CS151 Intro to Data Structures

Heaps

Announcements

- HW5 and Lab7 due Friday
- Quiz during lab on Wednesday

Outline

1. Heaps
   a. Definition
   b. Operations
   c. Implementing a heap
2. Selection Sort
3. Heapsort

Heaps

Heaps

Allows you to efficiently access and modify the element with the **lowest value**

Binary Heap Properties

1. Each node has **at most 2 children**

1. For every node n (except for the root): **n.key >= parent(n).key**

1. **Complete:** all levels of the tree, except possibly the last one are fully filled, and, if the last level of the tree is not fully filled, the nodes of that level are filled from left to right
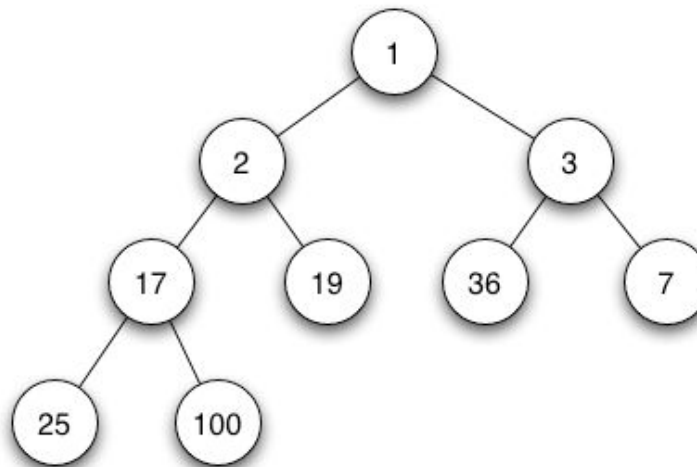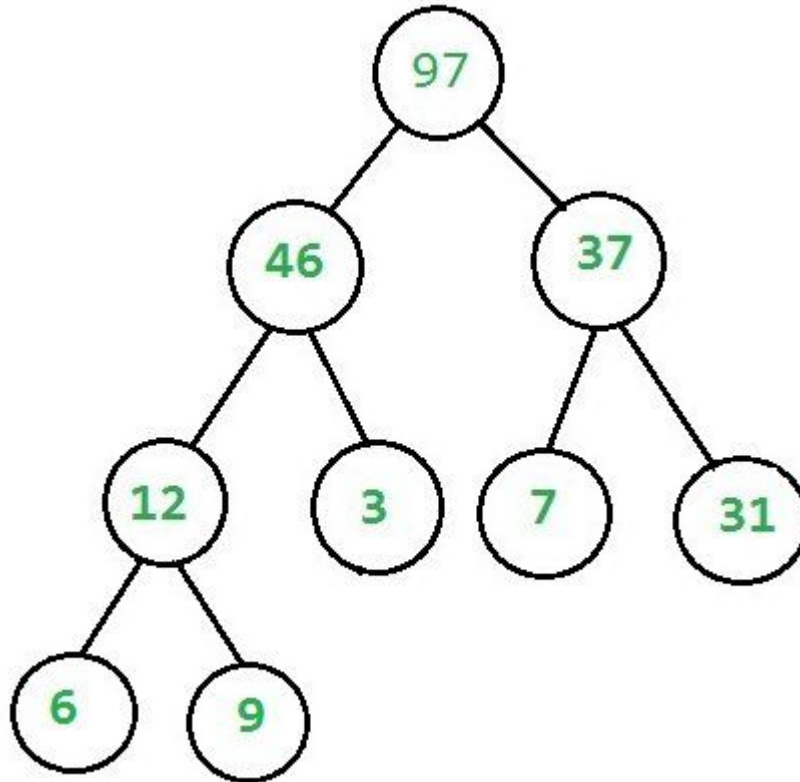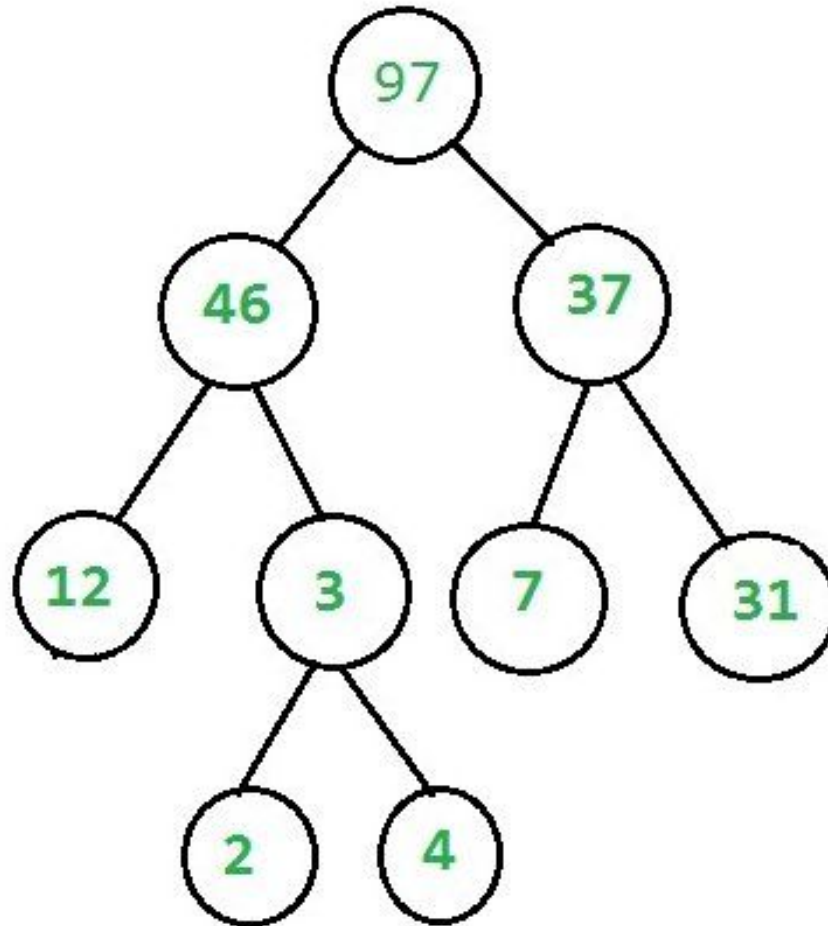
Complete Binary Trees

