9 - Heaps

Joseph Afework
CS 241

Dept. of Computer Science
California Polytechnic State University, Pomona, CA

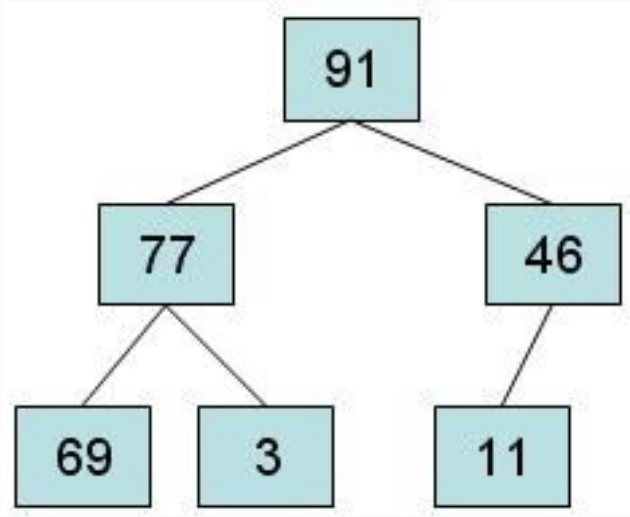
Agenda

- Intro
- Properties
- Search Operation
- Insertion Operation
- Deletion Operations
- Priority Queues

Reading Assignment

- Read Chapter 23 - Trees
  - Chapter 23 (Read about: **Heaps**)

Heaps



Binary Heap

Heap: A tree based data structure that satisfies the following heap property:

- If **B** is a child node of **A:**
  Key (A) must be greater than or equal to key (B)

Implications:

- Root always contains the element with the largest key. **(Max Heap)**

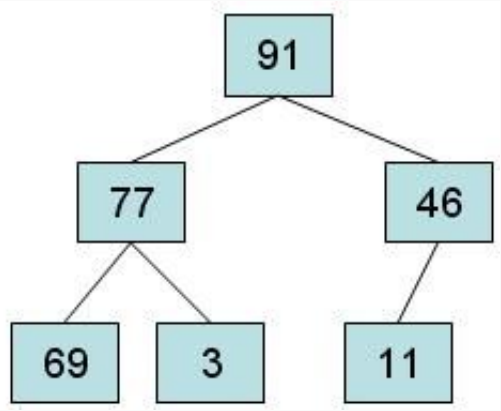
Note:

- Sometimes the opposite property is more useful: (Min Heap)
  a. If **B** is a child node of **A:** Key (B) must be greater than or equal to key (A)

Binary Heap Contd.

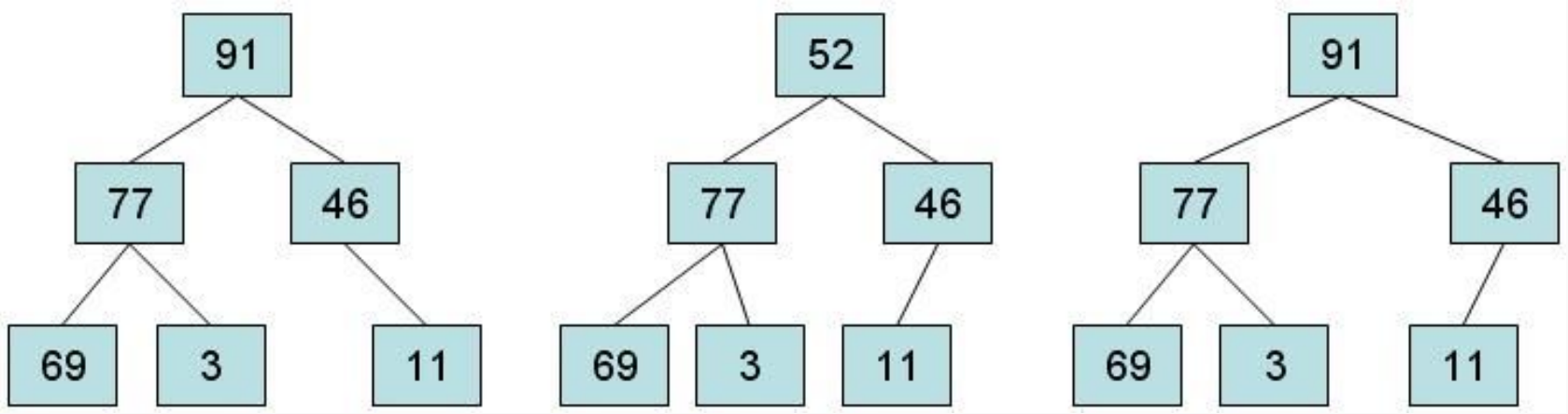
Binary Heap Storage Rules:

- The element contained by each node is greater than or equal to the elements of that node's children.
- The tree is a **complete binary tree**.
  -> Fix Violations



ICE 9.1 Binary Heap

Which of the following are binary heaps?



