CS151 Intro to Data Structures

Heaps & Priority Queues

Announcements

- Midterm grades coming
- **QUIZ NEXT WEEK (Wednesday 3/27)**
  - Will add points to your midterm exam
- HW05 was due last night
  - you have 2 more late days to submit
- If you re-did an assignment over spring break it's due by 4pm today. Let me know and i'll open the gradescope

Outline

1. Comparables review for your HW and lab
2. Heaps
    a. Definition
    b. Operations
    c. Implementing a heap
3. Selection Sort
4. Heapsort

Comparable and Generics

What do these mean?

1. `public interface BinaryTree<E extends Comparable<E>>`
   - The elements of the tree are comparable to eachother

2. `public class LinkedBinaryTree<E extends Comparable<E>> implements BinaryTree<E>`
   - LinkedBinaryTree has all the methods specified in Binary Tree and the elements are comparable to eachother

Comparable and Generics

1. `public class LinkedBinaryTree<E extends Comparable<E>> implements Comparable<LinkedBinaryTree<E>>, BinaryTree<E>`

   - LinkedBinaryTree is comparable to other LinkedBinaryTrees, the elements of the tree are comparable to eachother, and LinkedBinaryTree has all the methods specified in BinaryTree

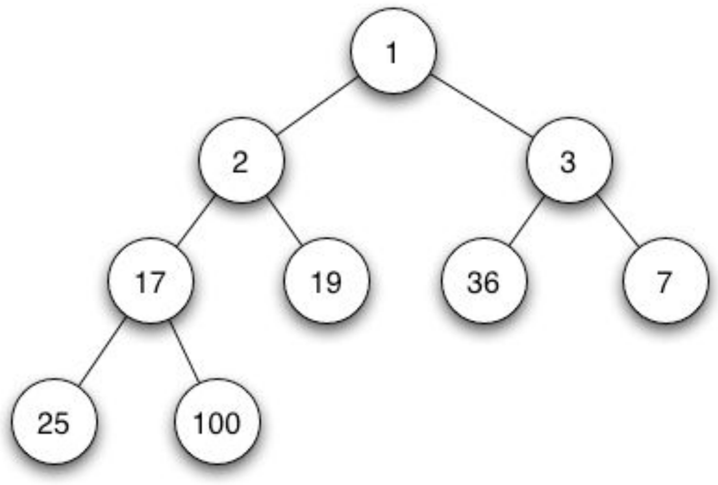
Heaps

Binary Heap Properties

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

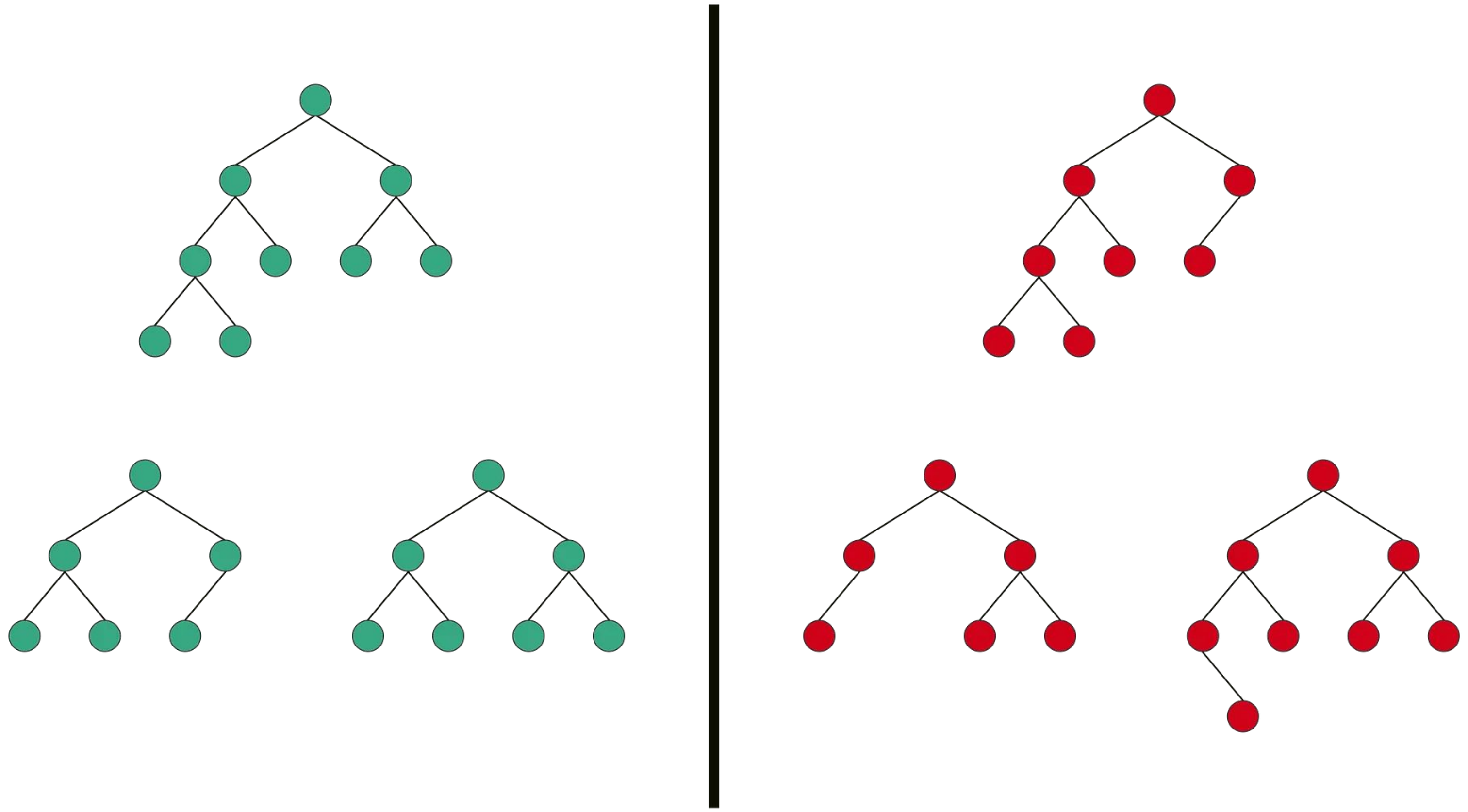
2. For every node n (except for the root): **n.key >= parent(n).key**

