CS 2100: Data Structures & Algorithms 1

Priority Queues / Heaps

Intro. To Priority Queues; Binary Heap Structure

Dr. Nada Basit // basit@virginia.edu

Spring 2022

Friendly Reminders

- The University updated the mask policy. As per my Request on Mar 28, 2022 (see Collab), I would greatly appreciate if you would do me a kind favor by **continuing to wear your masks** in CS 2100 (Ridley G008). I know it is a lot to ask, and it is **voluntary**, but I appreciate your understanding.
- If you forget your mask (or mask is lost/broken), I have a few available
  - Just come up to me at the start of class and ask!
- No eating or drinking in the classroom, please
- Our lectures will be **recorded** (see Collab) – please allow 24-48 hrs to post
- If you feel **unwell**, or think you are, please stay home
  - *We will work with you!*
  - At home: eye mask instead! Get some rest ☺

Priority Queues

An Abstract Data Type (ADT)

Motivation for Priority Queue

- Multiuser environment
  - Operating system must choose which process to run on CPU
- Management of limited resources
  - Bandwidth on network router
    - Limited bandwidth, but want to give best possible performance
    - Send traffic from highest priority queue first
      - Example: VoIP

Motivation for Priority Queue

- Hospital Waiting Room
- *Option A) Insert in FCFS order to List; Remove by searching for highest priority*
- *Option B) Insert in sorted order (priority) to List; Remove at one end of List*

- ***We want…***
  - ***Efficient*** *patient registration (insert)*

    AND

  - ***Efficient*** *removal (to see a Dr.) based on priority level*

- How can we achieve BOTH??

Solution? Heaps!

Heaps ("Binary Heaps")

- The **heap** data structure is an example of a ***balanced binary tree***
- Useful in solving three types of problems:
  - Finding the **min or max** value within a collection
  - **Sorting** numerical values into ascending or descending order
  - Implementing another important data structure called a **priority queue**

Other priority queue scenarios in real life:

- ➢*Professor office hours (what if another professor stops by, or the department chair?) or*
- ➢*Getting on an airplane (first class + families, frequent flyers, by row, etc.) or*
- ➢*Shipping packages (amount of shipping paid, destination, etc.)*

We will implement the abstract idea of a PRIORITY QUEUE with Binary Heap!

Priority Queue ADT - Model

- **<u>Operations</u>**
  - Push
    - Inserts with a *priority*
  - Peek
    - Finds the *minimum* element (doesn't remove)
  - Poll (remove)
    - Finds, returns, and *removes minimum* element

Priority Queue Data Structures

Data Structure	push	peek	poll (remove)
Unsorted array	$\Theta(1)$ *amortized*	$\Theta(n)$	$\Theta(n)$
Unsorted linked list	$\Theta(1)$	$\Theta(n)$	$\Theta(n)$
Sorted array	$\Theta(n)$	$\Theta(1)$	$\Theta(1)$
Sorted linked list	$\Theta(n)$	$\Theta(1)$	$\Theta(1)$

Structure	push	peek	poll (remove)
BST	$\Theta(\text{n})$	$\Theta(\text{n})$	$\Theta(\text{n})$
AVL / RB tree	$\Theta(\log \text{n})$	$\Theta(\log \text{n})$	$\Theta(\log \text{n})$
Hash table	*ideally constant*	$\Theta(\text{n})$	$\Theta(\text{n})$

- We would like:
  - **peek:** always **constant**
  - **push:** worst case **$\Theta(\log n)$**, typical case **constant**
  - **poll (remove):** worst and average case **$\Theta(\log n)$**

Binary Heaps

An Abstract Data Type (ADT)

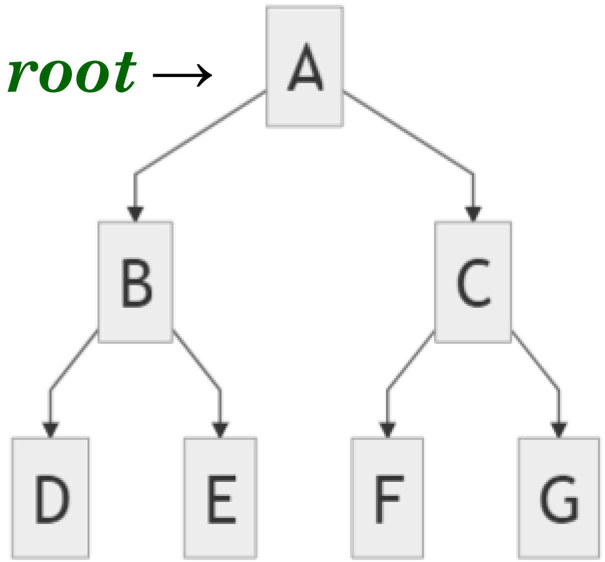
Heaps ("Binary Heaps")