Implementation

Because a binary heap is always a complete binary tree, it is best to use an **array based representation.**

(Remember how to represent a binary tree using an array...)

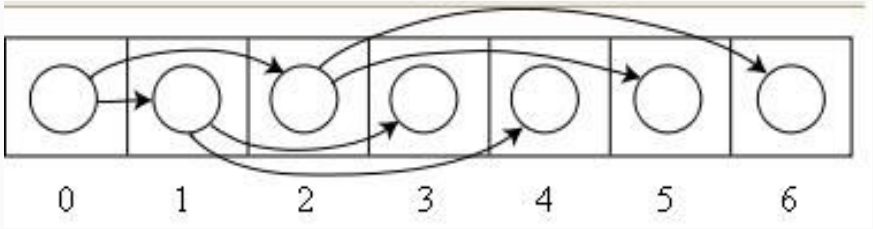
Benefits:

- Compact Storage
- No extra memory required for pointers
- Traversal using formulas (to find parent node, left child, right child)

Implementation Contd.

Array implementation pitfalls:

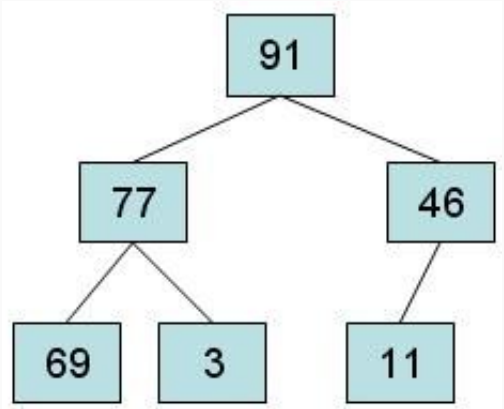
- Out of bound exceptions
- Resizing the array *(happens less often now)*



Node Implementation

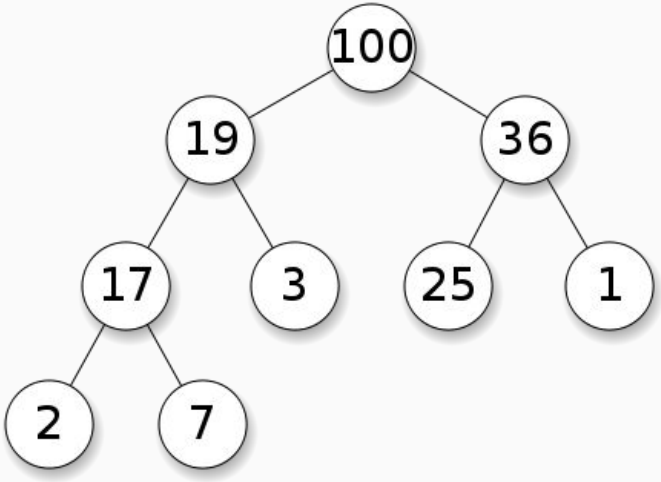
Node based representation is still ok:

- Insertion requires values to be placed the first open position in the binary tree…
- Iterating over the leafs (to find the adjacent elements) may require additional work...