All levels of the tree, except possibly the last one are fully filled, and, if the last level of the tree is not complete, the nodes of that level are filled from left to right
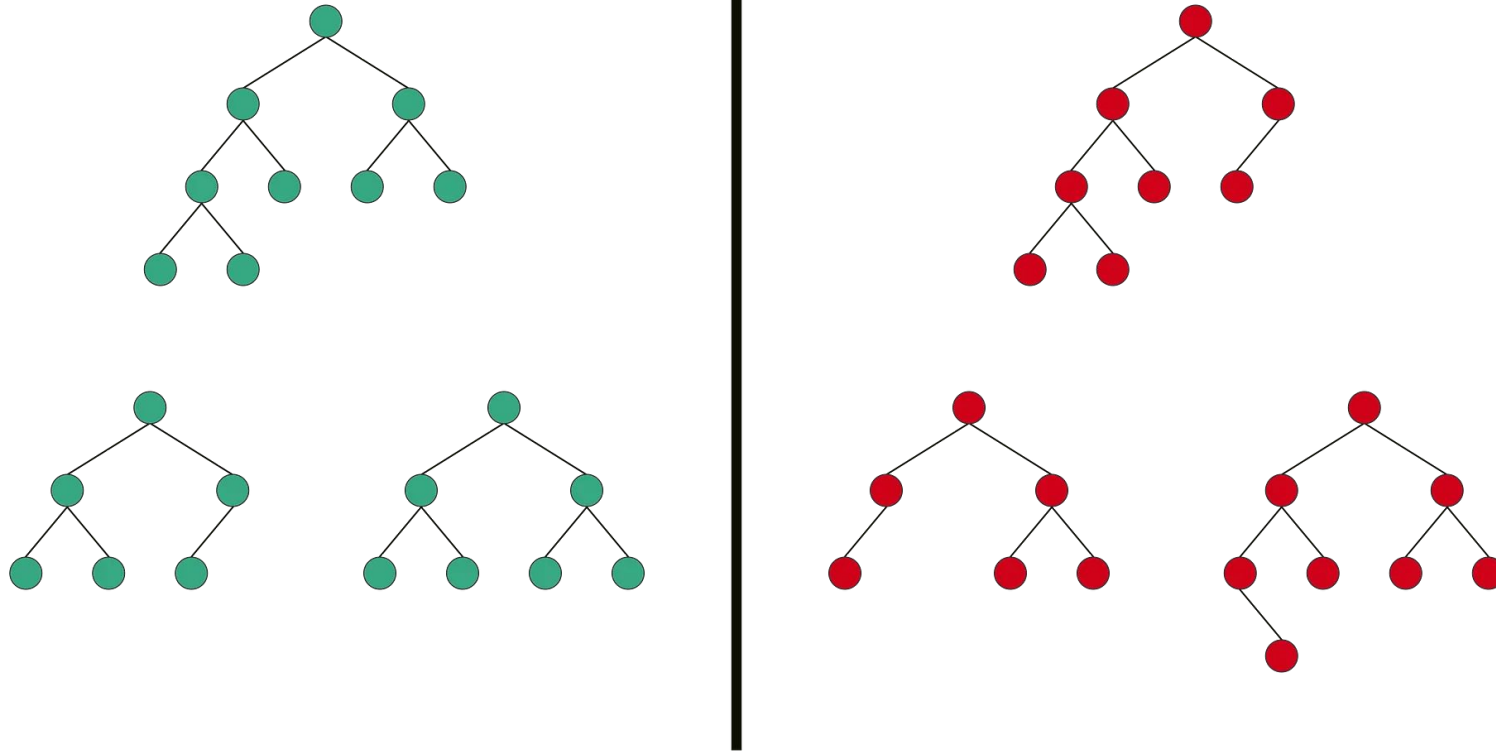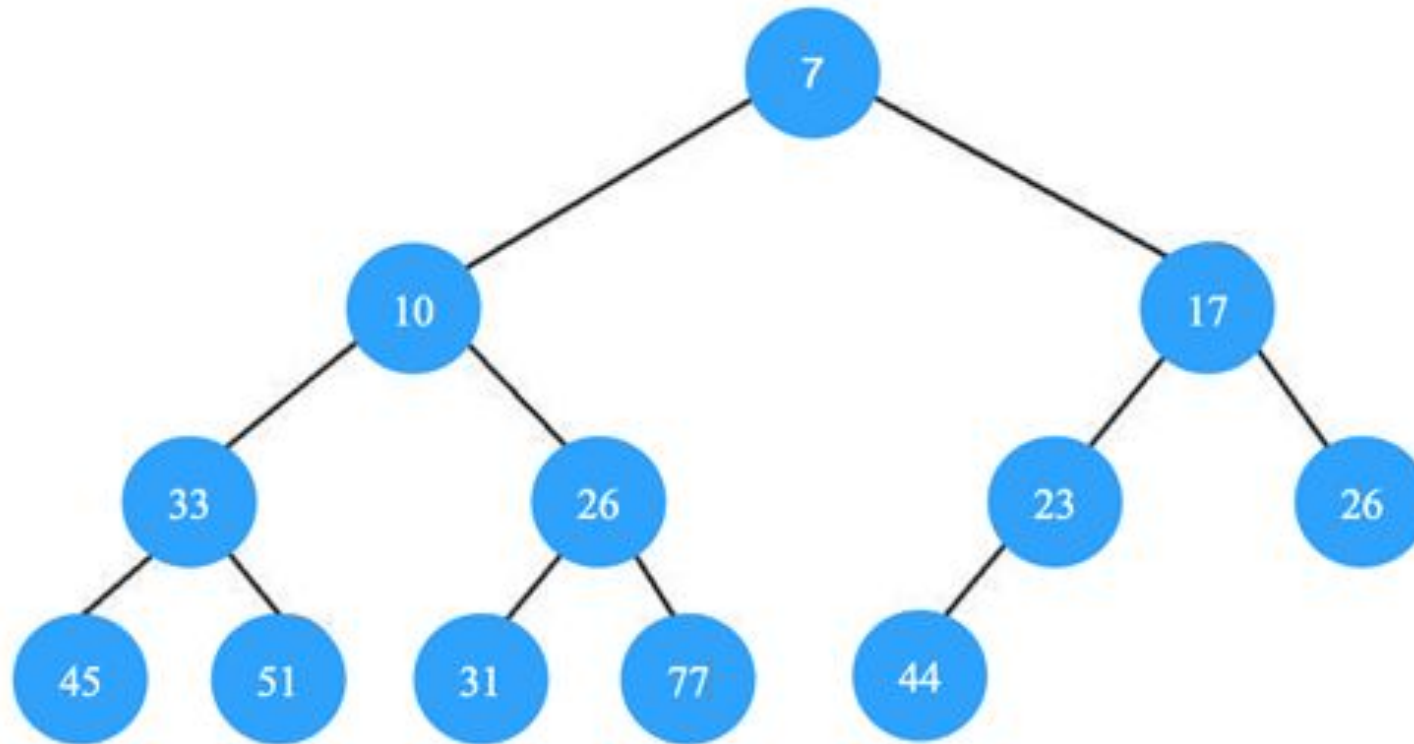
CS151 - Lecture 11 - Spring '25
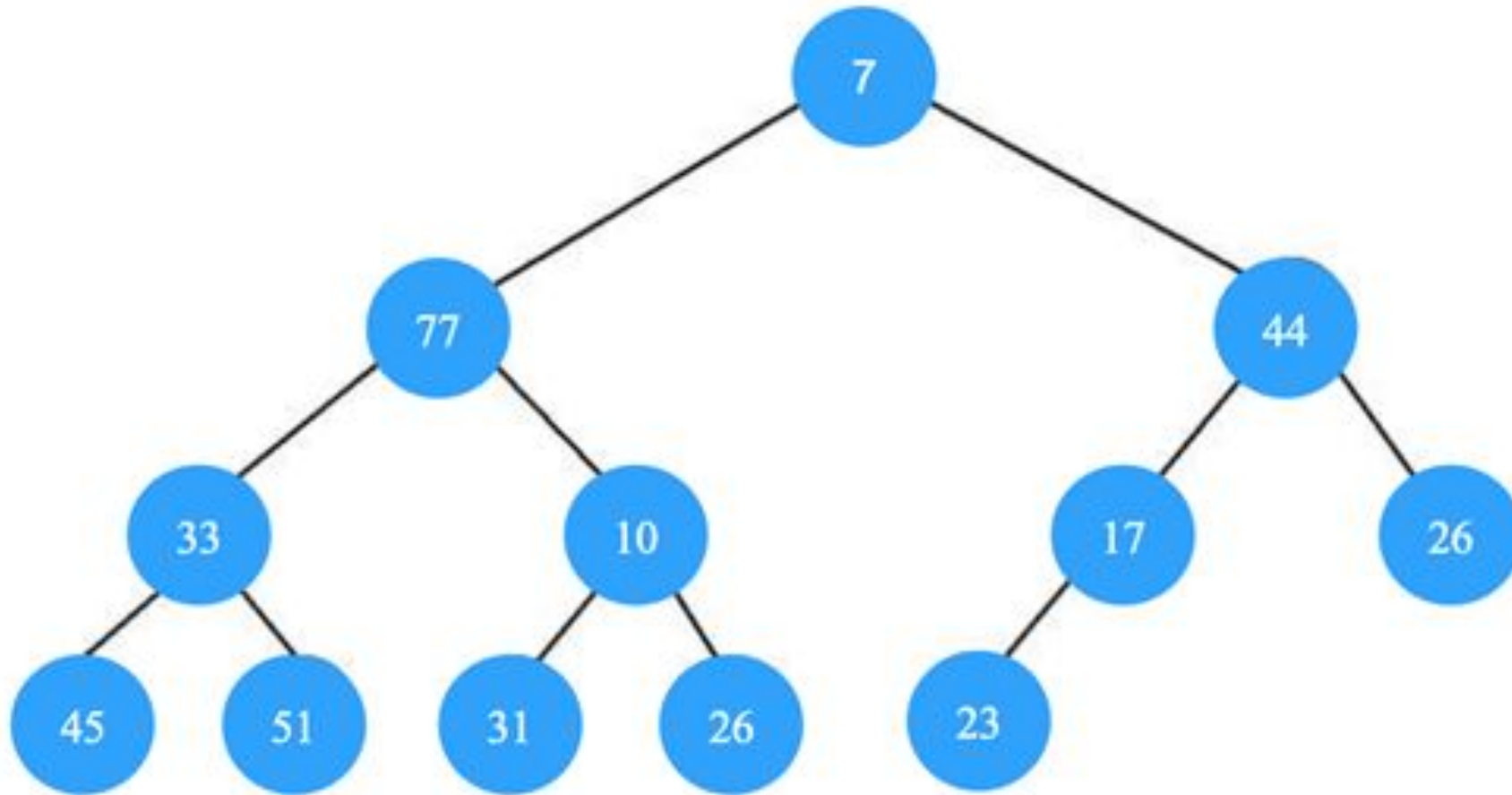
Is this a binary min heap?