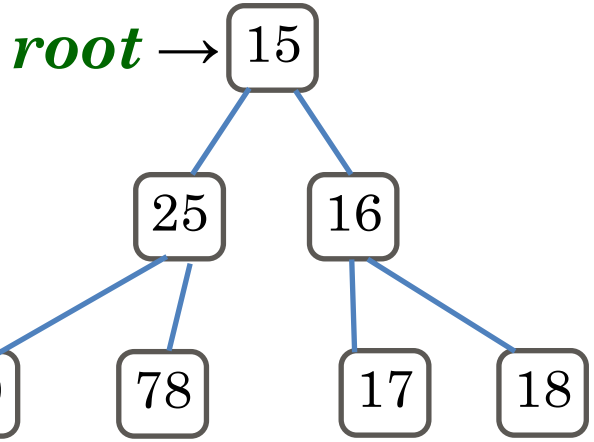
- A **binary heap** is a **heap data structure** that is one possible implementation of a priority queue

- It is a binary tree (not a BST) with *two additional **constraints***:
- **Shape (structure) property:**
  - A heap is a **complete binary tree**, a binary tree of height (i) in which all leaf nodes are located on **level (i) or level (i-1)**, and all the **leaves** on level (i) are as far to the **left** as possible
- **Order (heap) property:**
  - The data value stored in a node is **less than or equal to** the data values stored in all of that node's descendants
  - (Value stored in the root is always the **smallest** value in the heap)
  - Parent nodes have a higher ***priority*** than any of their children

Some Definitions

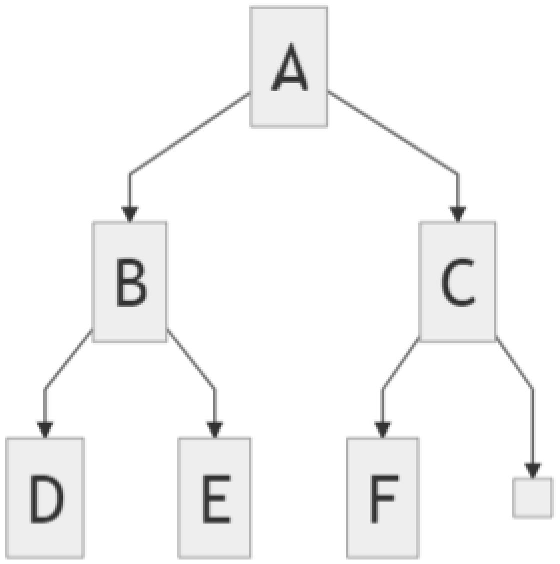
- A ***perfect* (or *complete*) binary tree** has all leaf nodes at the **same depth**; all internal nodes have **2 children**.