ICE 9.2 Heap Search

Search the binary heap for the following values: **24, 4, 7, 18**

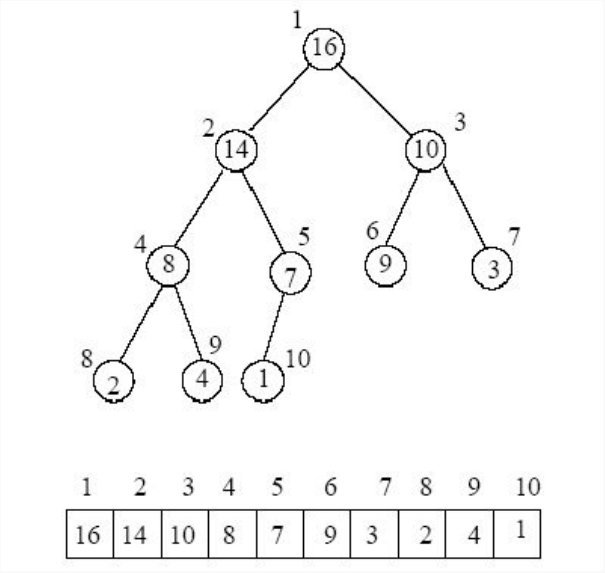


Search: O(n)

- Search is **not** one of the defined heap operations
  - Nodes are not stored in a sorted way
  - Search is exhaustive

Search: O(n)

- **Use the power of the array representation**
- Iterate over the values in the array
  - If value is found, return early
  - Else, continue until done



Insert (Max Heap)

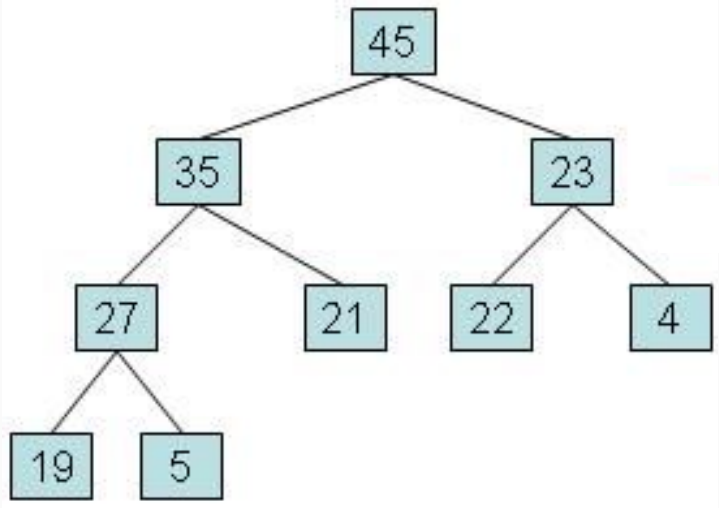
1. Place the new element in the heap in the first available location.
    - This keeps the structure as a complete binary tree, but it might no longer be a heap since the new element might have a greater value than its parent. **(Violation)**
2. While (new element is != root **AND** new element is greater than parent)
    - Swap parent with new element
3. **Done**

Note: that Step 2 will stop when the new element reaches the root **or** when the new element's parent has a value greater than or equal to the new element's value.

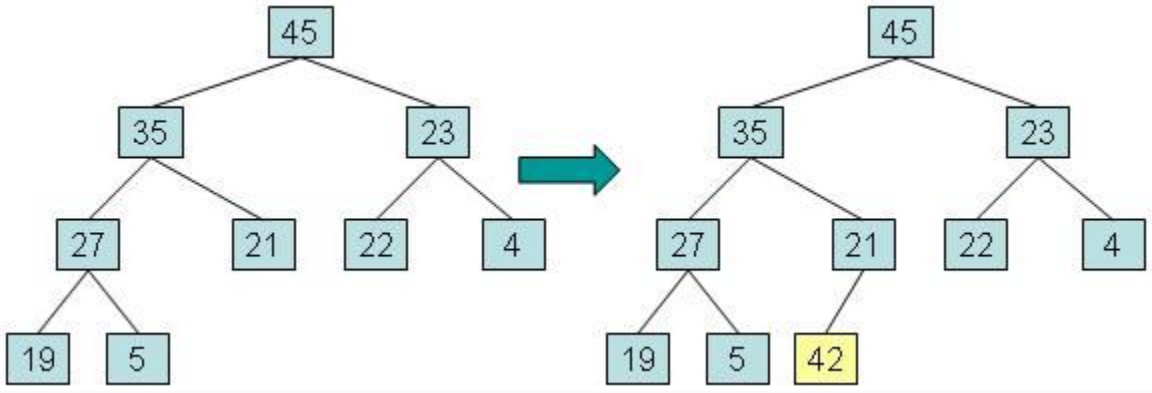
- Step 2 process is called **reheapification upward**.