Is this a binary min heap?
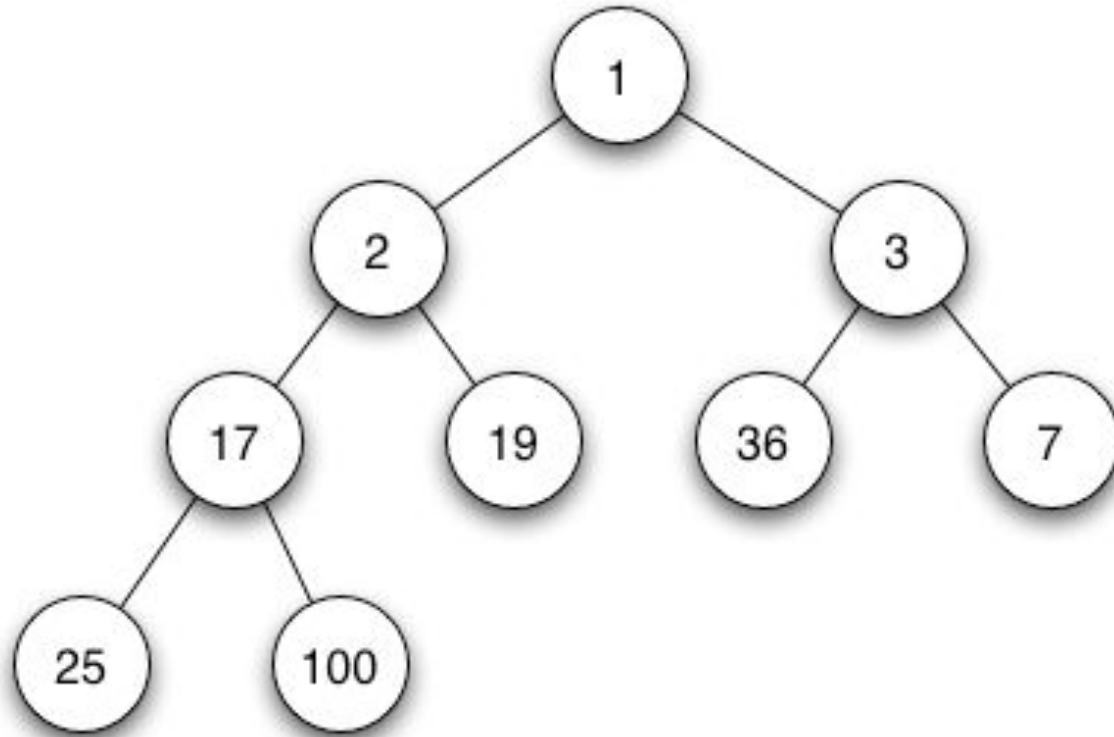
Is this a binary min heap?

Is this a binary min heap?
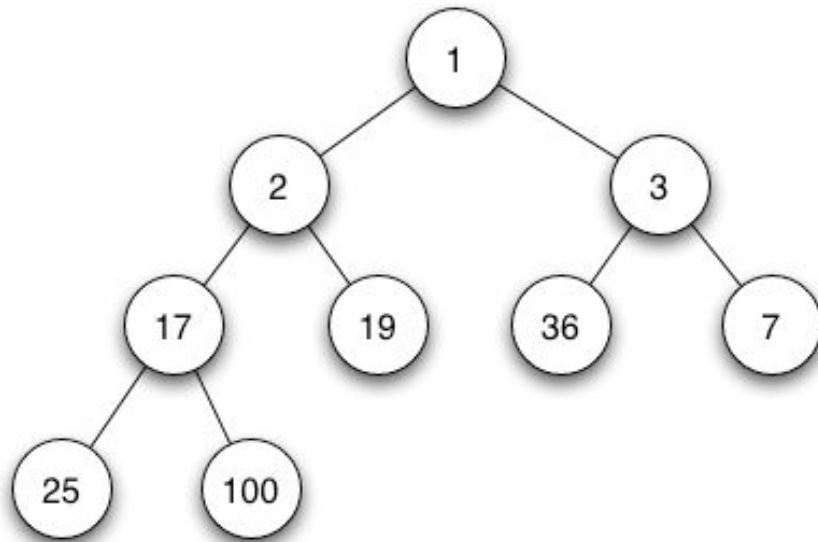
Heap Properties

- Height?
  - log(n)

Heap Operations:

1. Insert
   a. Upheap
2. Poll
   a. Downheap
3. Remove
4. Search

Heap Insertion

Need to maintain:

1. completeness
2. key order property: **n.key >= parent(n).key**

`insert(15)`

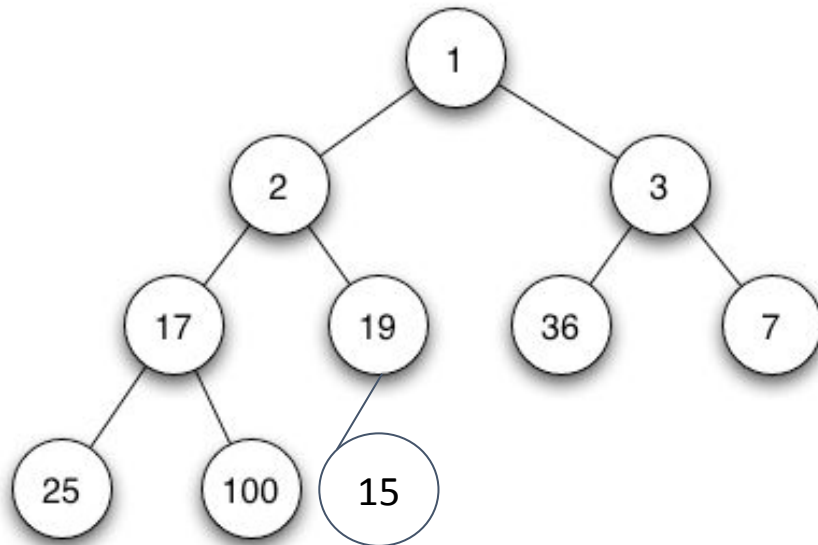# Heap Insertion

Need to maintain:

UPHEAP!

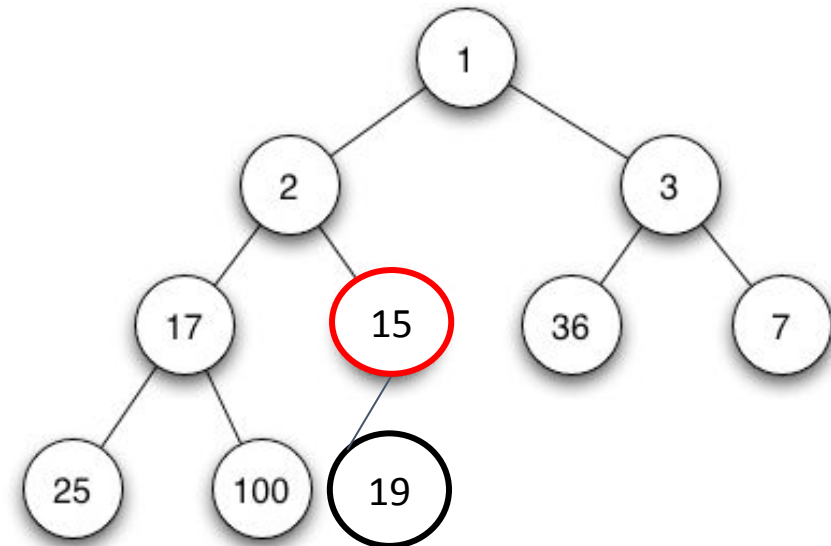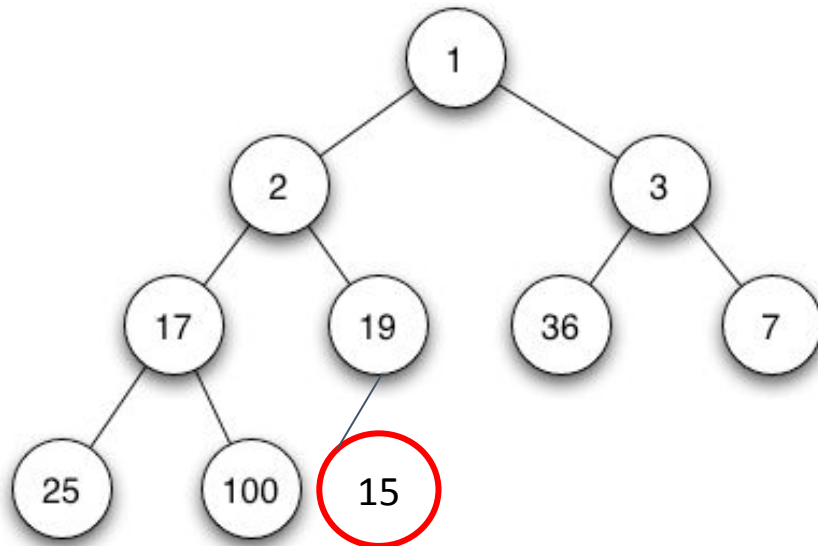1. completeness
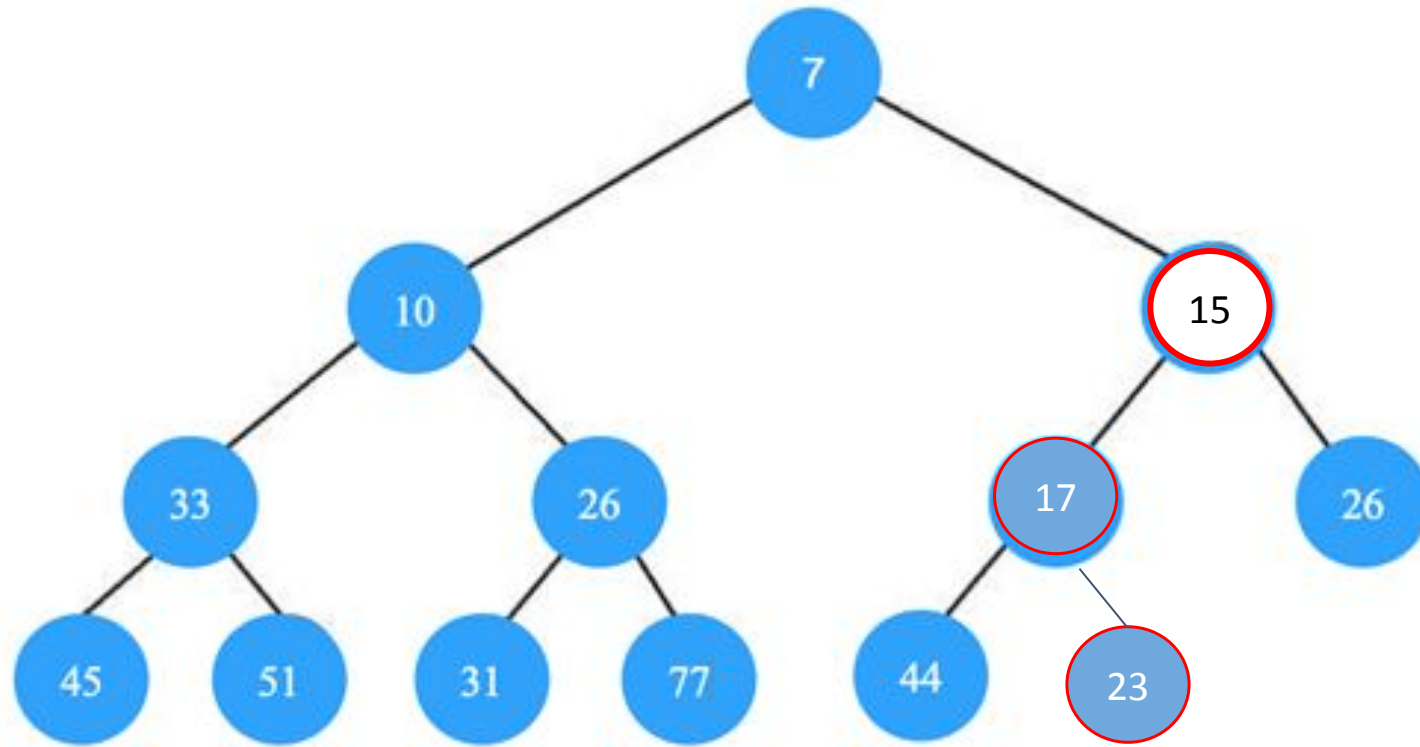2. key order property: **n.key >= parent(n).key**