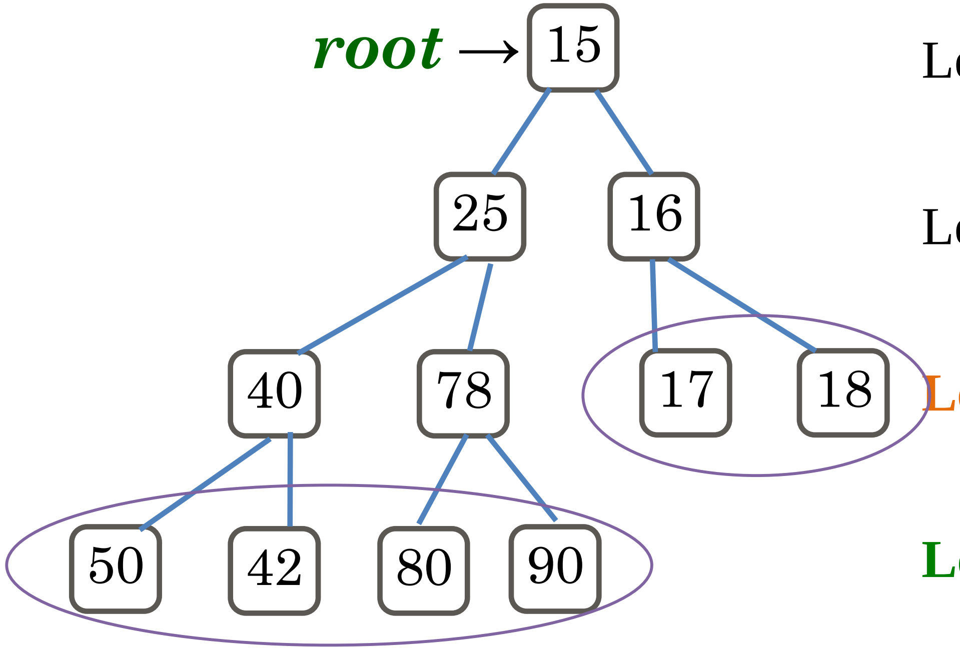
Heap Shape (Structure) Property

- A **binary heap** is an **almost complete** binary tree. the tree is completely filled, except possibly the bottom level, which is filled left to right.
- Almost complete binary tree of height ***h***:
- For h = 0, just a single node
- For h = 1, left child or two children
- For h ≥ 2, either:
  - the left subtree of the **root** is ***complete*** with height h-1 and the right is ***almost complete*** with height h-1, OR
  - the left is ***almost complete*** with height h-1 and the right is ***complete*** with height h-2



Leaf nodes on level (i) or level (i-1)?

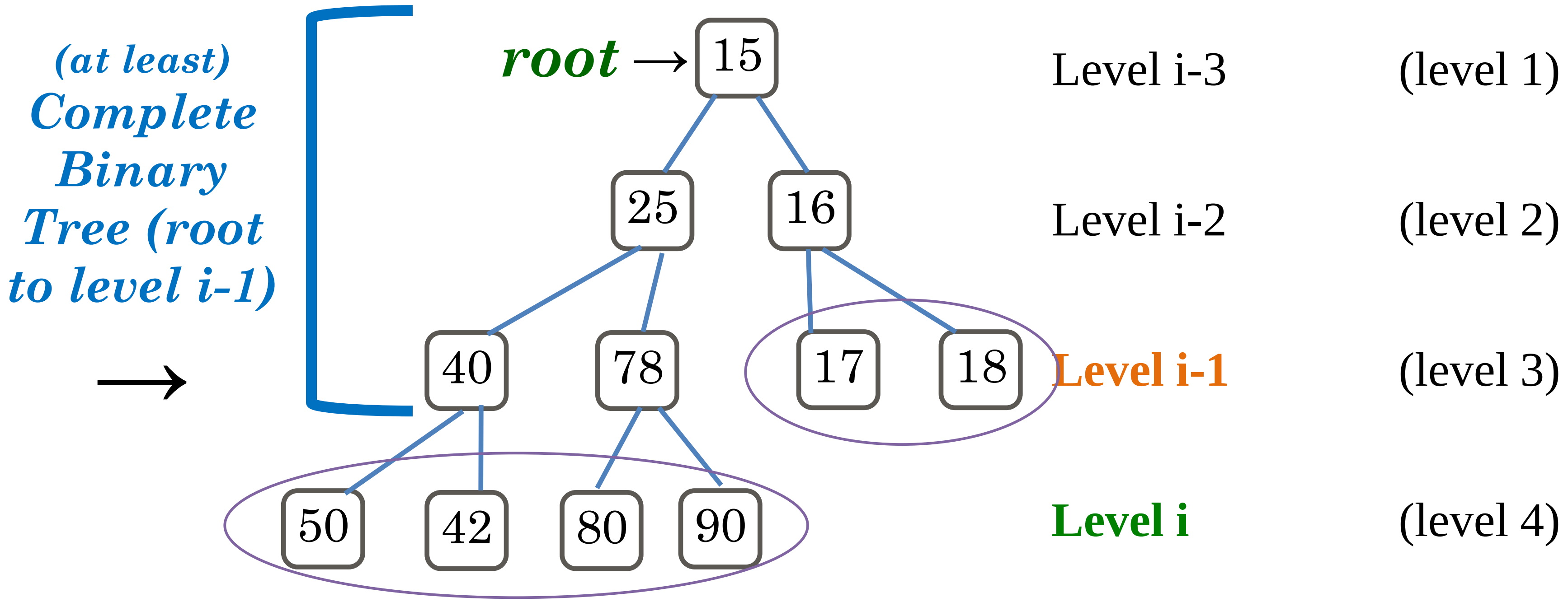
- Notice that all leaves are located on **level (i)** or **level (i-1)**
- Where **level (i)** is the *furthest away* from the **root**



Level i-3	(level 1)
Level i-2	(level 2)
Level i-1	(level 3)
Level i	(level 4)

Leaf nodes on level (i) or level (i-1)?