3. **Complete:** 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

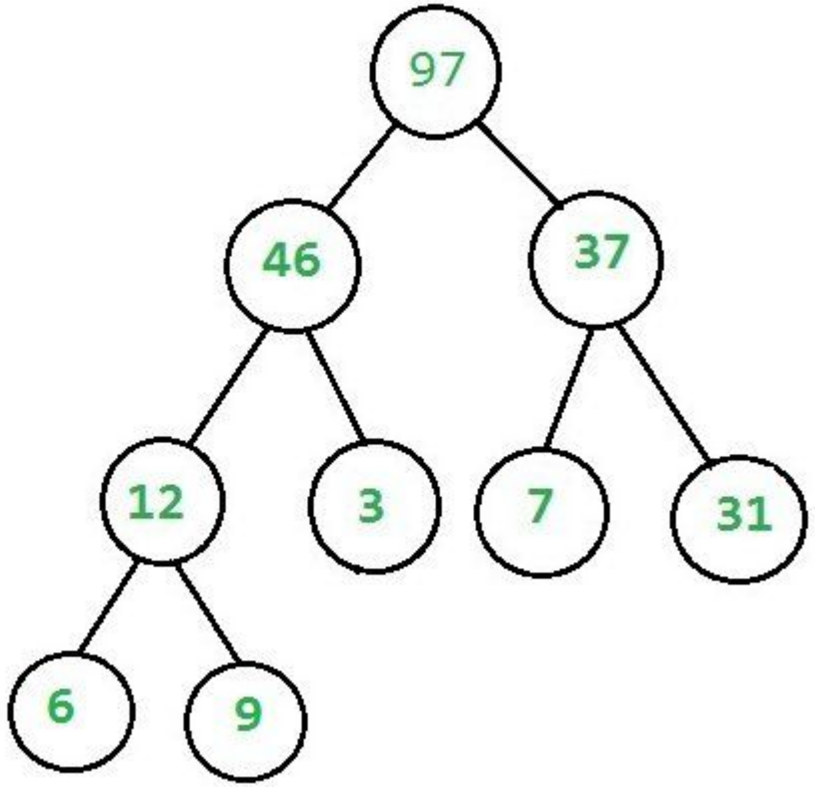


Complete Binary Trees

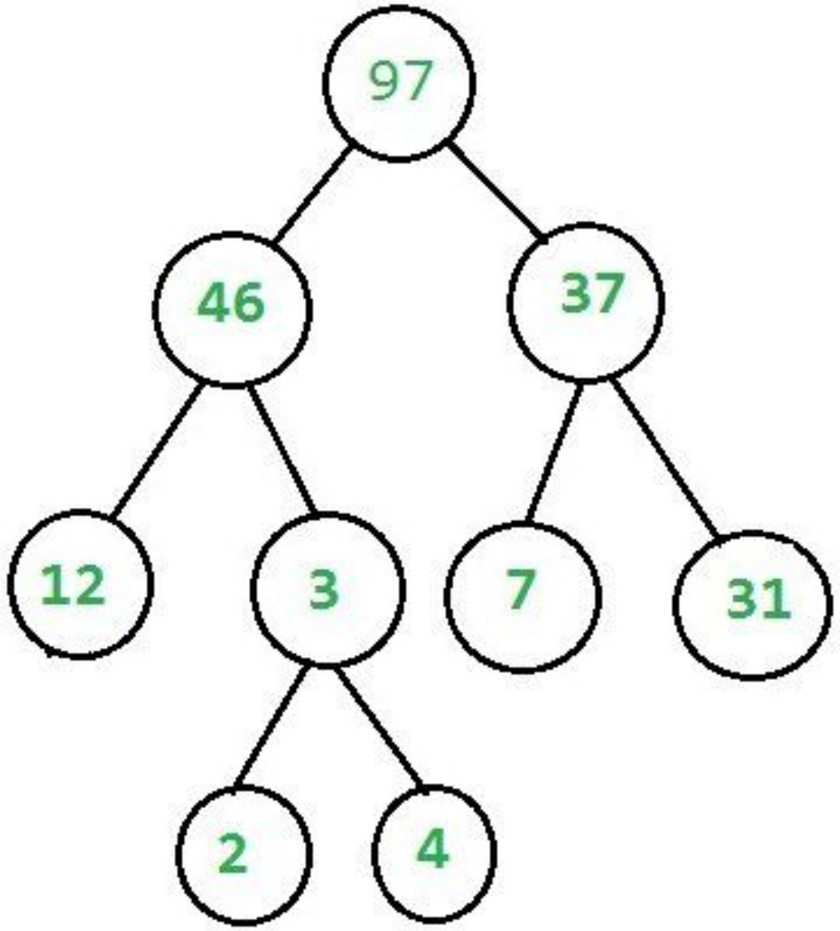
All levels of the tree, except possibly the last one are fully filled, and, if the last level of the tree is not complete, the nodes of that level are filled from left to right



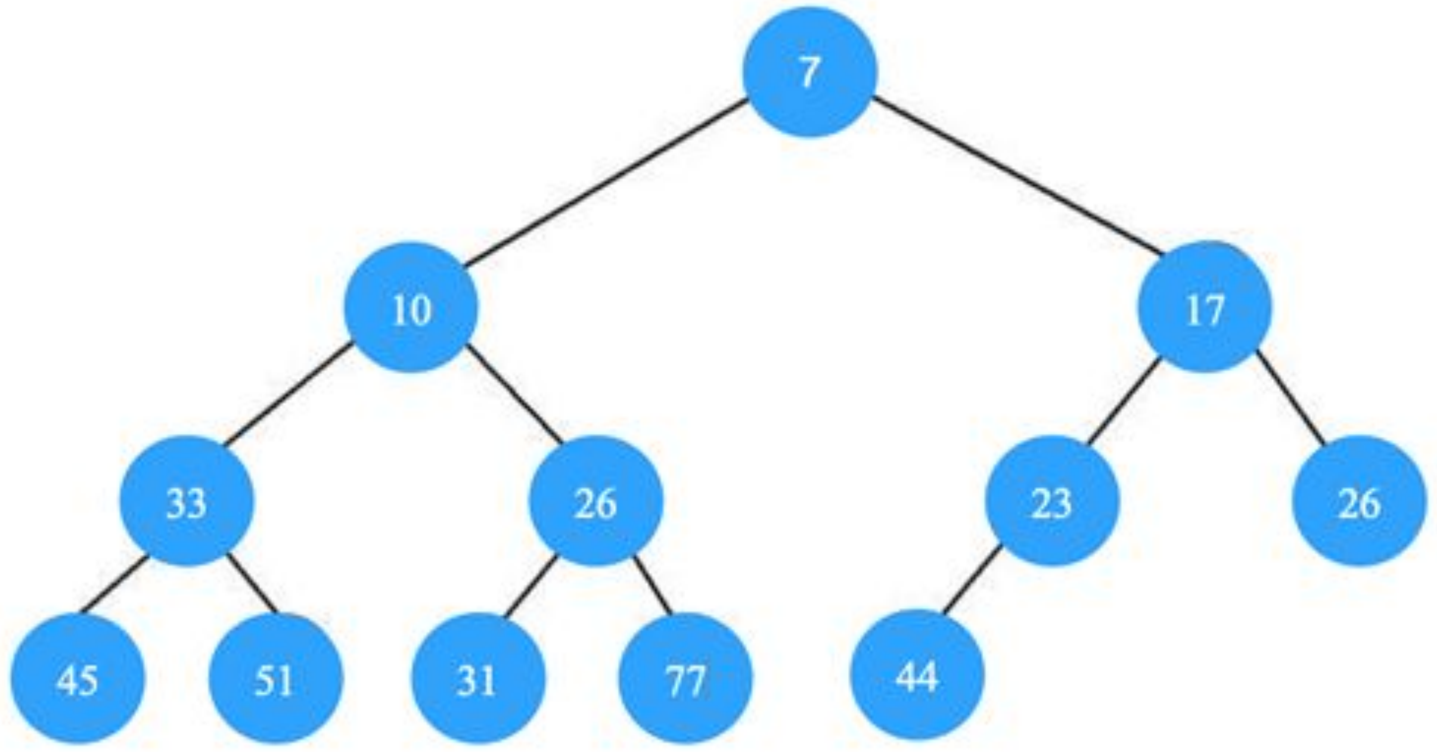
Is this a binary heap?



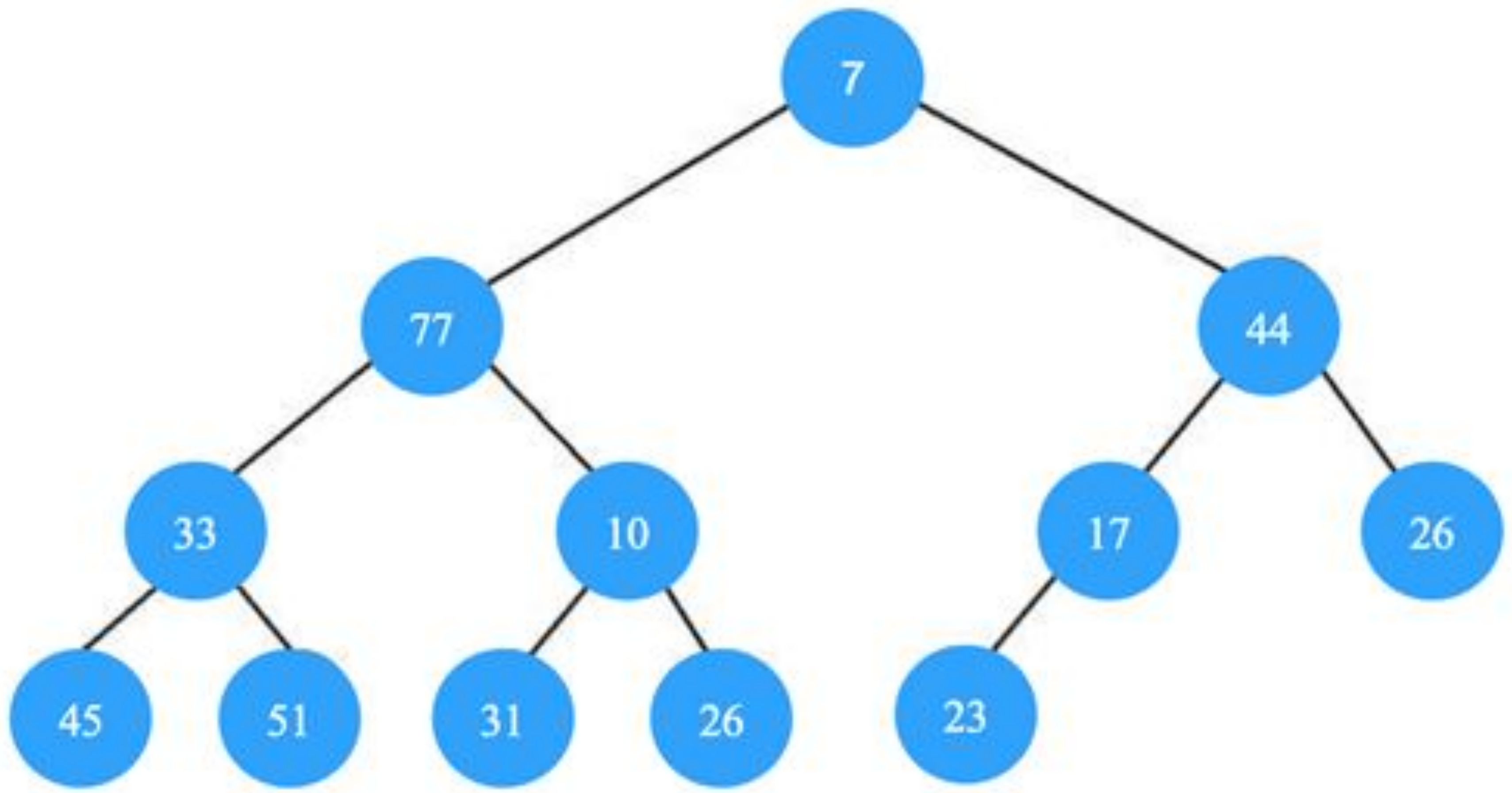
Is this a binary heap?