insert(15)

Heap Insertion - Upheap

1. Compare the added element with its parent; if they are in the correct order, stop.
2. If not, swap the element with its parent and return to the previous step.

Heap Insertion

insert(15)

Heap insertion

1. Add the element to the bottom level of the heap at the leftmost open space.
2. Compare the added element with its parent; if they are in the correct order, stop.
3. If not, swap the element with its parent and return to the previous step.
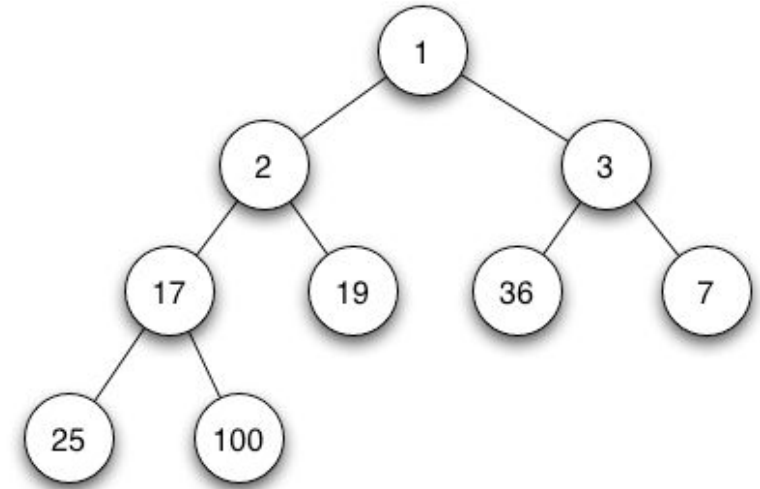
Runtime complexity?

O(logn)

What about for an array implementation?

O(logn) assuming expansion is not necessary

Heap **Poll**

- Removing the root
- Also called extract

What properties do we need to maintain?
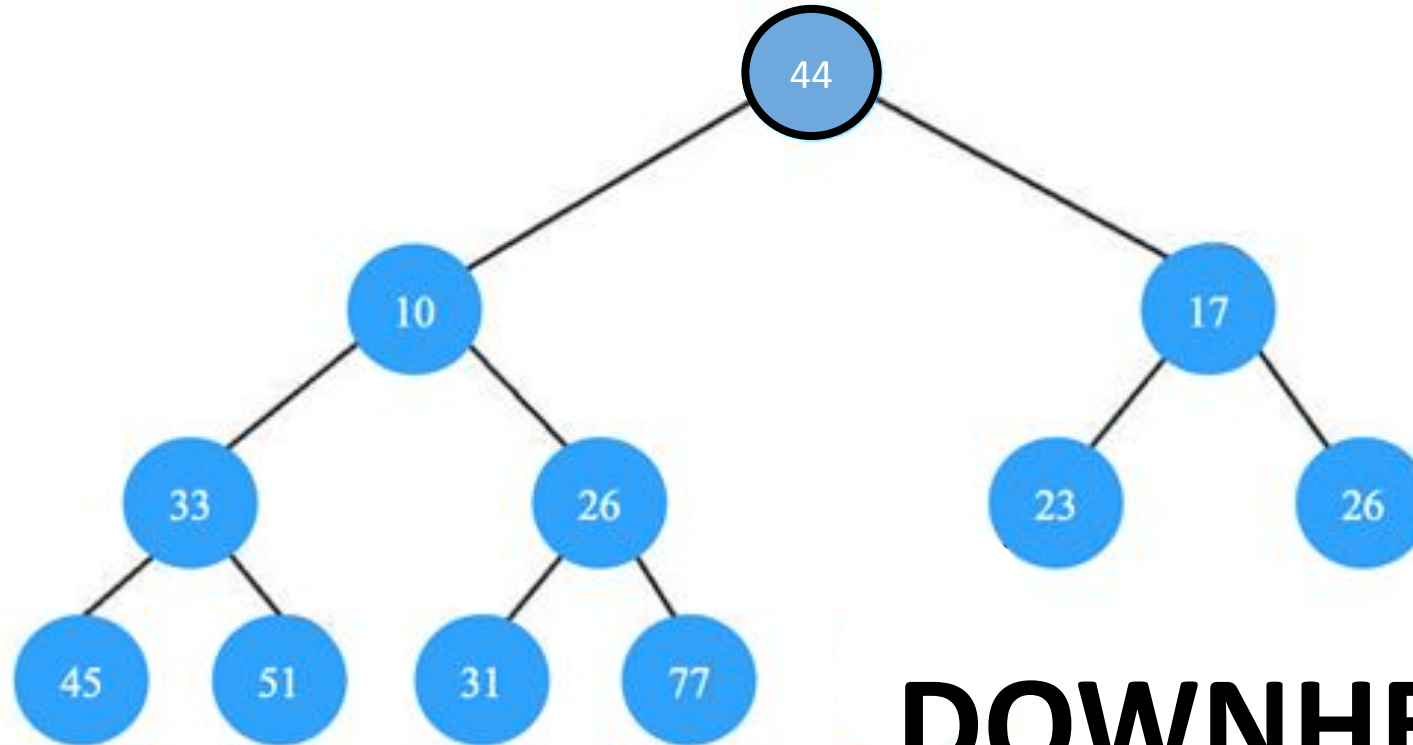
1. completeness
2. key order property: **n.key >= parent(n).key**

Heap Poll

1. Replace the root of the heap with the last element on the last level.
2. Compare the new root with its children; if they are in the correct order, stop.
3. If not, swap the element with one of its smallest children and return to the previous step

Heap Poll Example

1. Replace the root of the heap with the last element on the last level.

DOWNHEAP!