- Notice that all leaves are located on **level (i)** or **level (i-1)**
- Where **level (i)** is the *furthest away* from the **root**



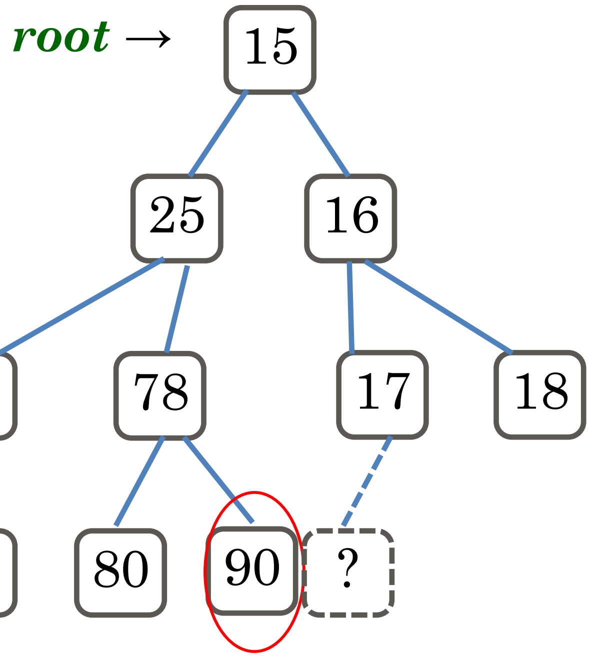
Heap Shape (Structure) Property - Implementation

- all leaves are on the *lowest two levels*
- **nodes are <u>added</u> on the *lowest level, from left to right***
- **nodes are <u>removed</u>** (to replace the root) **from the *lowest level, from right to left***

In order to maintain the **shape property**

Where are nodes added or removed?

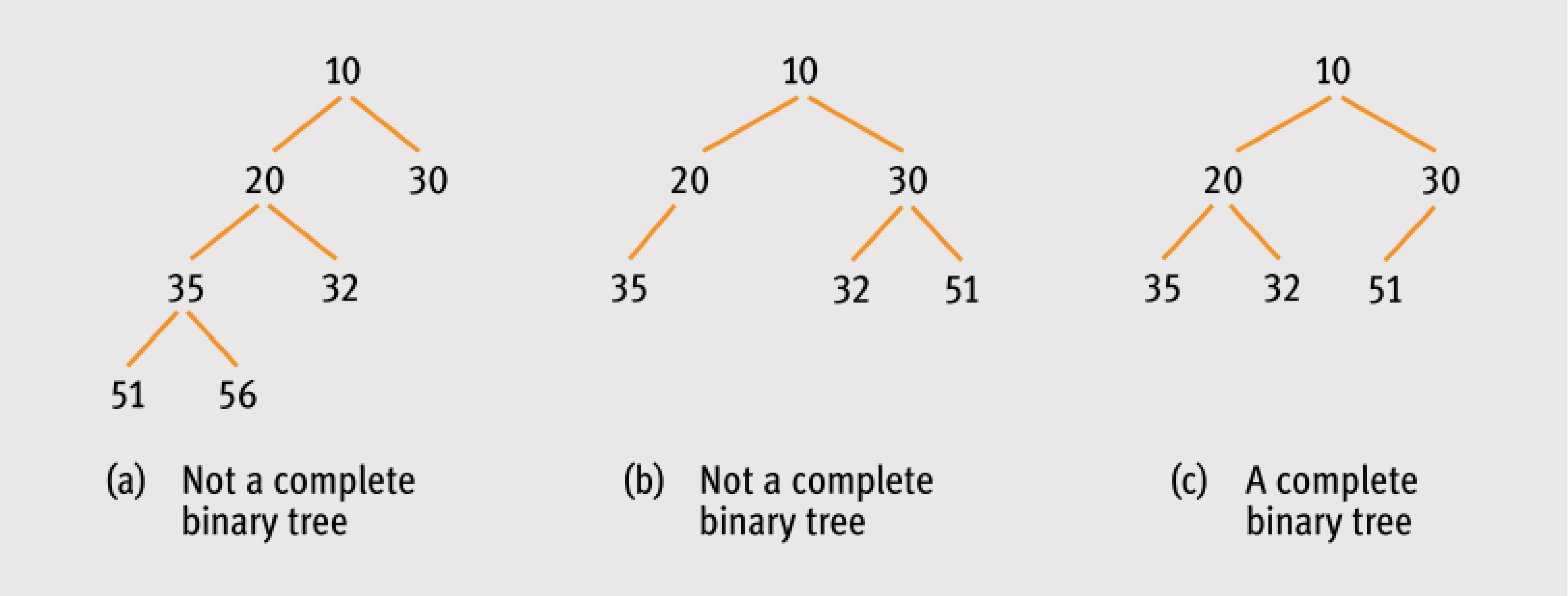
- Where to **add**? Left child of 17
- What to **remove**? Node 90
  (to replace the **root**)