Is this a binary heap?

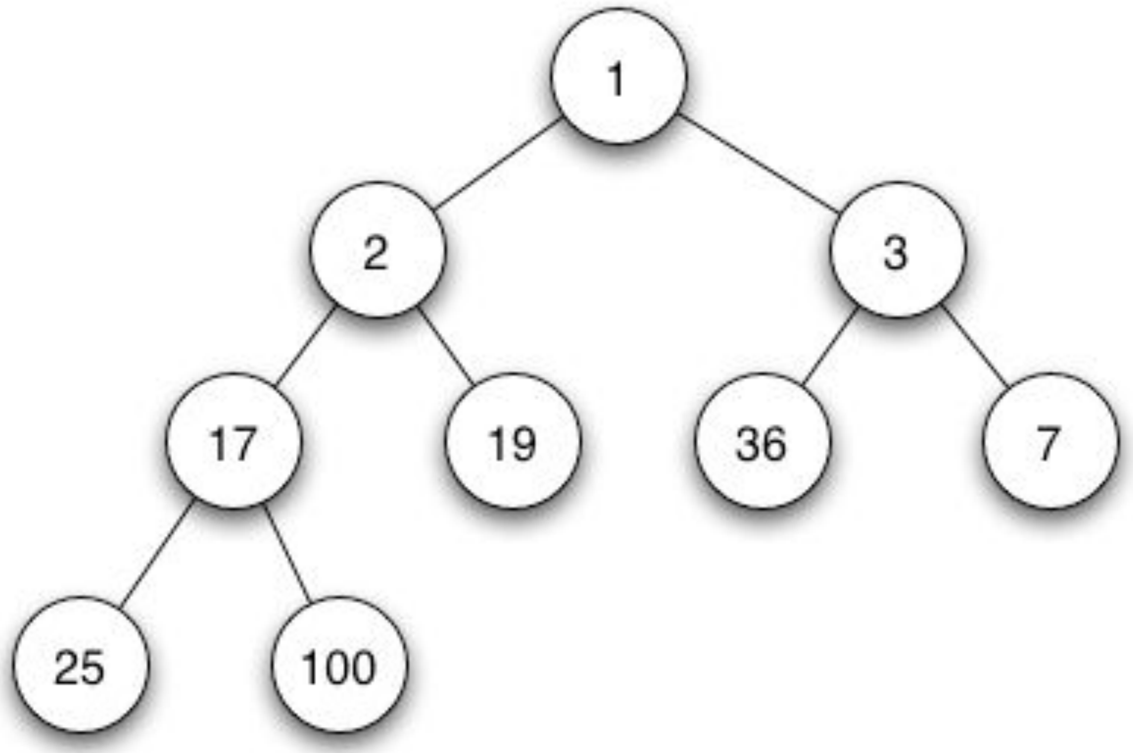


Is this a binary heap?



Heap Properties

- Height?
  - log(n)



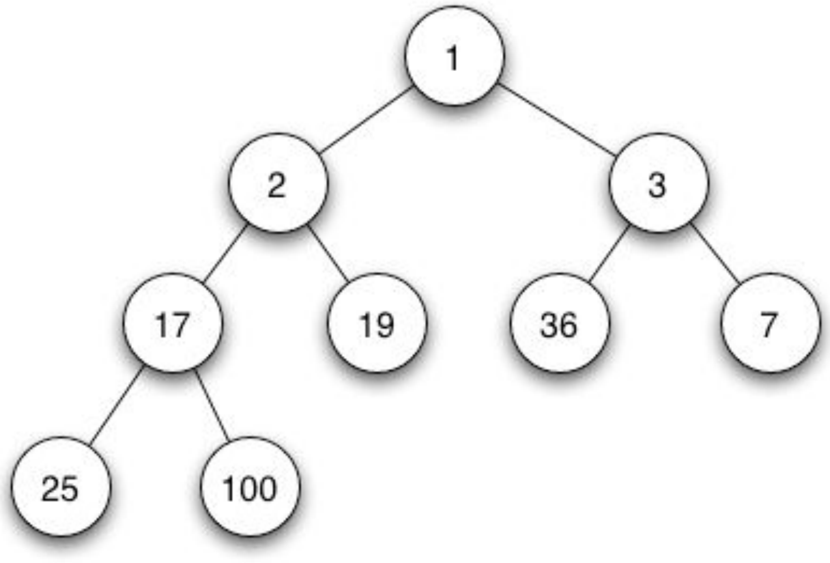
Heap Operations:

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**



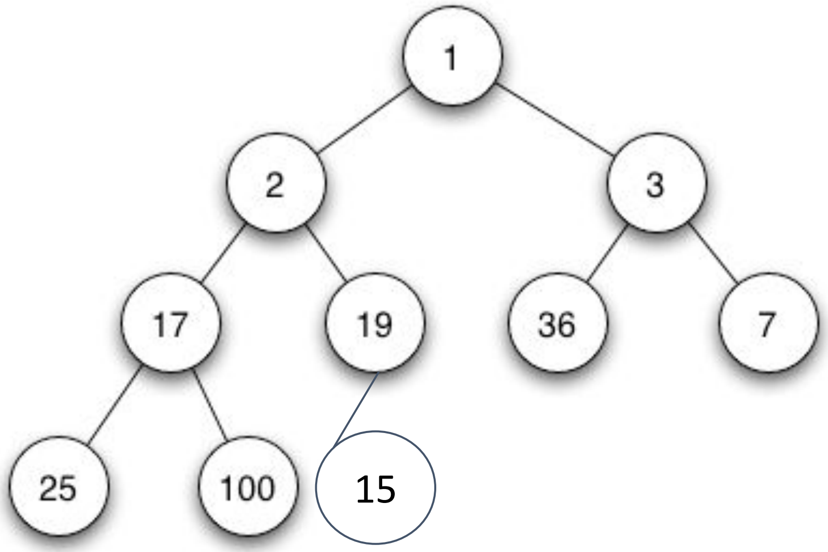
`insert(15)`

Heap Insertion

Need to maintain:

UPHEAP!