1. Compare the new root with its children; if they are in the correct order, stop.
2. If not, swap the element with one of its smallest children and return to the previous step

Heap Poll

1. Replace the root of the heap with the last element on the last level.
2. Compare the new root with its children; if they are in the correct order, stop.
3. If not, swap the element with one of its smallest children and return to the previous step
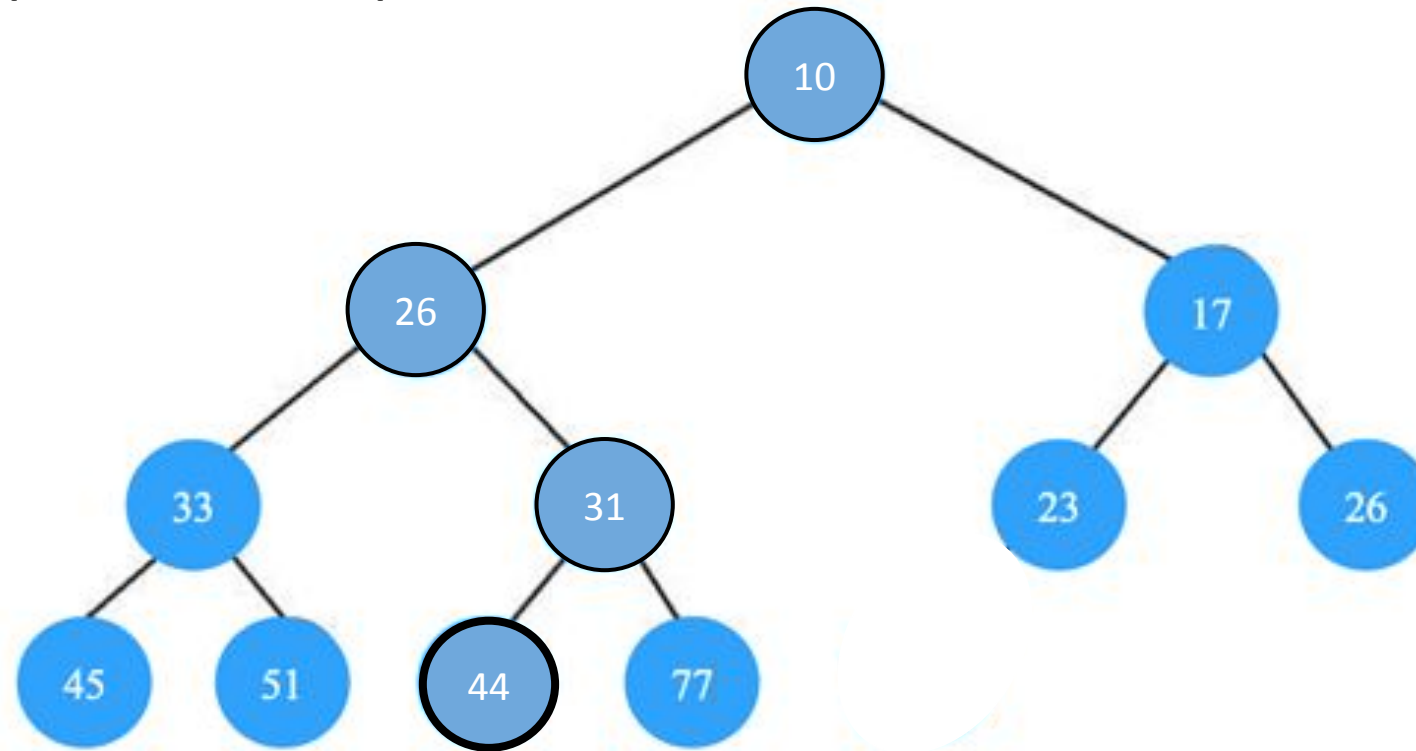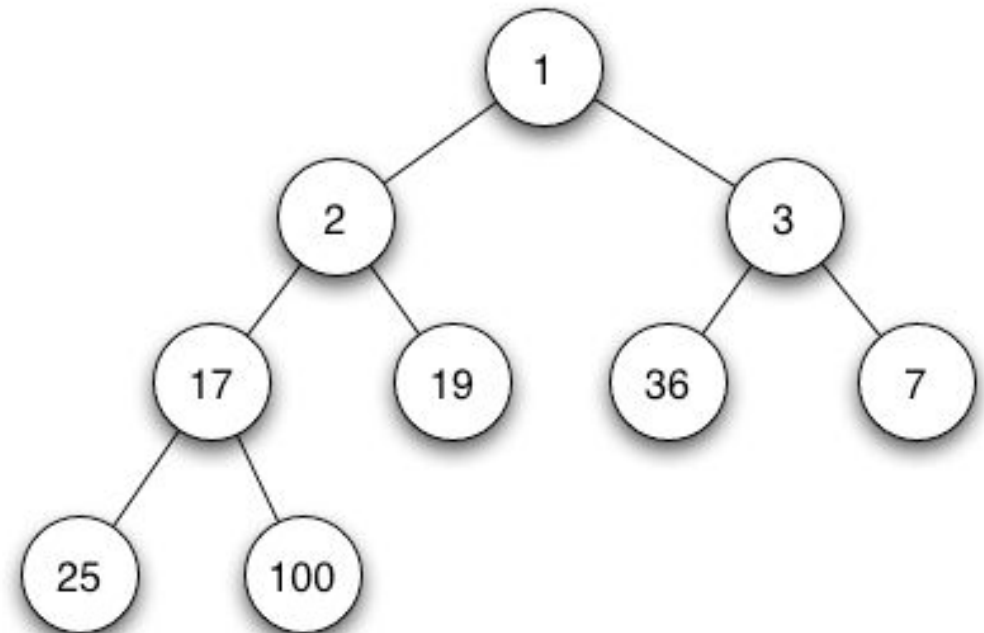
Runtime?

   O(logn)

Array Implementation?

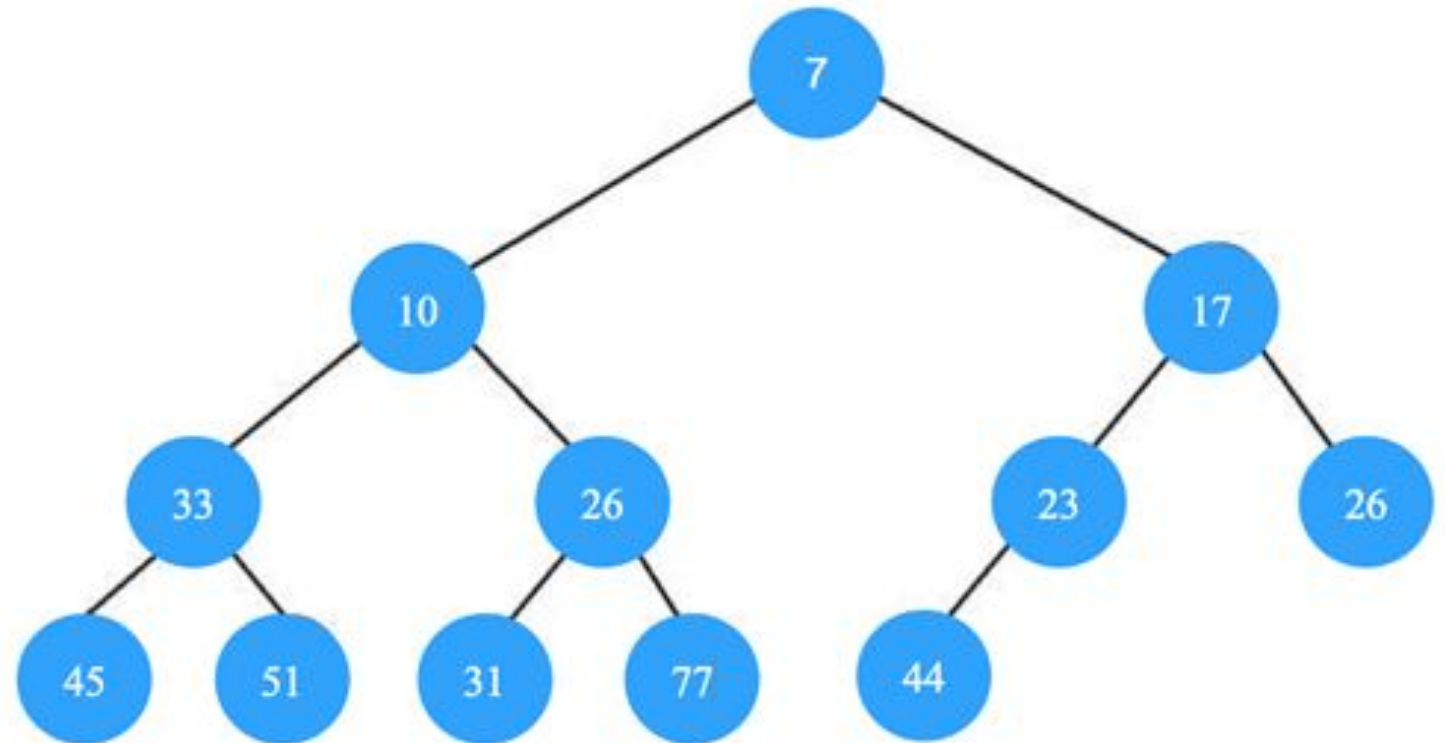   O(logn) - no expansions necessary for poll

Heap Search

- Is the element 36 in the heap?

- not as efficient as searching a BST
  - Weaker ordering property!
  - Best we can do is check every node
  - Shortcuts?

- Runtime complexity?
  - O(n)

Heap Search

- Is the element 11 in the heap?