- Nodes added: → → → → *from left to right (no gaps)* → → → →
- Nodes removed:← ← ← ← *from right to left (no gaps)* ← ← ← ←

Complete Binary Tree

Which of these trees is a complete binary tree?



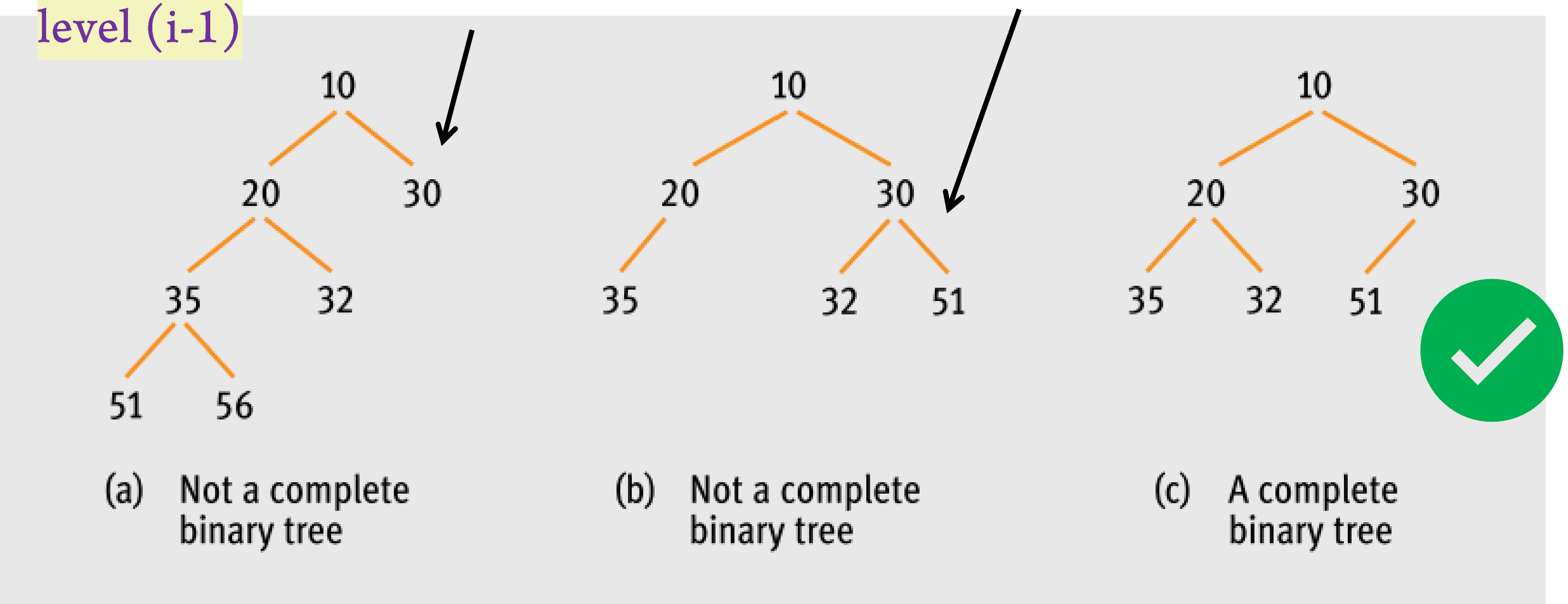
[FIGURE 7-29] Examples of valid and invalid complete binary trees

(complete except for the 'last' level)

Why Are The First Two Invalid??