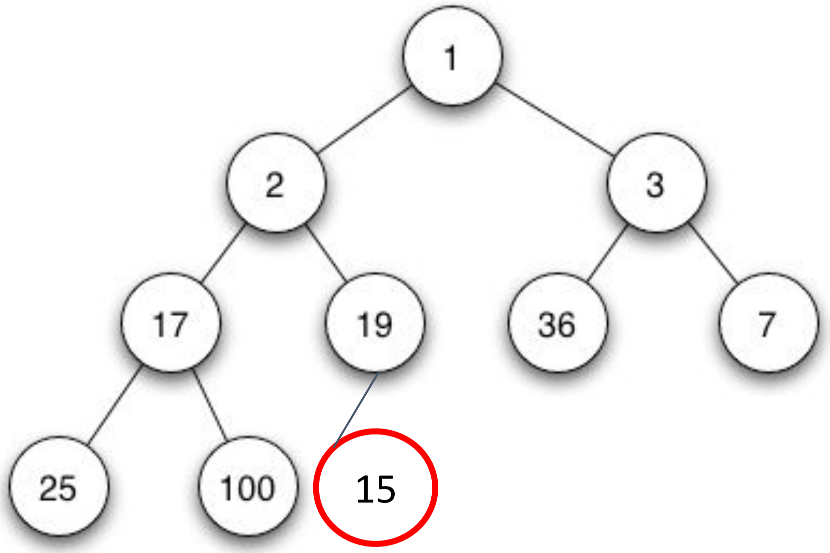
1. completeness
2. key order property: **n.key >= parent(n).key**

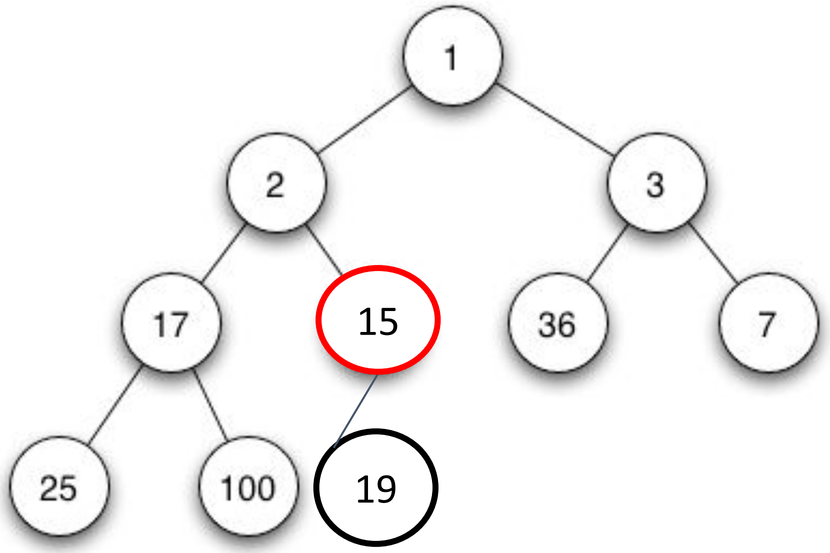


```
insert(15)
```

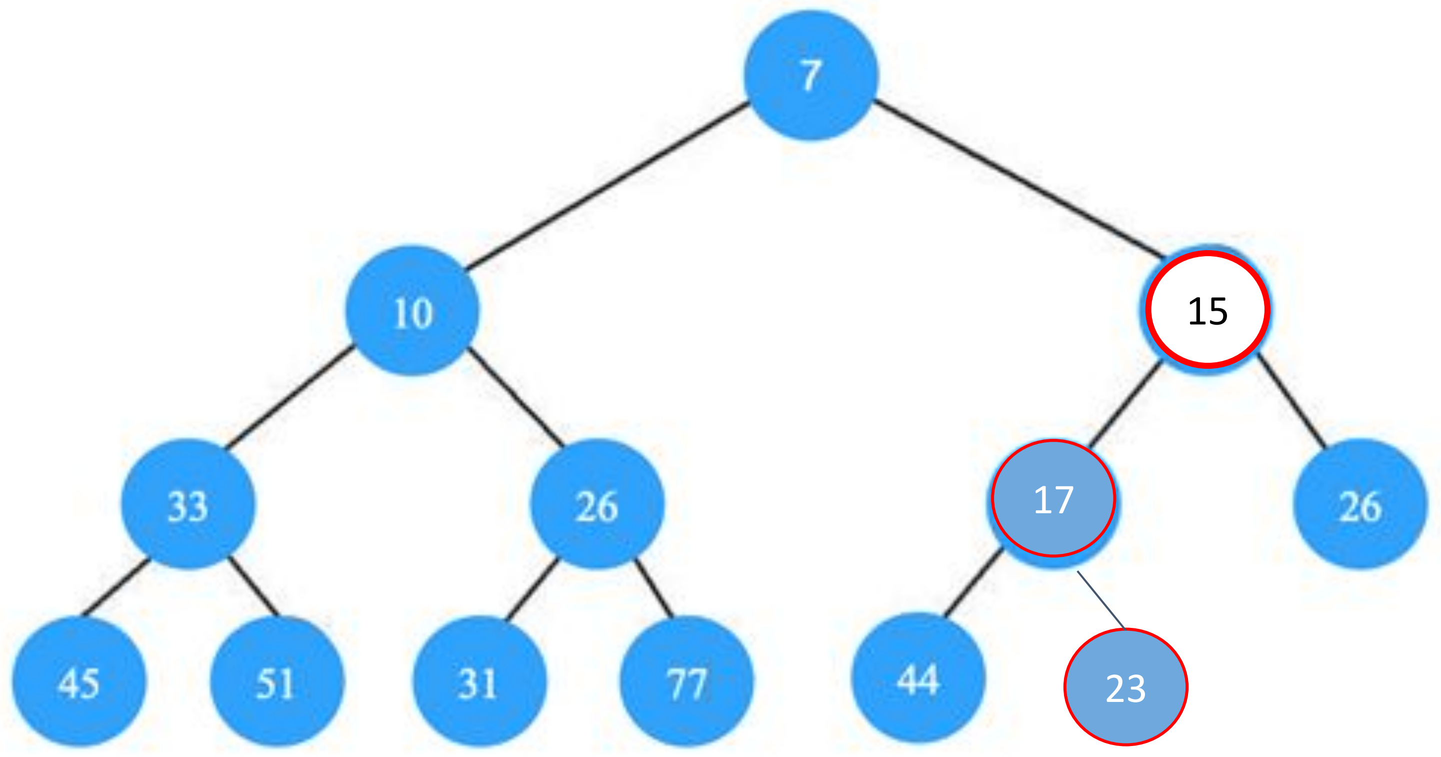
Heap Insertion - Upheap

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.

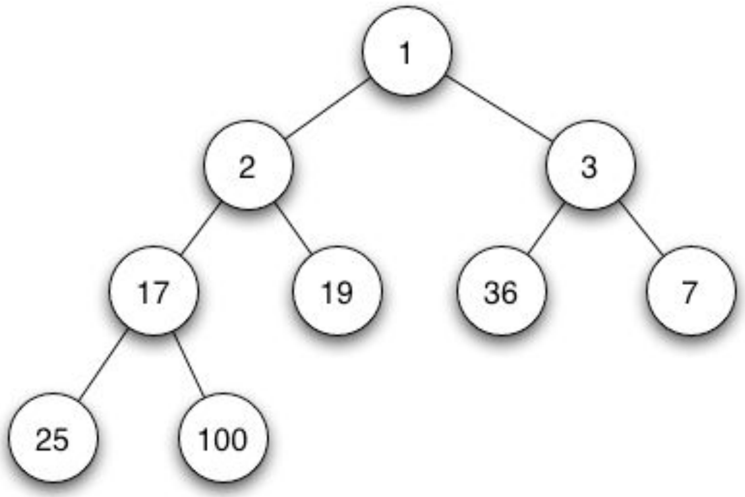
Runtime complexity?

O(logn)

* assuming we know where the leftmost open slot is and expansion is not necessary (array implementation)

Heap **Poll**

- Removing the root
- Also called extract



What properties do we need to maintain?