- Shortcuts?
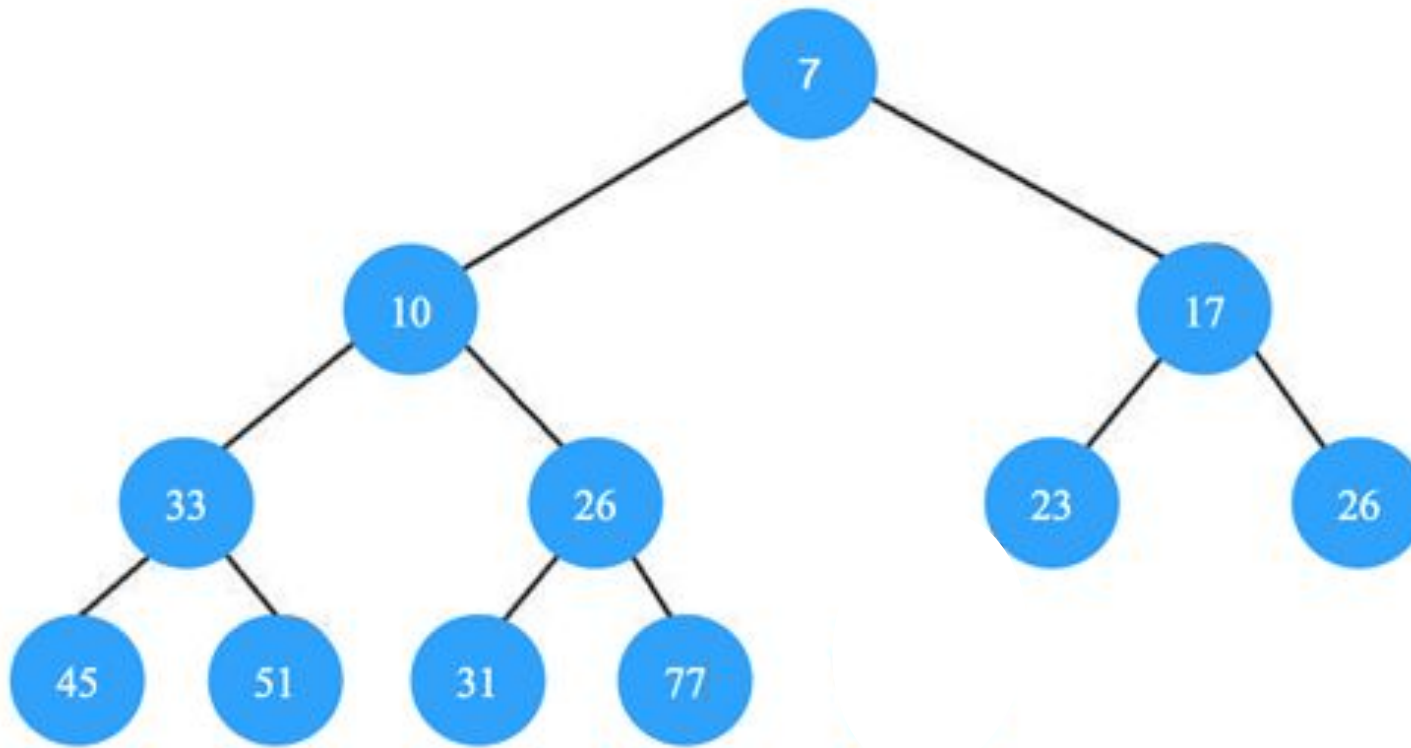 - Can stop once we hit

- Runtime complexity?
  - O(n)

Heap **Remove**

More general case of poll

1. Find the index i of the element we want to delete
2. Swap this element with the last element (rightmost leaf)
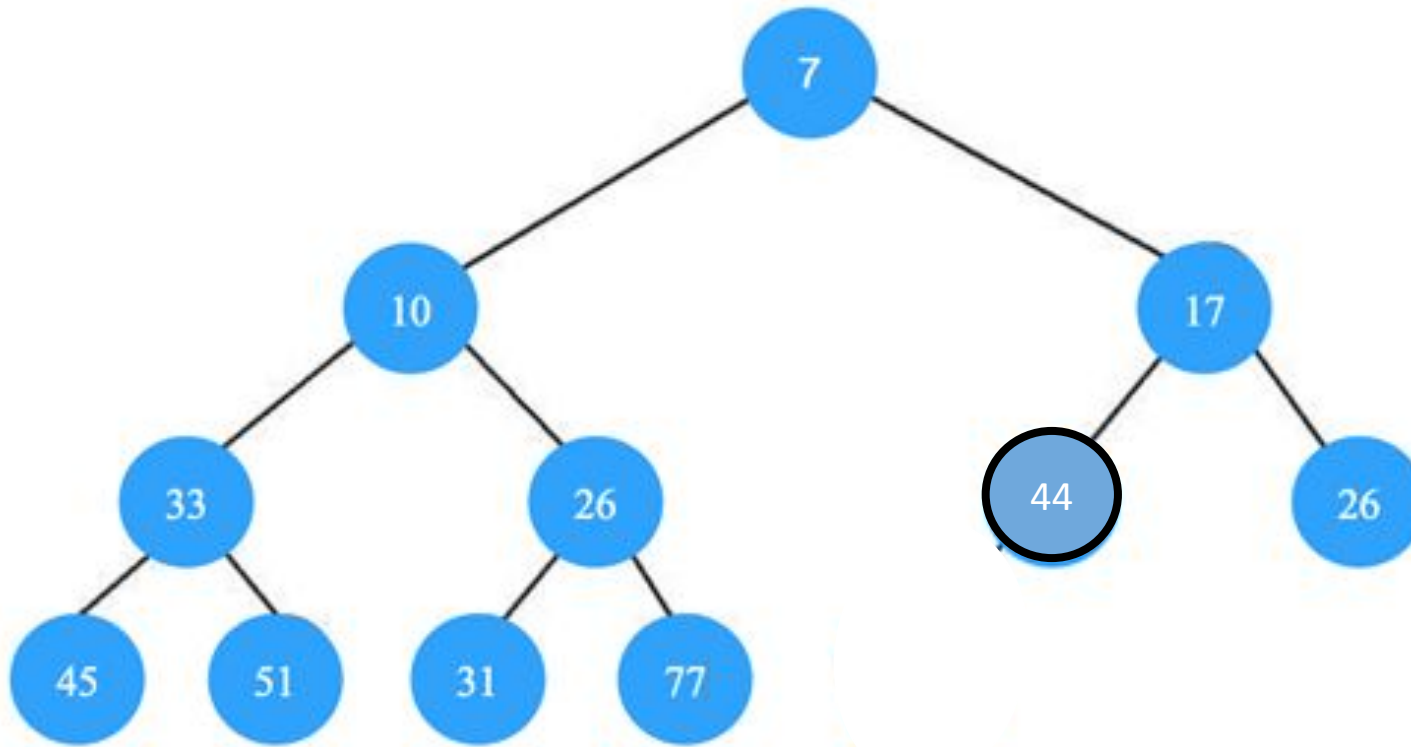3. Downheap to restore the heap property

Heap Remove: Example 1
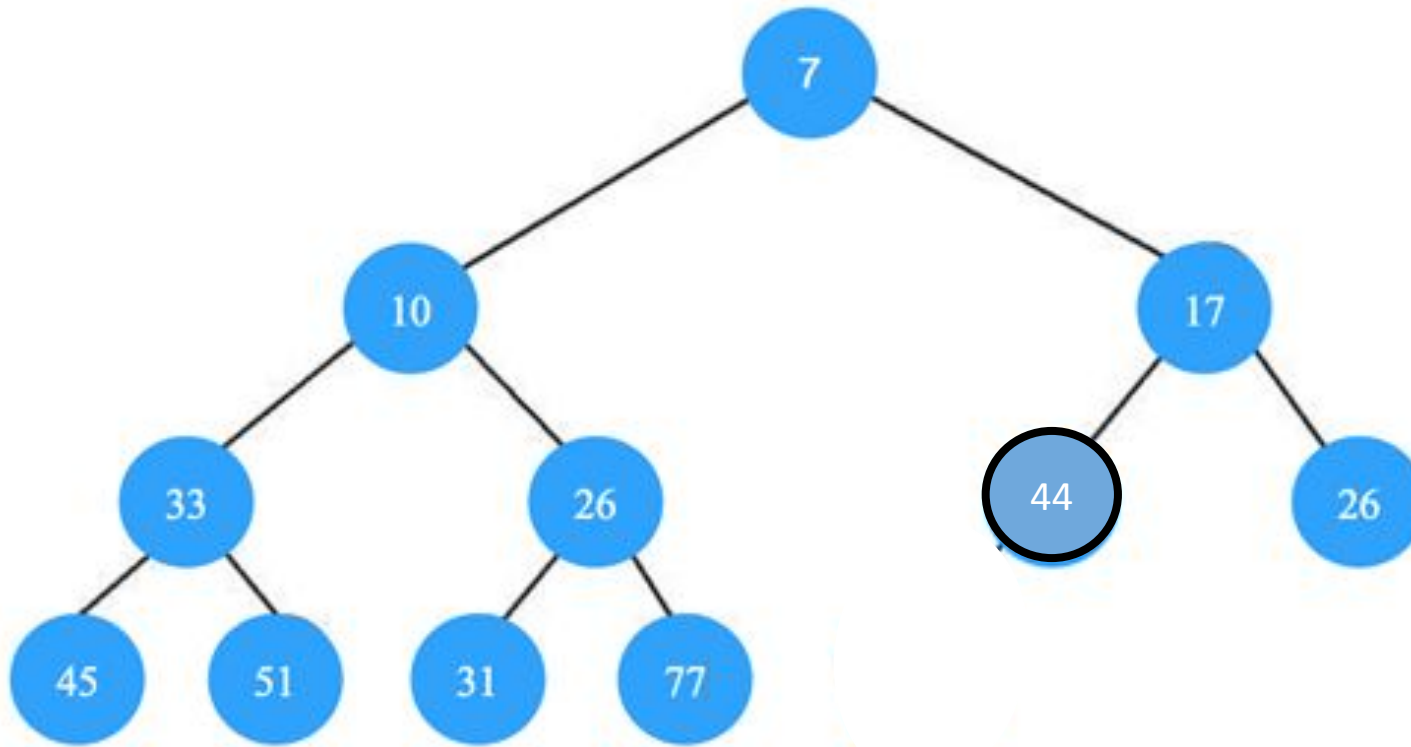

`remove(44)`

Heap Remove: Example 2

remove(23)