All leaf nodes are NOT located on level (i) or level (i-1)

Leaves on level i are NOT as far to the left as possible

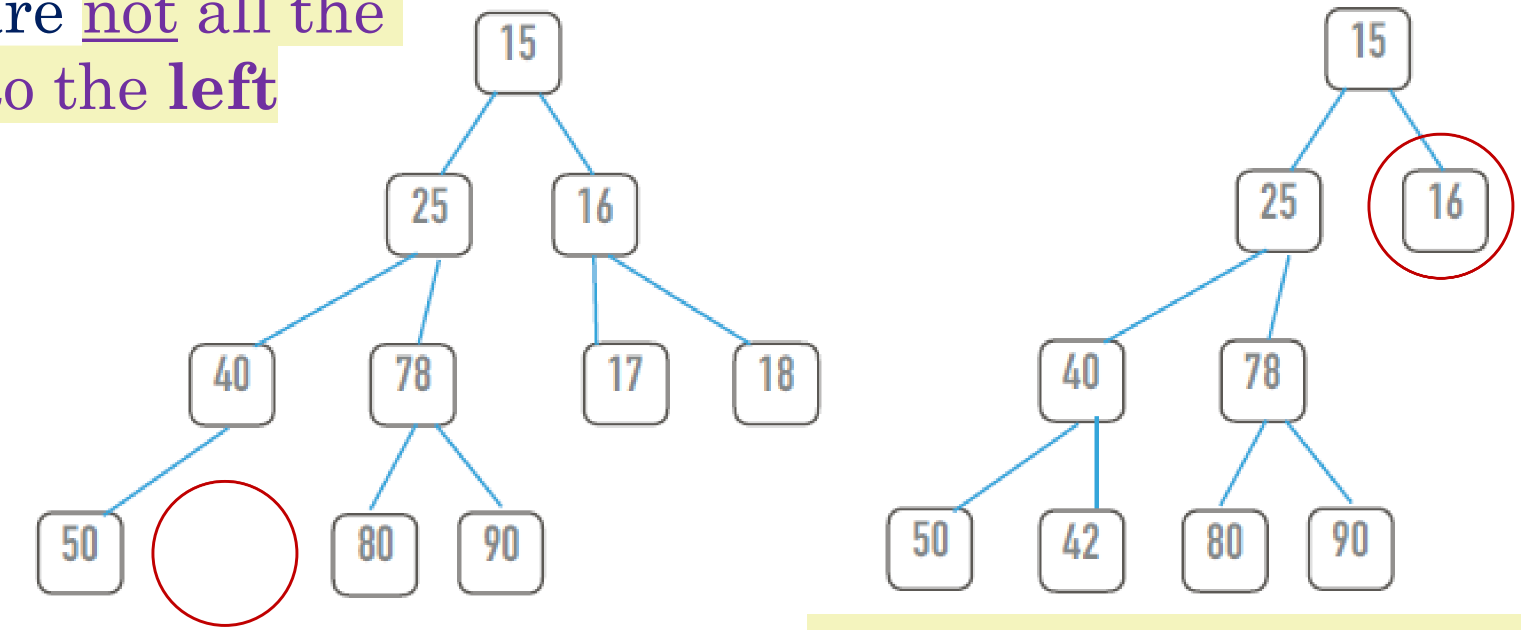


[FIGURE 7-29] Examples of valid and invalid complete binary trees

(complete except for the 'last' level)

Examples of Invalid Heaps

Nodes on the bottom row are <u>not</u> all the way to the **left**

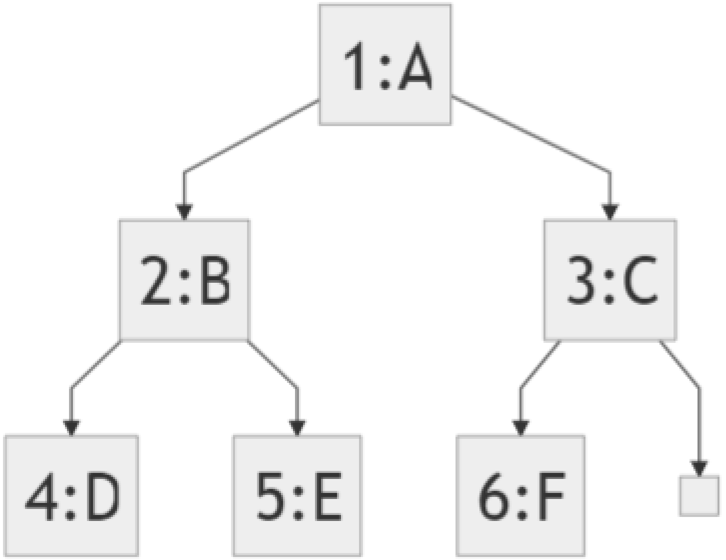


The right leaf is **not balanced**

(Leaf nodes appear at an *inappropriate* level – not level (i) or (i-1))