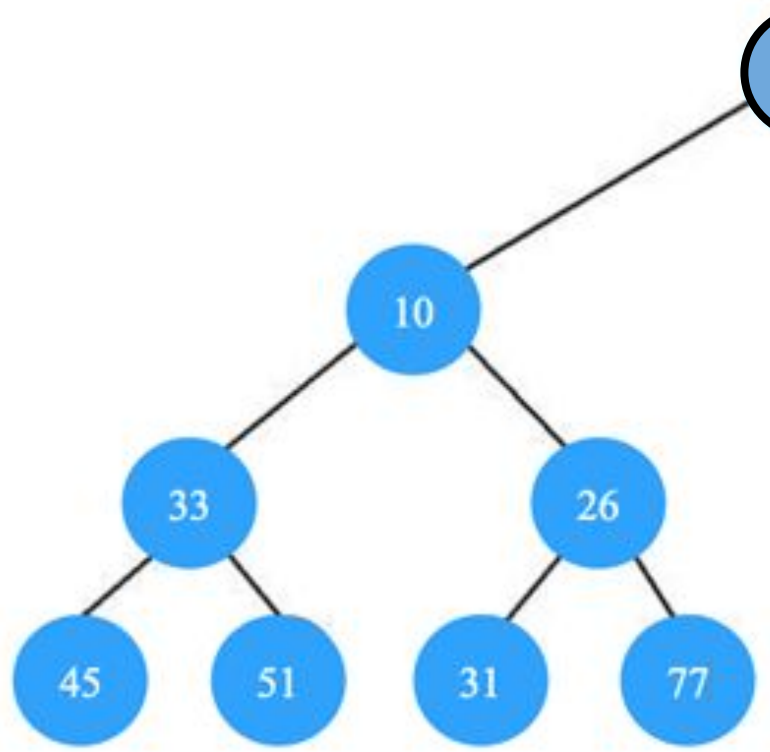
1. completeness
2. key order property: **n.key >= parent(n).key**

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

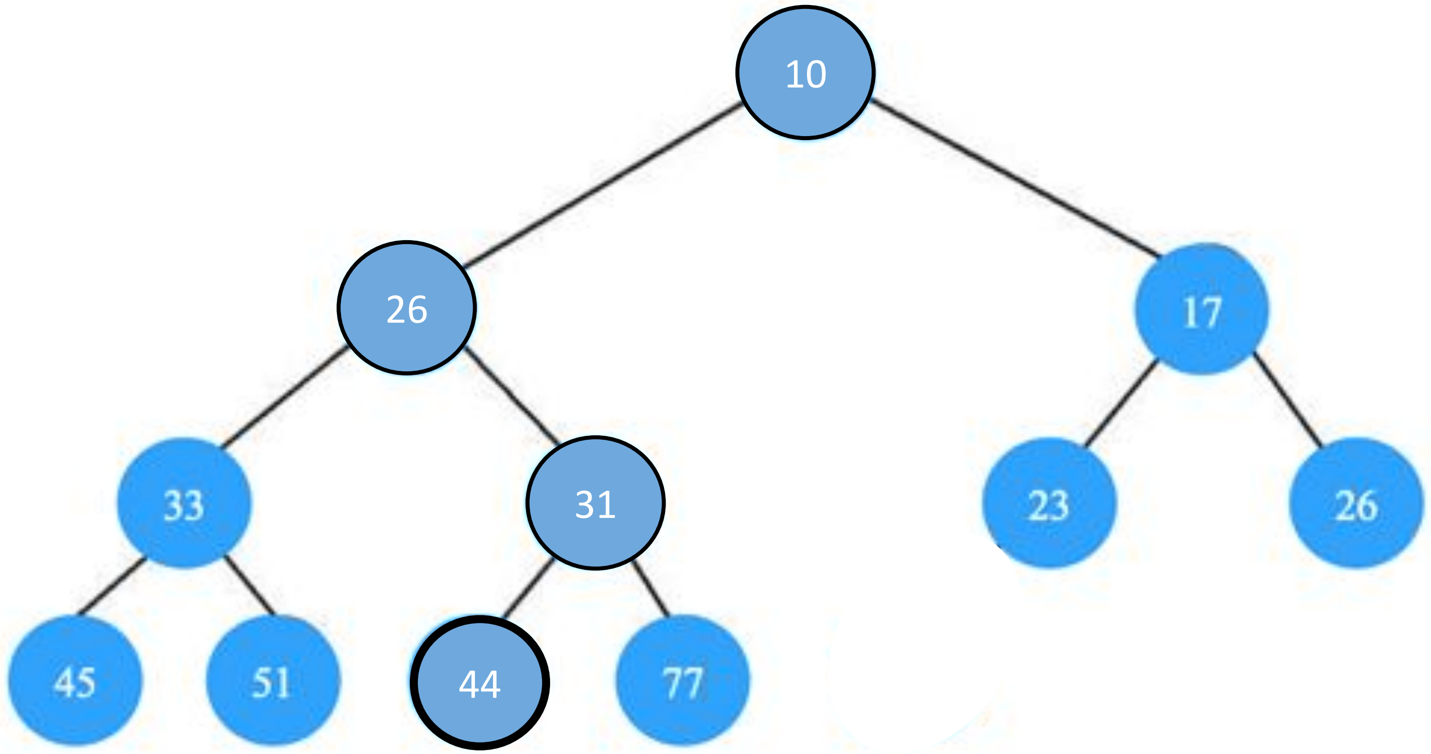
Heap Poll Example

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

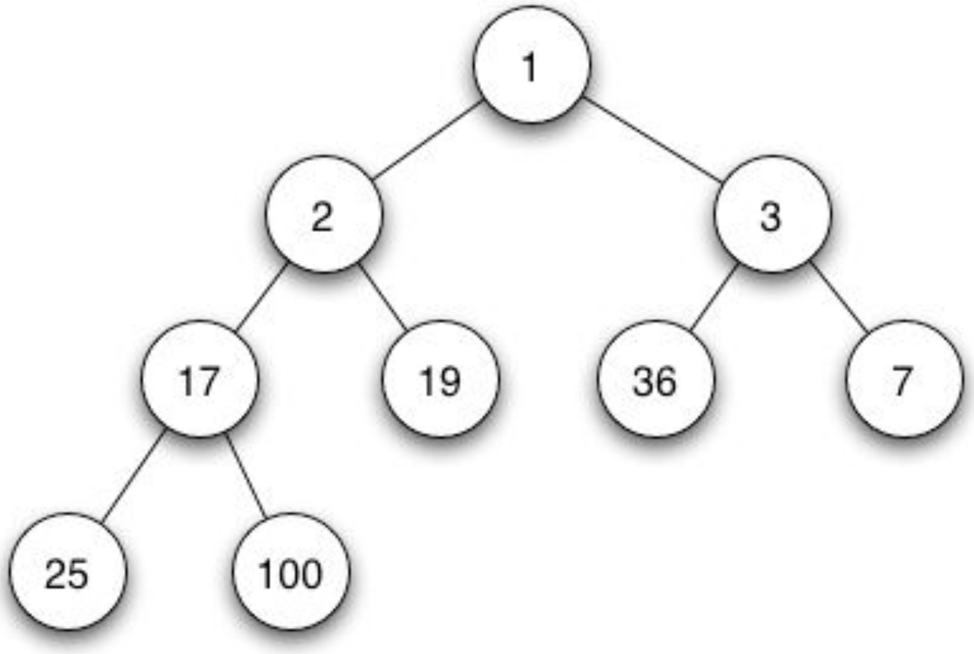
Runtime?

O(logn)

assuming... don’t need to expand and we know where the last elem is

Heap Search

- Is the element 36 in the heap?
- Runtime complexity?
    - Best case?
    - Worst case?



