o children are leaves



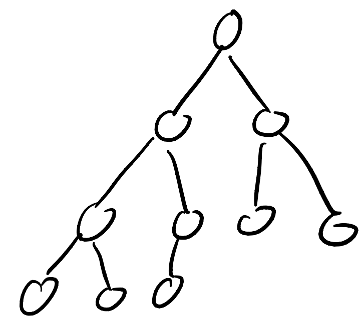
- The height of a node is the length of the longest downward path from the node to a leaf (number of edges in the path)
- The height of the tree is the height of the root
- The depth of a node is the length of the path from the node to the root



- Binary trees
    - Tree where every node has at most 2 children
    - Complete binary tree (according to our book terminology)
        - All leaves have the same depth
        - All nodes have 0 or 2 children
        - Number of nodes $n = 2^{h+1} - 1$
        - height $h = \log_2 (n+1) - 1 = \Theta(\lg n)$
        - number of leaves is 2^h



- nearly complete binary tree
    - Every level, except possibly the last, is completely filled
    - All leaves are as far to the left as possible

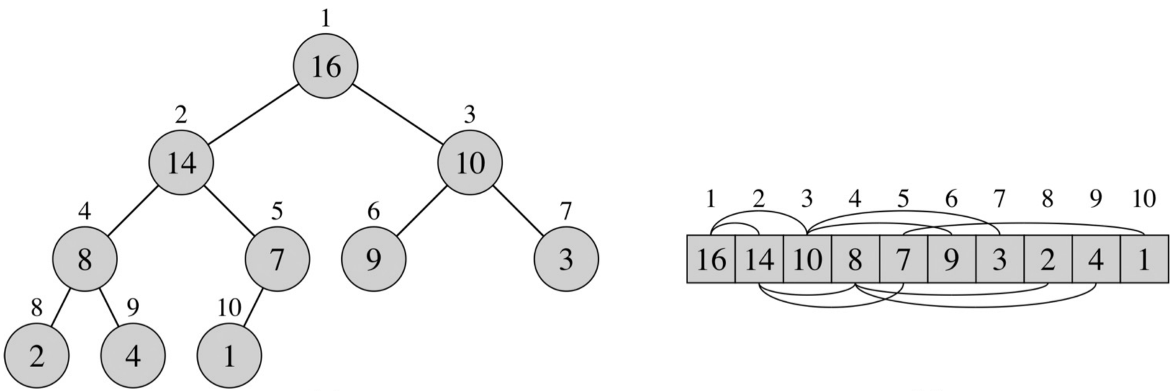


Representing Trees

- If a node may have an unbounded number of children
    - Use a linked structure
    - A node's children are stored as an array or list of pointers

- representing complete or nearly complete binary trees

  - Use an array



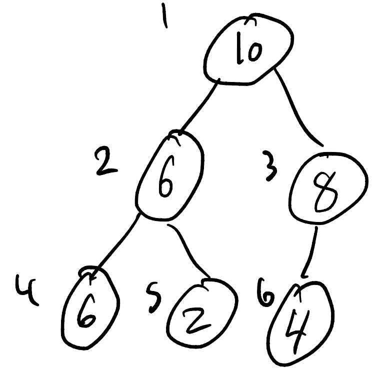
(a) (b)

- First element (index $i = 1$) is the root

- The left child of node at index i is at $2i$, the right at $2i+1$

  LEFT(i) returns $2i$, RIGHT(i) returns $2i+1$, PARENT(i) returns $\lfloor i/2 \rfloor$

Binary Heap

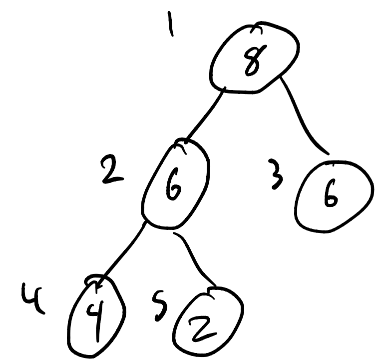


10	6	8	6	2	4
1	2	3	4	5	6

- Nearly complete binary tree
- Represented by an array
- Max heap
  - The value of a parent is greater than or equal to its children
  - The largest value is the root
  - Max heap property: $A[PARENT(i)] \geq A[i]$
- Min heap
  - Smallest value is the root, $A[PARENT(i)] \leq A[i]$