Heap Remove: Example 2

remove(23)

Downheap:

1. Compare the new root with its children; if they are in the correct order, stop.
2. If not, swap the element with one of its smallest children and return to the previous step
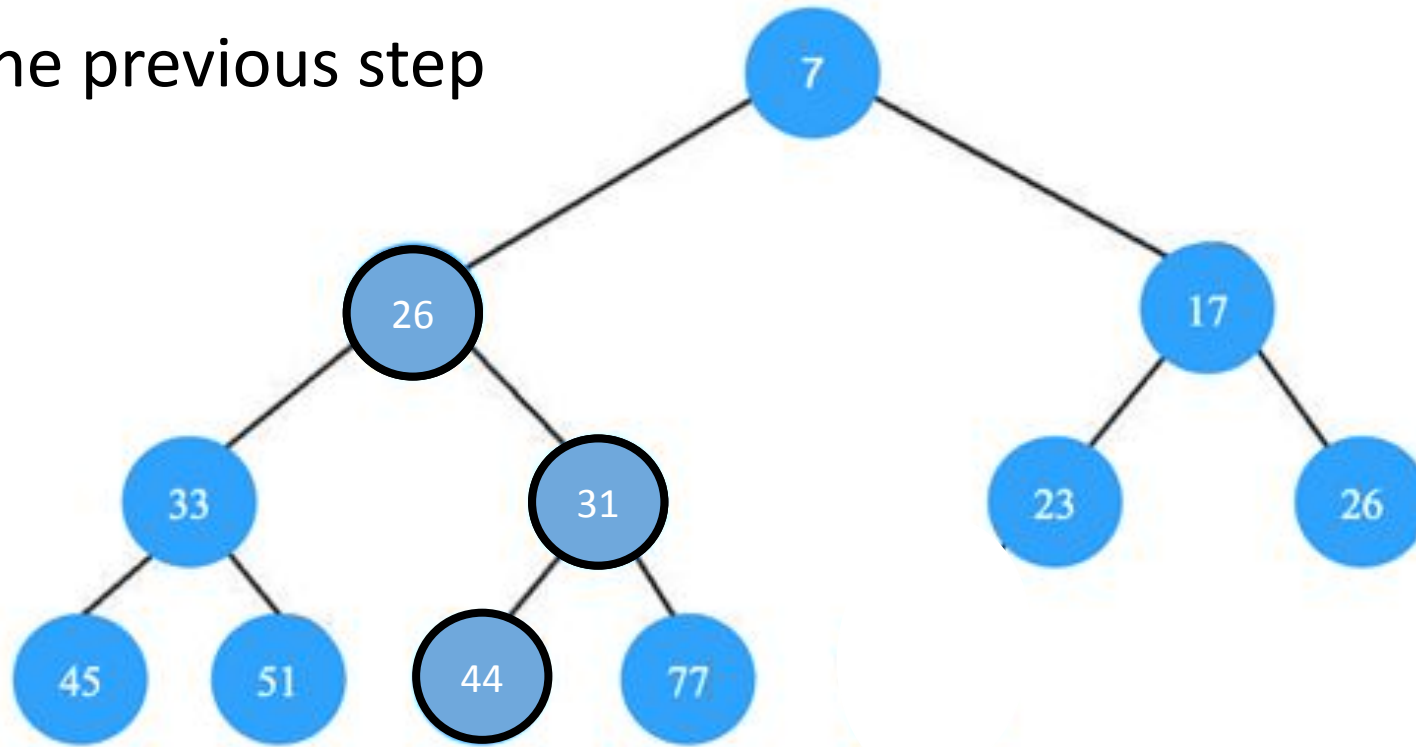
remove(10)

Heap **Remove**

More general case of poll

1. Find the index i of the element we want to delete
2. Swap this element with the last element
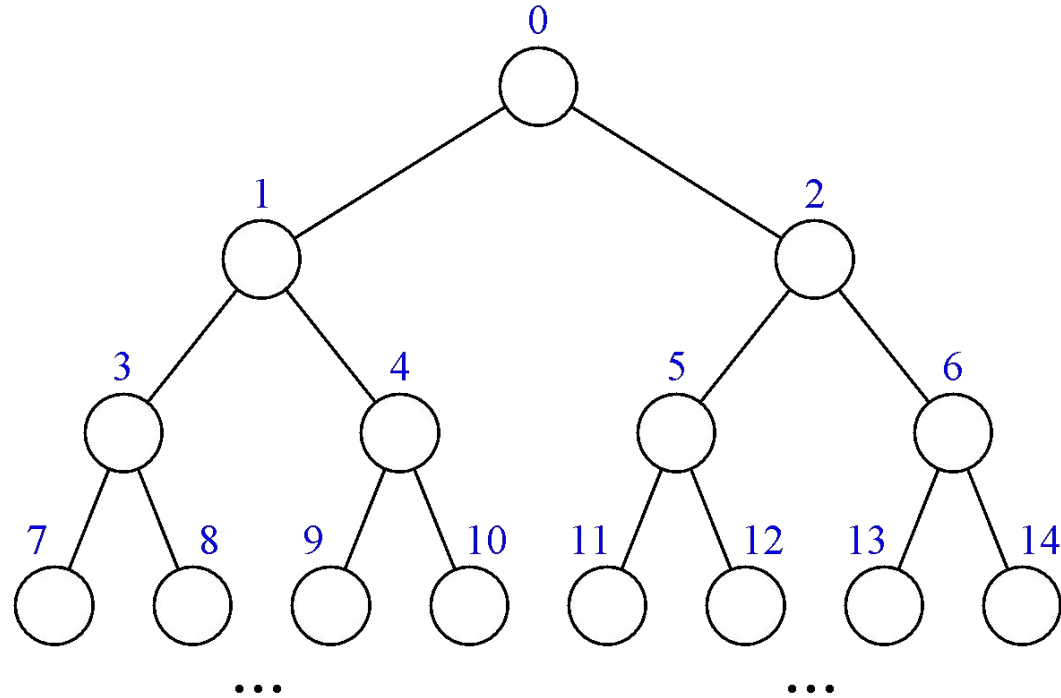3. Downheap to restore the heap property

Runtime complexity?

O(nlogn)

Heap Implementation
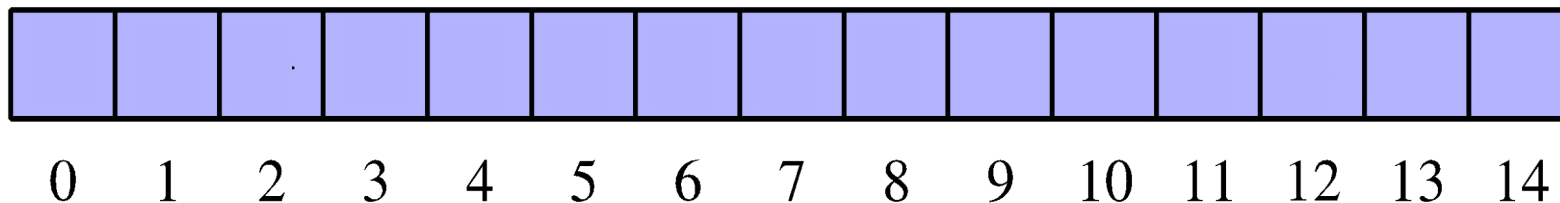
Array-based **Binary Tree** Implementation

- Number nodes level-by-level, left-to-right
- $f(root) = 0$
- $f(l) = 2f(p) + 1$
- $f(r) = 2f(p) + 2$
- Numbering is based on all positions, not just occupied positions

Array-based Binary Tree

- The numbering can then be used as indices for storing the nodes directly in an array
- $f(root) = 0$
- $f(l) = 2f(p) + 1$
- $f(r) = 2f(p) + 2$
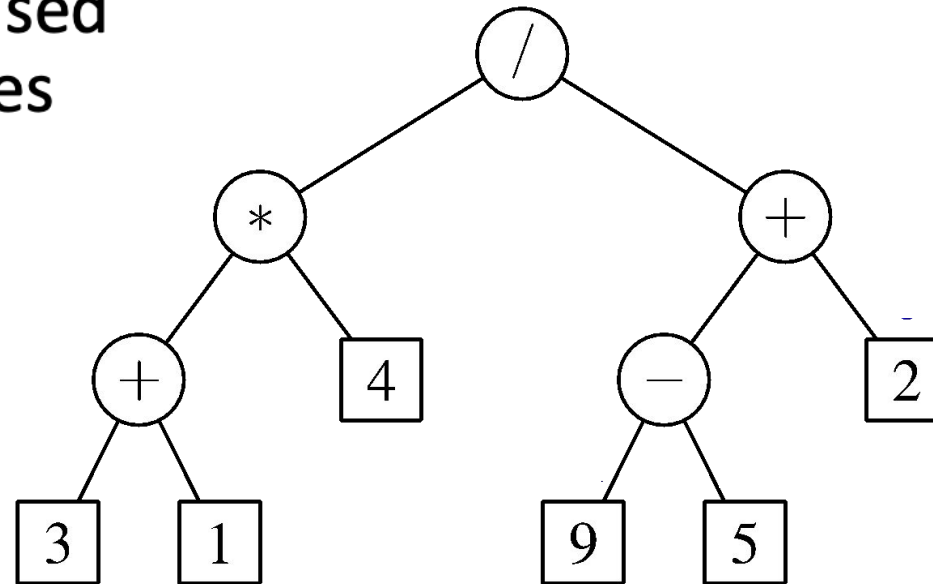
02/28/24   CS151 - Lecture 12 - Spring '25