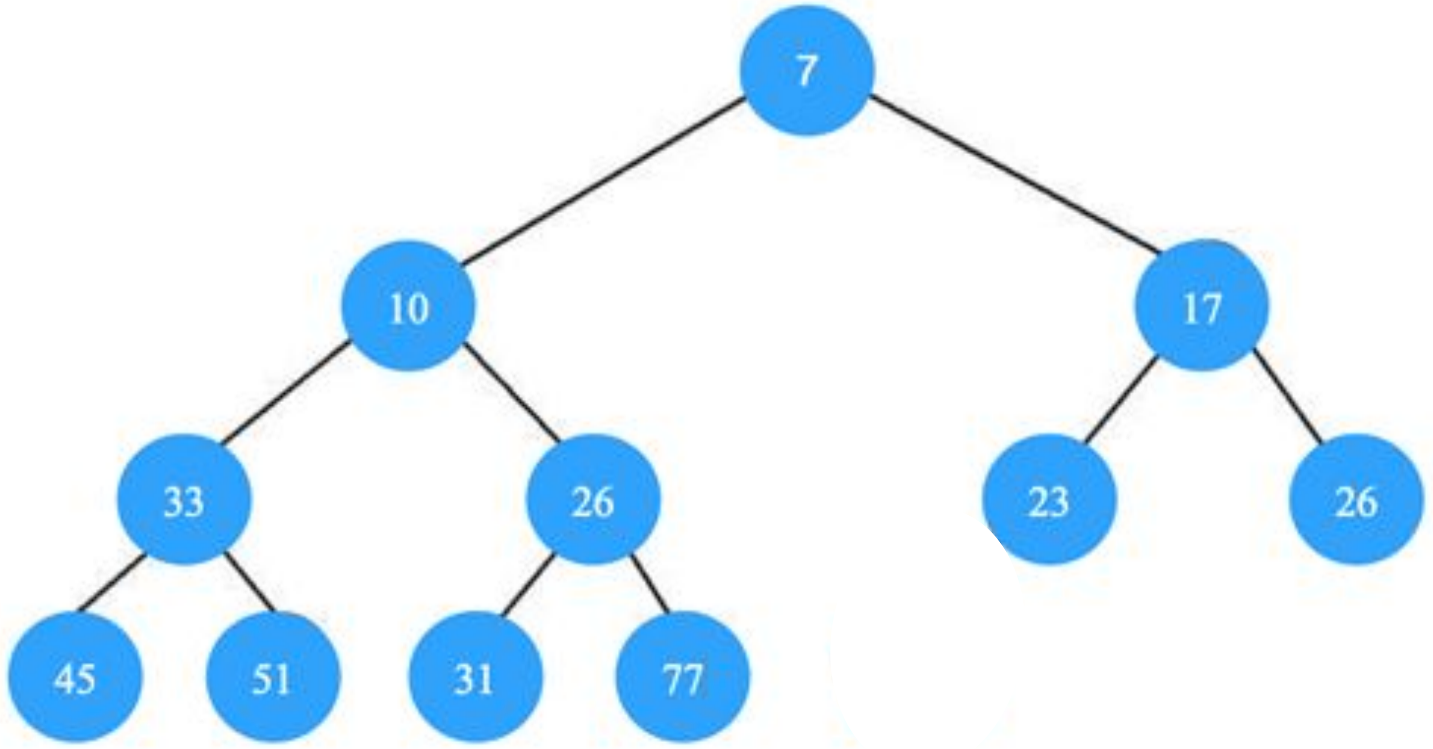
Heap **Remove**

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

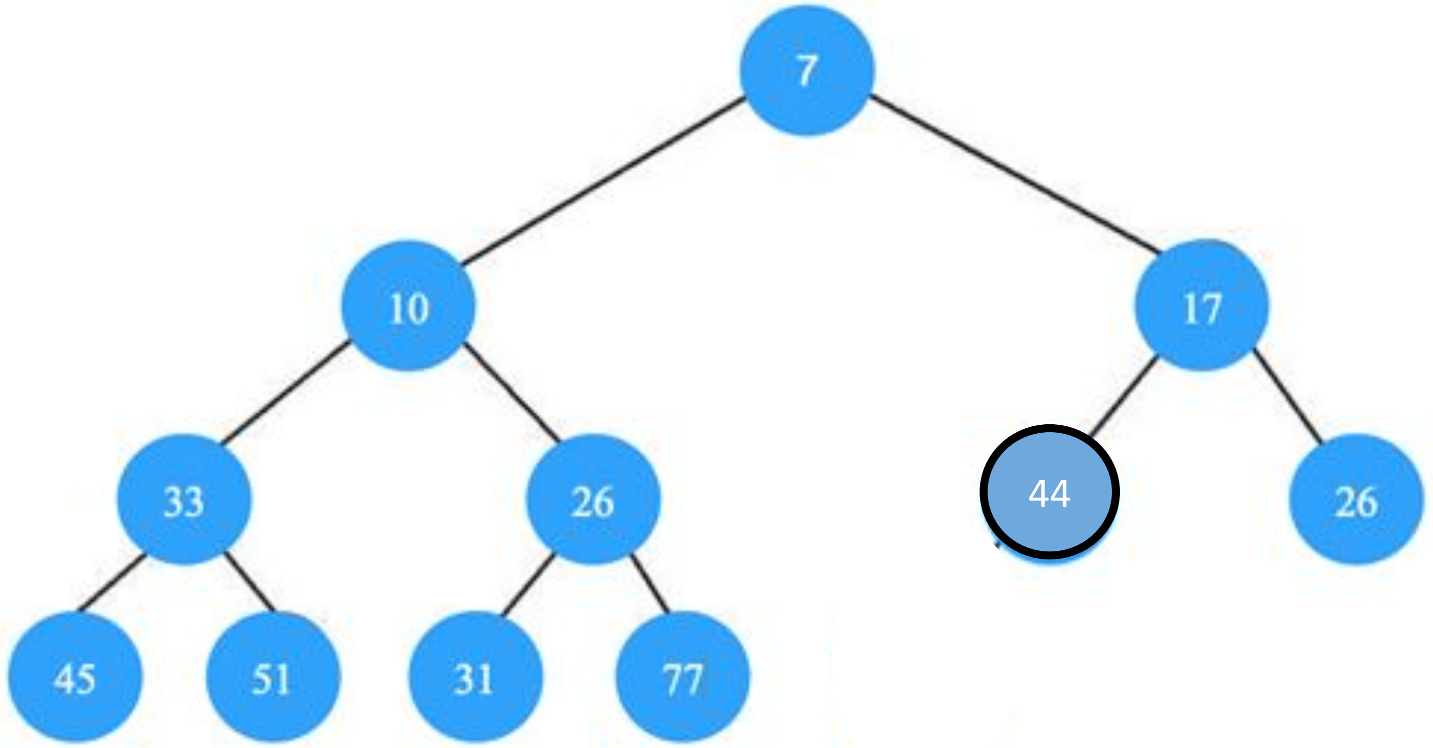
Heap Remove: Example 1

`remove(44)`



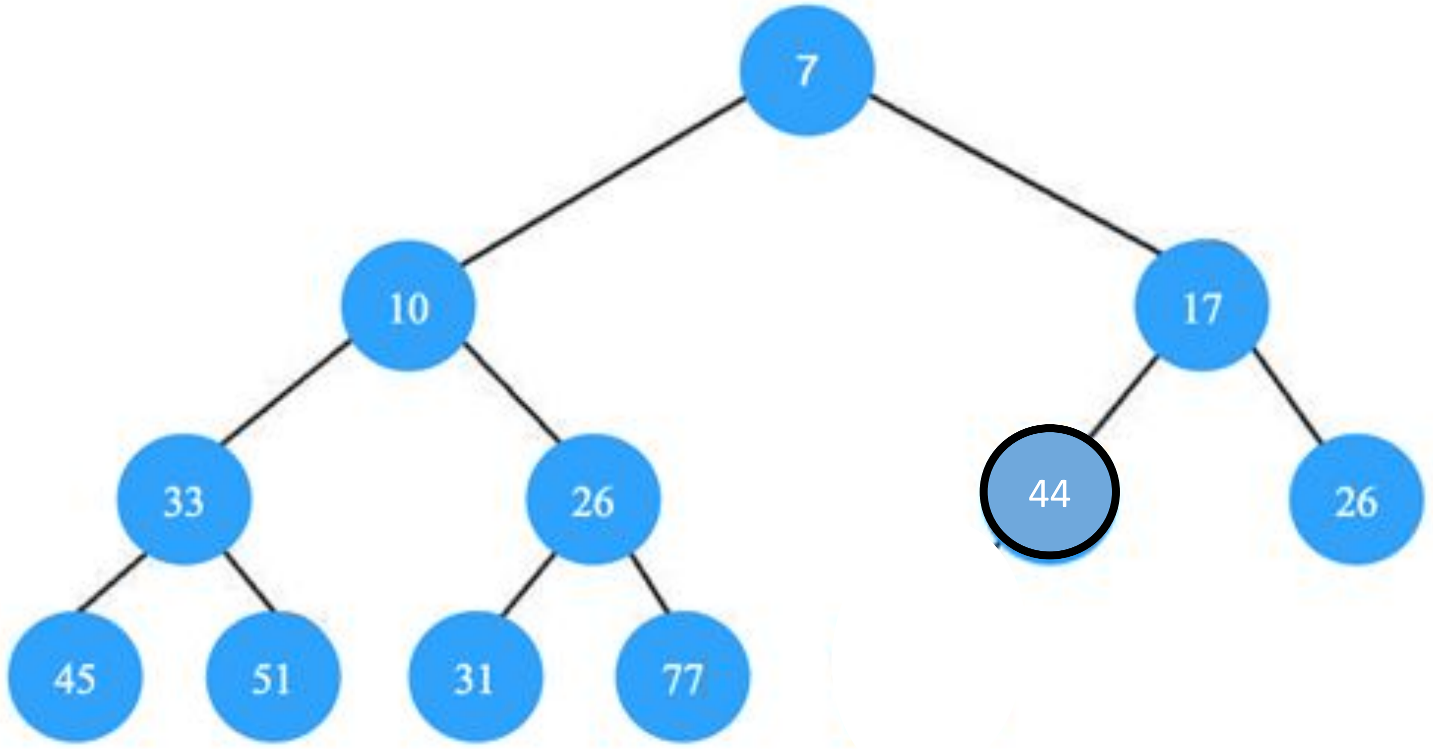
Heap Remove: Example 2

`remove(23)`