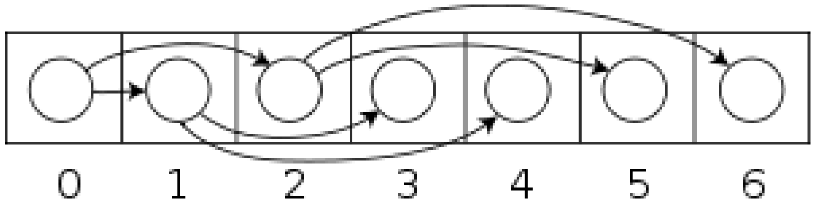
Complete Binary Trees in 1-D Arrays

- We can store the elements of our heap in a one-dimensional array in strict left-to-right, **level order** ***("breadth-first traversal")***
- That is, we store all of the nodes on level *i* from left to right before storing the nodes on level *i* + 1. This one-dimensional array representation of a heap is called a **heapform**
  - *Usually we ignore index position 0*
  - *Some real handy and simple formulas can be used to compute children, siblings,…*
    - `2i: left child, 2i+1: right child`
    - `Math.floor(i/2): parent`



-1	A	B	C	D	E	F

Implementing a Heap in an Array



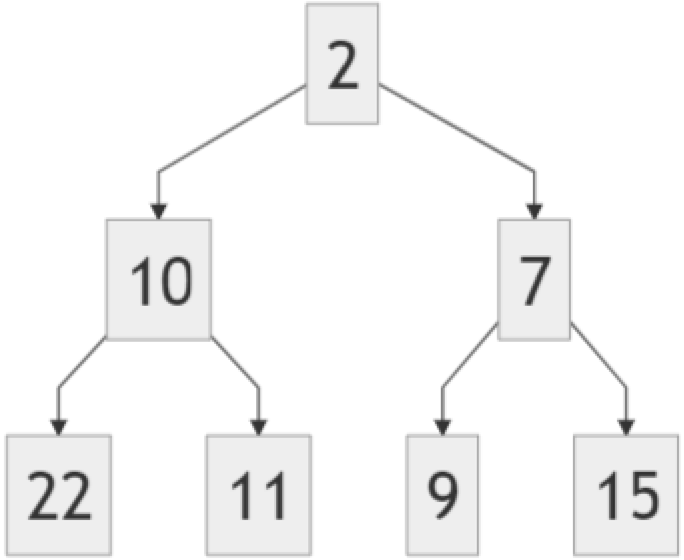
- Several methods can be implemented without recursion.
  For a heap with a **starting index of 1:**
  - `int getParent ( i )      { return Math.floor(i / 2); }`
  - `int getLeftChild ( i )   { return 2i; }`
  - `int getRightChild ( i ) { return 2i + 1; }`
  - `int getSibling ( i )     { if i is even and i < n: i+1,`
    `else if i is odd and i > 2: i-1; }`
- For a heap with a **starting index of 0:**
  - `int getParent ( i )      { return Math.floor[(i-1) / 2]; }`
  - `int getLeftChild ( i )   { return 2i + 1; }`
  - `int getRightChild ( i ) { return 2i + 2; }`
  - `int getSibling ( i )     { if i is odd and i < n-1: i+1,`
    `else if i is even and i > 1: i-1; }`

Why Better Than References?

- We do not need pointers/references in this array-based representation because the parent, children, and siblings of a given node must be placed into array locations that can be determined with some simple calculations (see previous slide)
- **Saves space**
  - No need to store parent/child references
  - Arrays are more compact in memory
- **Saves time**
  - Arrays work better with cache
  - (*2), (/2), + operations are faster than dereferences
  - Allocating objects is slow compared to arrays
- Parent is easy to locate (i.e. free parent pointer)

Heap Order (Heap) Property

- The data value stored in a node is **less than or equal to** the data values stored in all of that node's descendants
- (Value stored in the root is always the **smallest** value in the heap)
- Parent nodes have a higher ***priority*** than any of their children



- For every non-root node X, the key in the parent of X is less than (or equal to) the key in X. Thus, the tree is partially ordered.