ICE 9.3 Insert

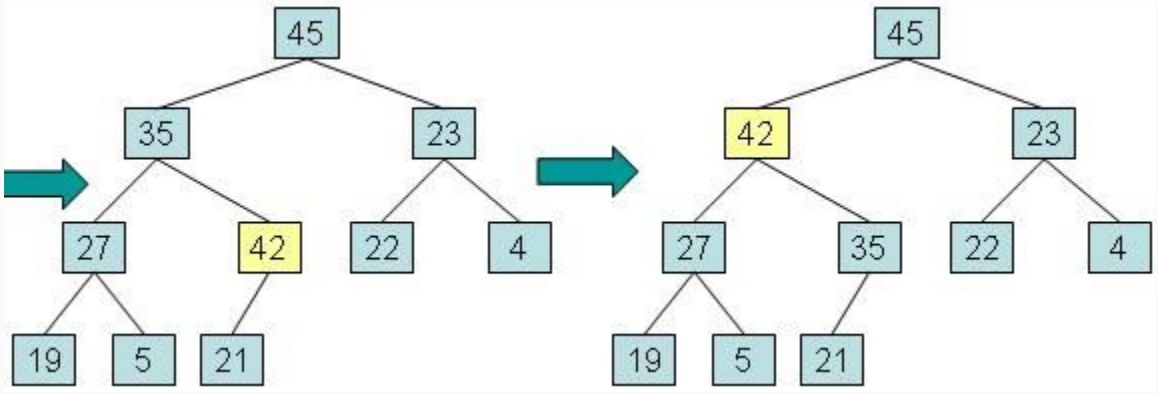
Insert the following value into the Max Binary Heap: **42**



ICE 9.3 Insert



ICE 9.3 Insert



ICE 9.3 Insert

Worst-Case: O(log n)

Why?

- Insertion starts at a fixed point, which is constant time
- How many swap operations can occur?
  - Tree is balanced, so the maximum path to the root is log(n)

Deletion (from root)

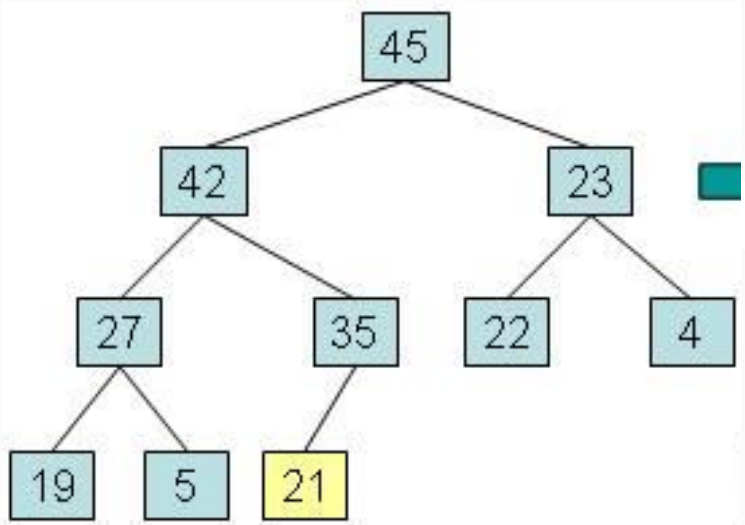
1. Replace the root with last element in the heap.
2. while (the moved element has a value that is lower than one of its children) swap the moved element with its greatest-value child.
3. Done

Note: Step 2 will stop when the moved element reaches a leaf or it has a value that is greater or equal to all its children.

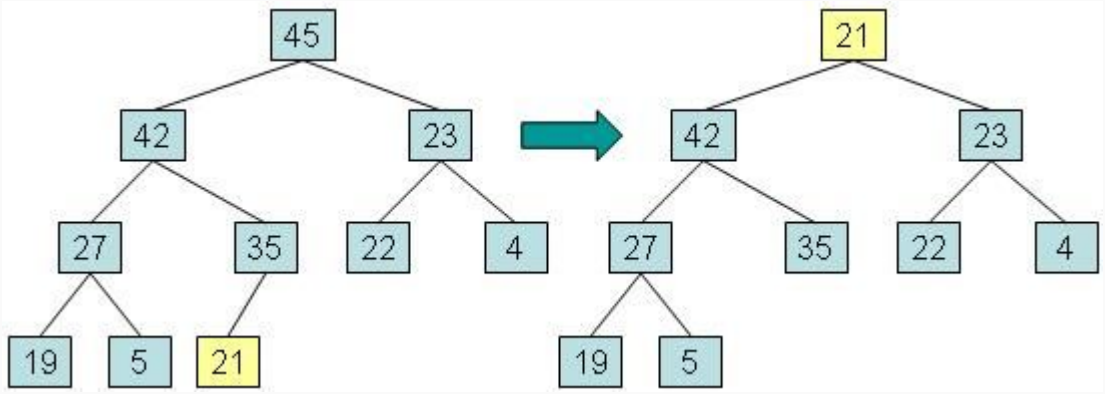
- Step 2 process is called **reheapification downward**.