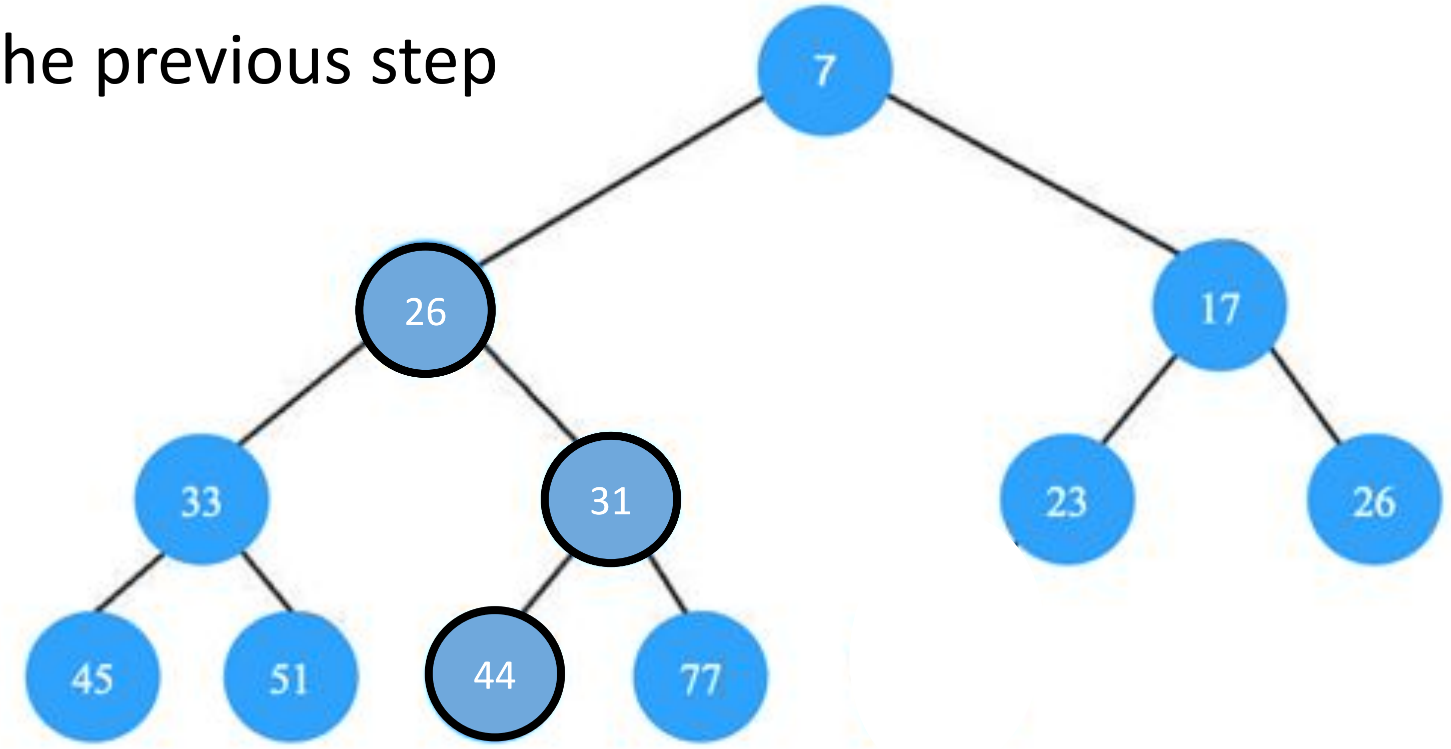
Heap Remove: Example 2

`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

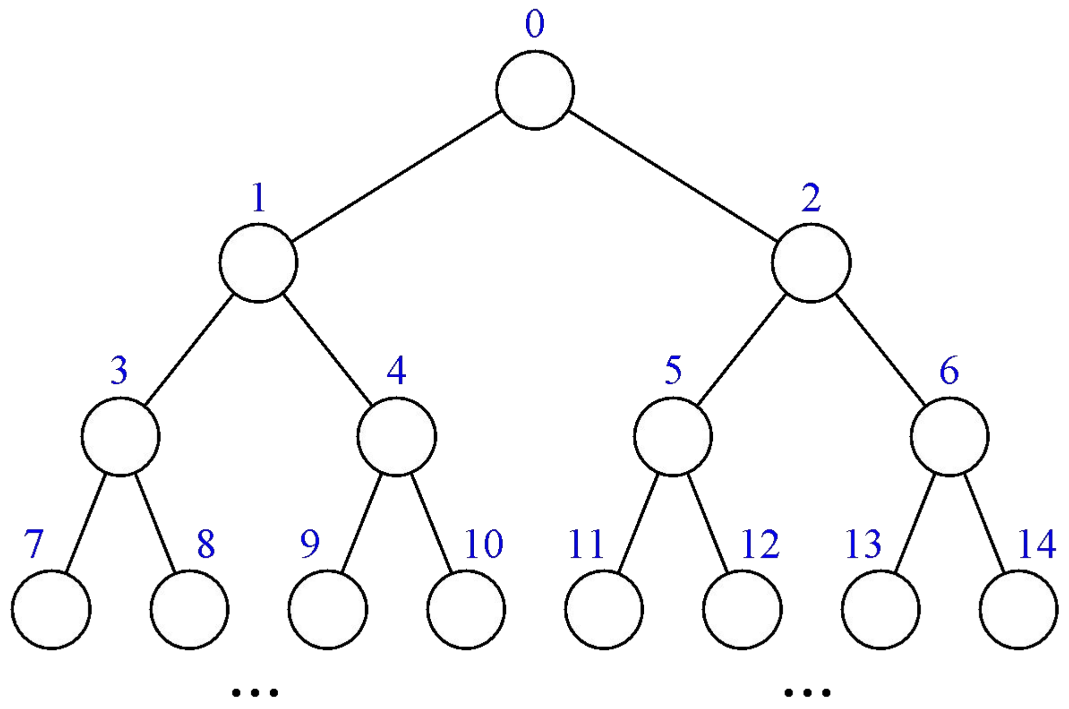
Runtime complexity?

O(nlogn)

Heap Implementation

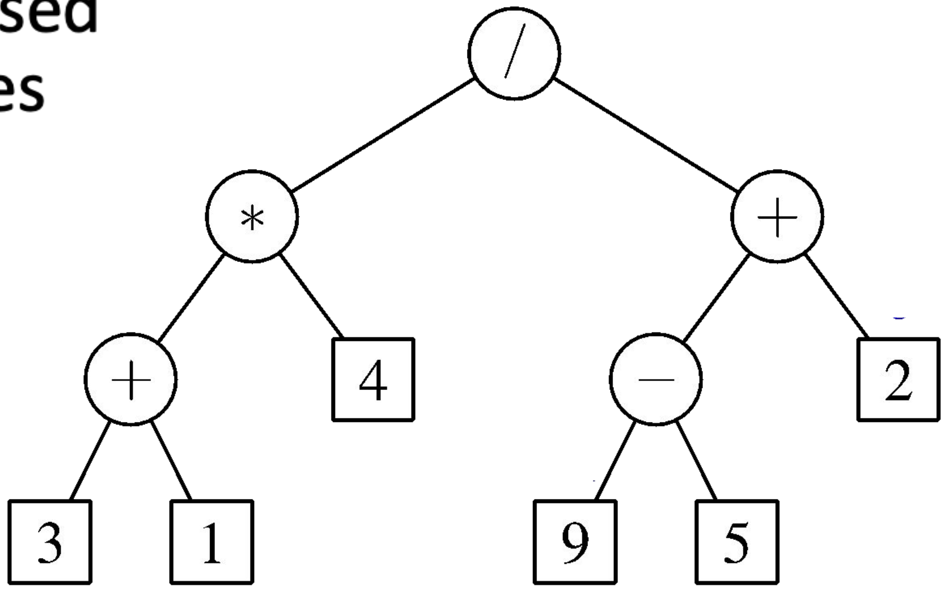
Array-based **Binary Tree** Implementation