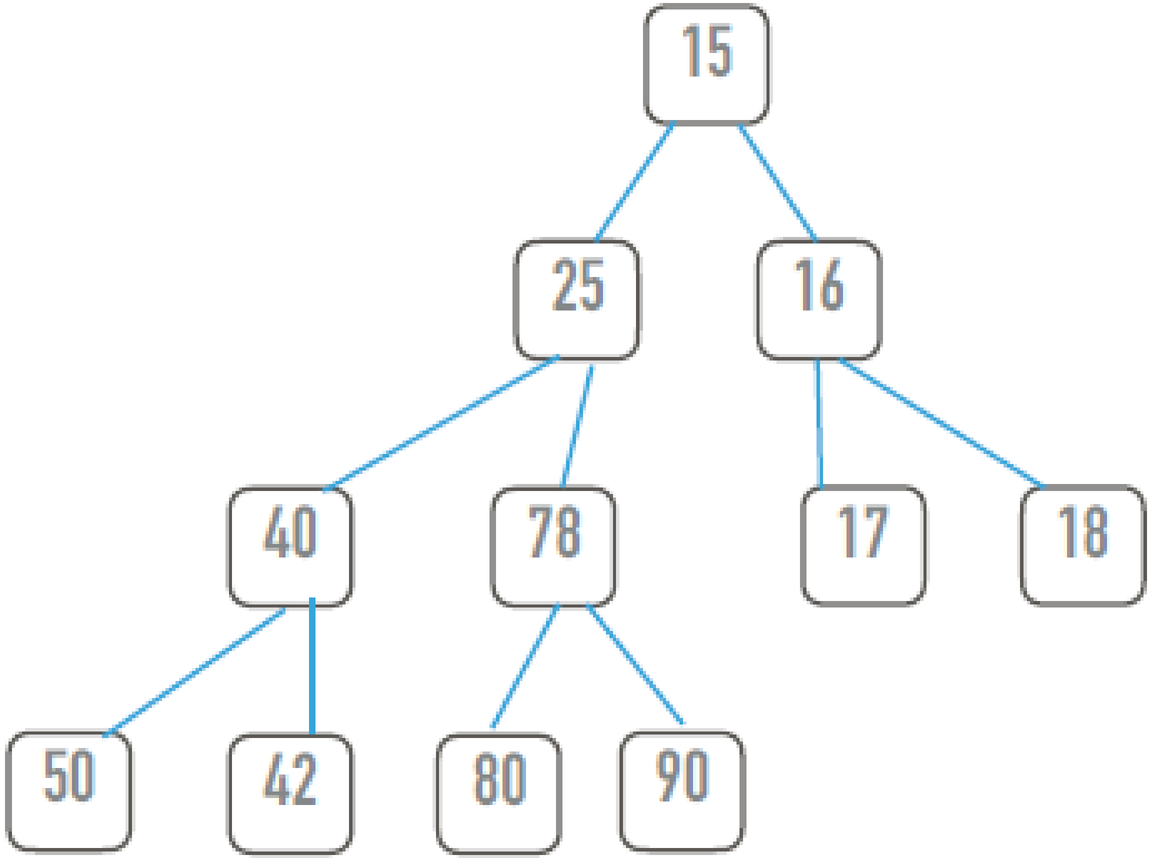
Minheap vs Maxheap

- We could just as easily define a heap in which a node's value is ***greater than or equal to*** the data values stored in all of that node's descendants.

- In this case, all algorithms would simply change the < operator to a >, and every occurrence of the word smallest would be replaced by largest.

Minheap

- This is a **min heap**
- The **smallest** value is the **root** of the tree
- All nodes are **smaller** than ALL its descendants



- Note: a heap is **NOT** a binary search tree – values larger than the root can appear on either side as children

No orderings between sibling nodes

- There are no implied orderings between siblings, therefore, **both** of the trees below are min-heaps :

```
      5              5
    /   \          /   \
  10     12      12     10
```

- What does matter is **the parent-child relationship (top-bottom)** rather than siblings (left-right)

Heap Order (Heap) Property:
Can you recognize min heaps... (1)

- Which of the following are min-heaps?

1)
```
            5
          /   \
       10       8
      /  \     /  \
    12    85  14   13
   /  \
 22    11
```

2)
```
            13
          /    \
        19      36
       /  \    /  \
     24    99 46   42
    /  \
  25    26
```

- Answer:

Heap Order (Heap) Property: Can you recognize min heaps... (1)

- Which of the following are min-heaps?

1)
```
          5
        /   \
     10       8
    /  \     /  \
   12   85  14   13
  /  \
22    11
```

2)
```
          13
        /    \
      19      36
     /  \    /  \
   24   99  46   42
  /  \
25    26
```

- Answer: Only heap 2.

Heap Order (Heap) Property:
Can you recognize min heaps… (2)

- This one is NOT a minheap. Why?