ICE 9.4 Deletion

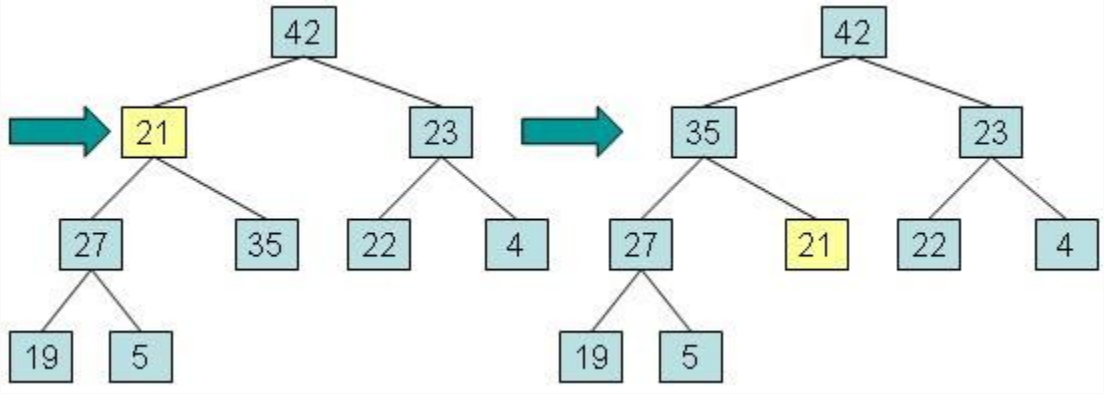
Delete the following value into the Max Binary Heap: **45**



ICE 9.4 Deletion

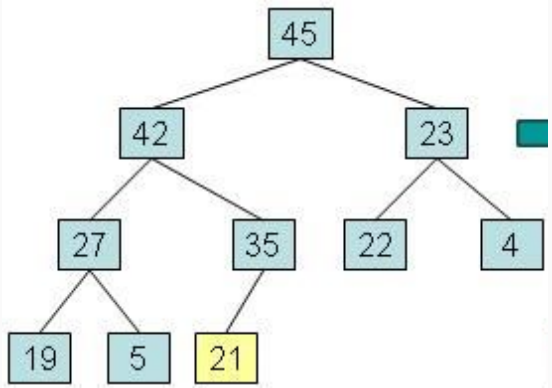


ICE 9.4 Deletion



Deletion (any node)

- Same rules as deleting from the root, but instead of replacing the root with the last element, replace the node you are targeting for deletion.
- Perform downwards reheapification



Deletion

Worst-Case: O(log n)

Why?

- Insertion starts at a fixed point, which is constant time
- How many swap operations can occur?
    - Tree is balanced, so the maximum path to the root is log(n)

Heap Construction

1. **Successive Insertion Method**
   - Restore the heap properties with every insertion (follow the insertion rules)
   - **Worst case: O(n log n)**

2. **Optimal Insertion Method**
   - **Worst case: O(n)** - see proof

Optimal Method (Max Heap)

1. Arbitrarily place the elements in a complete binary tree.
2. Starting from the **parent of the first (leaf)** and moving upwards (**leftwards**), swap the node with its largest child.
    a. **Note: you are iterating from n/2 to 0 (where root is).**
3. Once root is reached, perform downward reheapification on root.
4. Done

Note: Parent for leaf can be found starting at **Array [n/2] - 1 …. 0**

See: Building a heap section

Priority Queue

You can Implement a priority queue using a heap:

- Each node of the heap contains one element along with the element's priority
- Tree is maintained to follow modified heap storage rules:
  - The element contained by each node has a priority that is greater than or equal to the priorities of the elements of that node's children.
  - The tree is a complete binary tree.