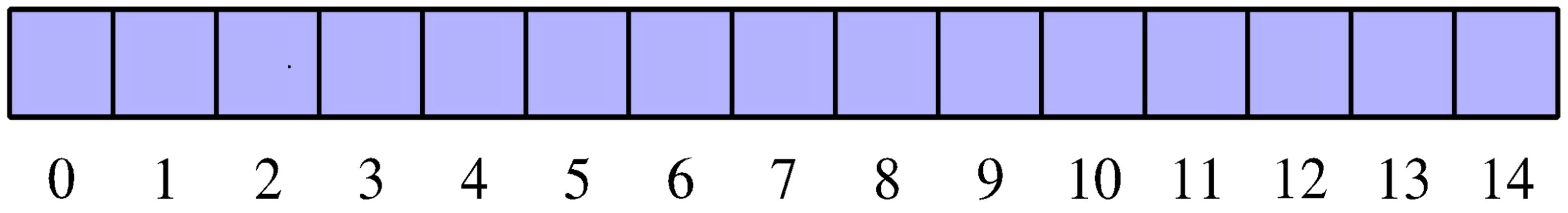
- Number nodes level-by-level, left-to-right
- $f(root) = 0$
- $f(l) = 2f(p) + 1$
- $f(r) = 2f(p) + 2$
- Numbering is based on all positions, not just occupied positions



Array-based Binary Tree

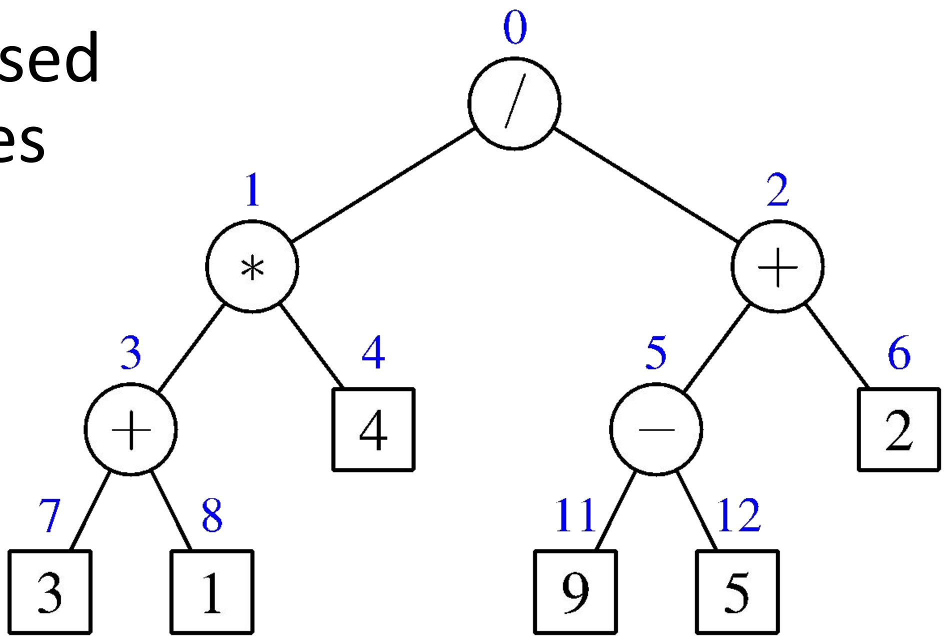
- The numbering can then be used as indices for storing the nodes directly in an array
- $f(root) = 0$
- $f(l) = 2f(p) + 1$
- $f(r) = 2f(p) + 2$





Array-based Binary Tree

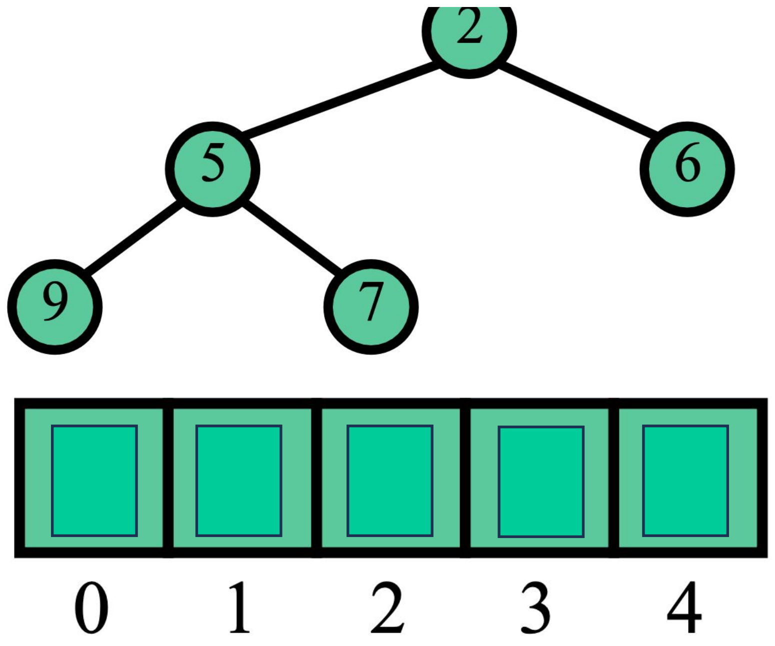
- The numbering can then be used as indices for storing the nodes directly in an array



/	*	+	+	4	−	2	3	1			9	5		
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Array based heap

Array/ArrayList of length n for heap with n keys

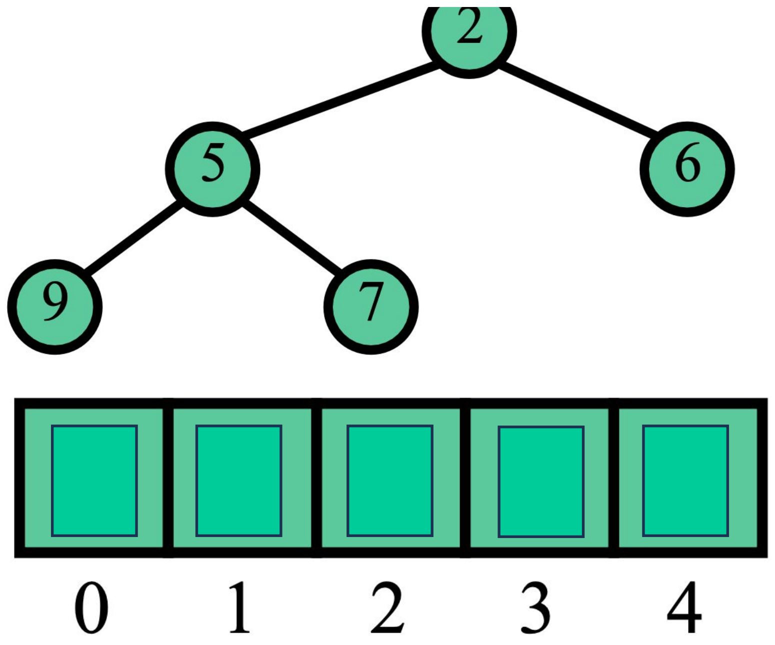


Array based heap

Array/ArrayList of length n for heap with n keys

Node at index i

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



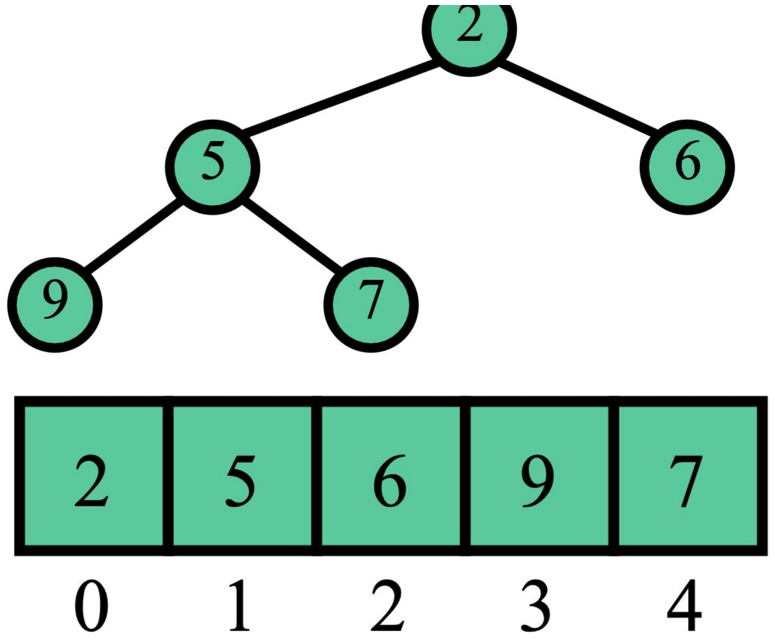
Array based heap

Array/ArrayList of length n for heap with n keys

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(n)

Heap Sort

Heap Sort

- “selection sort with the correct data structure”
- in place algorithm

- 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:
    a. runtime complexity?

3. Overall Runtime Complexity?