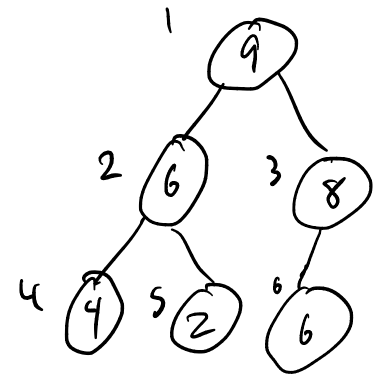
Priority Queue

- Like a normal queue, items can be pushed into the queue
- Unlike a normal queue, the next item to be popped is the item with the highest (or lowest with a min heap) value
- Implementation using a heap
  - Next element to pop is the root
  - When popping, replace the root with the rightmost leaf in the lowest level
  - While the heap property is violated, swap the node that violates the property with its largest child



- Popping is $O(\lg n)$ because the max number of swaps is the height of the tree

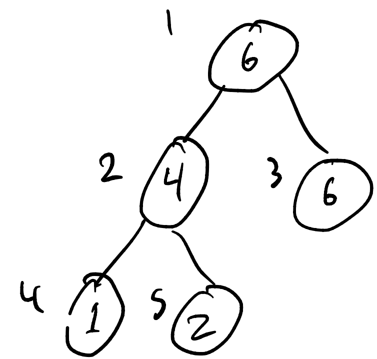
- To push, add a node as the leftmost possible leaf. While the heap property is violated, repeatedly swap the node that violates the property with its parent



- Pushing is also $O(\lg n)$

A heap in array A of size n. Assumes the trees rooted at LEFT(i) and RIGHT(i) are max heaps, but A[i] might be smaller than its children

MAX-HEAPIFY(A, i, n)
  $l = \text{LEFT}(i)$
  $r = \text{RIGHT}(i)$
  if $l \leq n$ and $A[l] > A[i]$
    $largest = l$
  else $largest = i$
  if $r \leq n$ and $A[r] > A[largest]$
    $largest = r$
  if $largest \neq i$
    exchange $A[i]$ with $A[largest]$
    MAX-HEAPIFY$(A, largest, n)$



$i = 2$

$n = 5$

- Used when popping from a heap priority queue

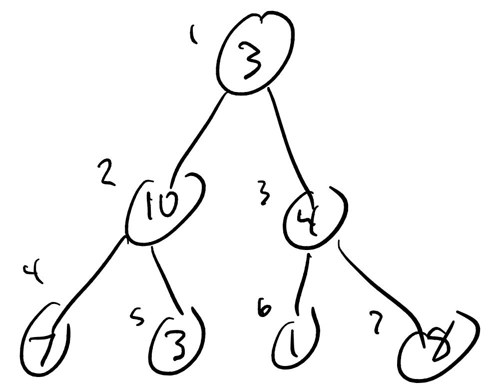
$O(\lg n)$

Turns array A into a max heap

BUILD-MAX-HEAP(A, n)
  for $i = \lfloor n/2 \rfloor$ **downto** 1
    MAX-HEAPIFY(A, i, n)

Iteratively uses MAX-HEAPIFY starting at the second to lowest level and works its way up

This is $O(n \lg n)$, but that is not the tightest bound



Heap height is $\lfloor \lg n \rfloor$

There are at most $\left\lceil \frac{n}{2^{h+1}} \right\rceil$ nodes at any height h

MAX-HEAPIFY requires $O(h)$ time for a node of height h

$$\sum_{h=0}^{\lfloor \lg n \rfloor} \left\lceil \frac{n}{2^{h+1}} \right\rceil O(h) = O\left(n \sum_{h=0}^{\lfloor \lg n \rfloor} \frac{h}{2^h}\right)$$

By the equation A.8 in the appendix

$$\sum_{h=0}^{\infty} \frac{h}{2^h} = \frac{\frac{1}{2}}{(1-\frac{1}{2})^2} = 2$$

The whole thing is $O(n)$

$O(n \lg n)$

HEAPSORT(A, n)

```
BUILD-MAX-HEAP(A, n)
for i = n downto 2
    exchange A[1] with A[i]
    MAX-HEAPIFY(A, 1, i - 1)
```