Array-based Binary Tree

- The numbering can then be used as indices for storing the nodes directly in an array
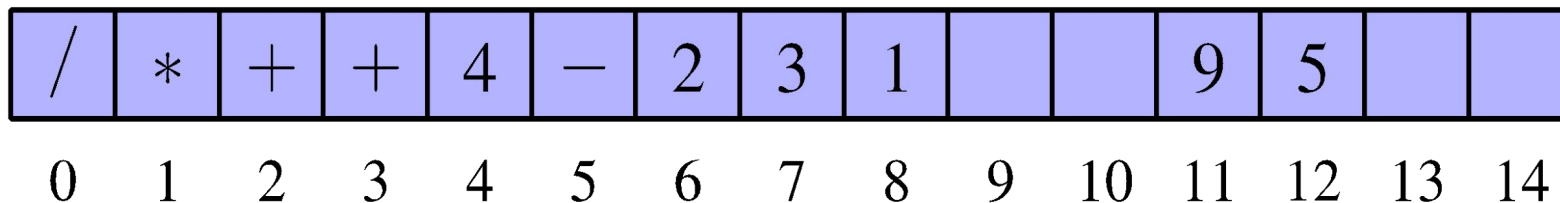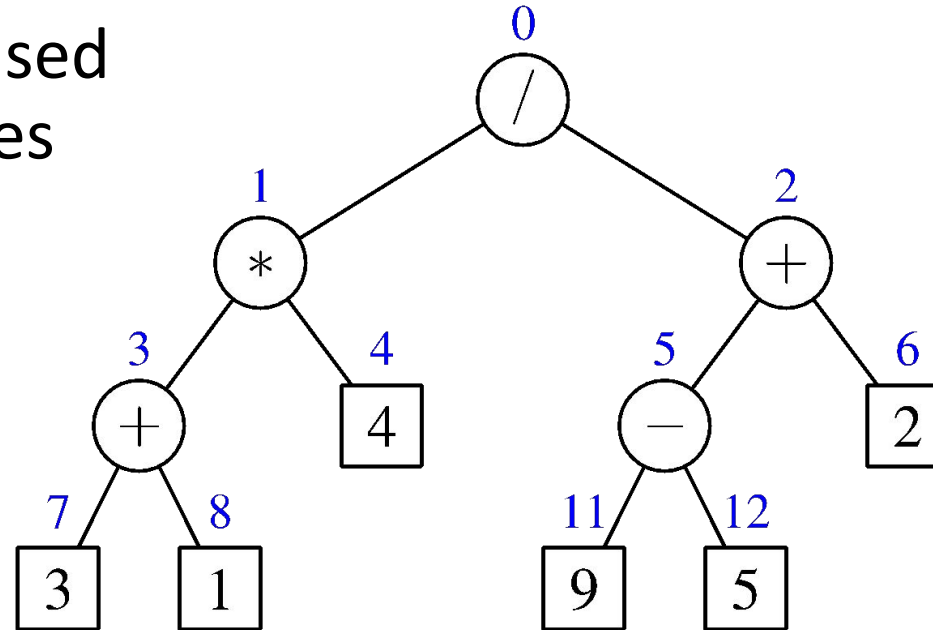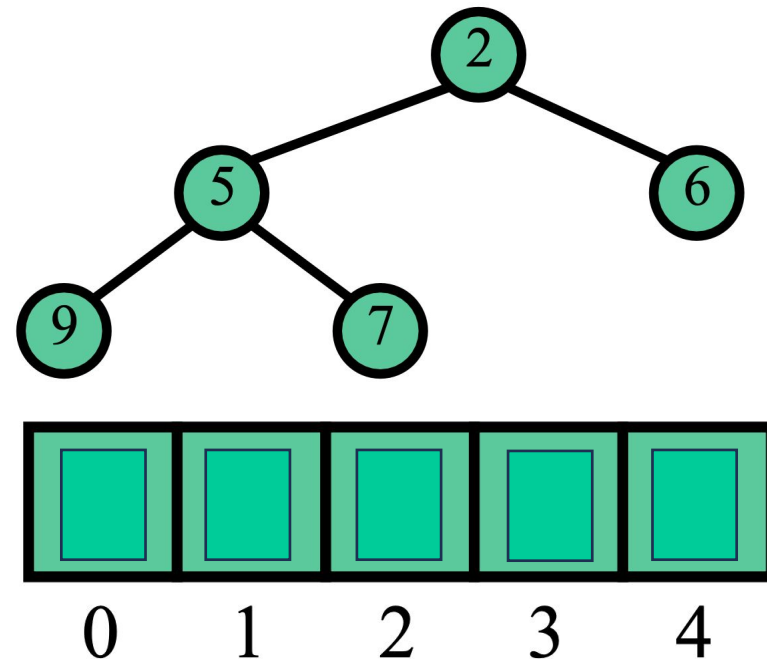
Array based heap

Node at index i
- Left child index:
  - 2i + 1
- Right child index:
  - 2i + 2

Array based heap

Node at index i
- Left child index:
  - 2i + 1
- Right child index:
  - 2i + 2

- Peek:
  - Get element at index 0

- Poll:
  - Remove element at index 0

- No need to store references/links
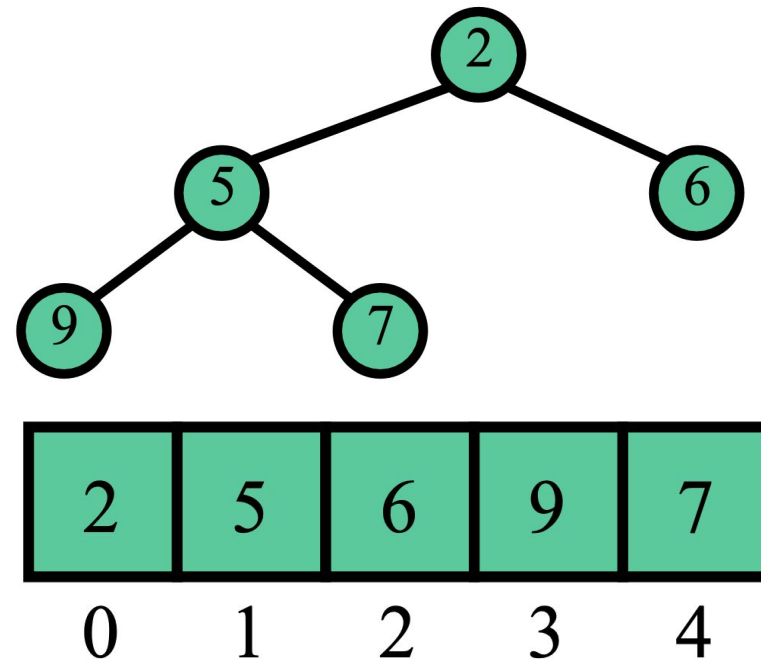
Selection Sort

Selection Sort

In place sorting algorithm

1. Separate the array into "sorted" and "unsorted"
   a. sorted starts empty
2. Find the min element in the unsorted array
3. Swap min with the first element in unsorted
4. repeat

Selection sort

code

Runtime complexity?
    O(n^2)

Space complexity?

    O(1)

Heap Sort

Heap Sort

- "selection sort with the correct data structure"

- divides input into a sorted and an unsorted region
- iteratively shrinks the unsorted region by extracting the min element from it and inserting it into the sorted region

Heap Sort

- "selection sort with the correct data structure"

Expensive portion of selection sort?

Heap sort: instead of looping over the unsorted portion of the array, store the unsorted data in a heap!

Now the min is always at the top.

Runtime complexity of poll? O(logn)

Heap Sort

1. Heap construction
    a. rearrange the array into a heap
2. Heap extraction
    a. iteratively **poll** and insert into the sorted portion

Heap Sort - Example

[64, 25, 12, 22, 11]

Heap Sort

1. Heap Construction Phase:
   a. runtime complexity?

2. Heap Extraction Phase:
   b. runtime complexity?

Overall Runtime Complexity?

Space complexity?

Summary

1. Heaps
   a. DS for efficient removal of the min element
   b. Can be implemented as an array or node based
   c. logarithmic insertion, poll
   d. linear search
   e. log-linear removal
2. Selection Sort
   a. O(n^2) sort algorithm
3. Heap Sort
   a. O(nlogn) sort algorithm