Lecture 2

Instructor: Karteek Sreenivasaiah

2nd August 2018

Plan for the day

- Revise binary Heaps.
- Binary Search Trees.

Heap

A binary max-heap:

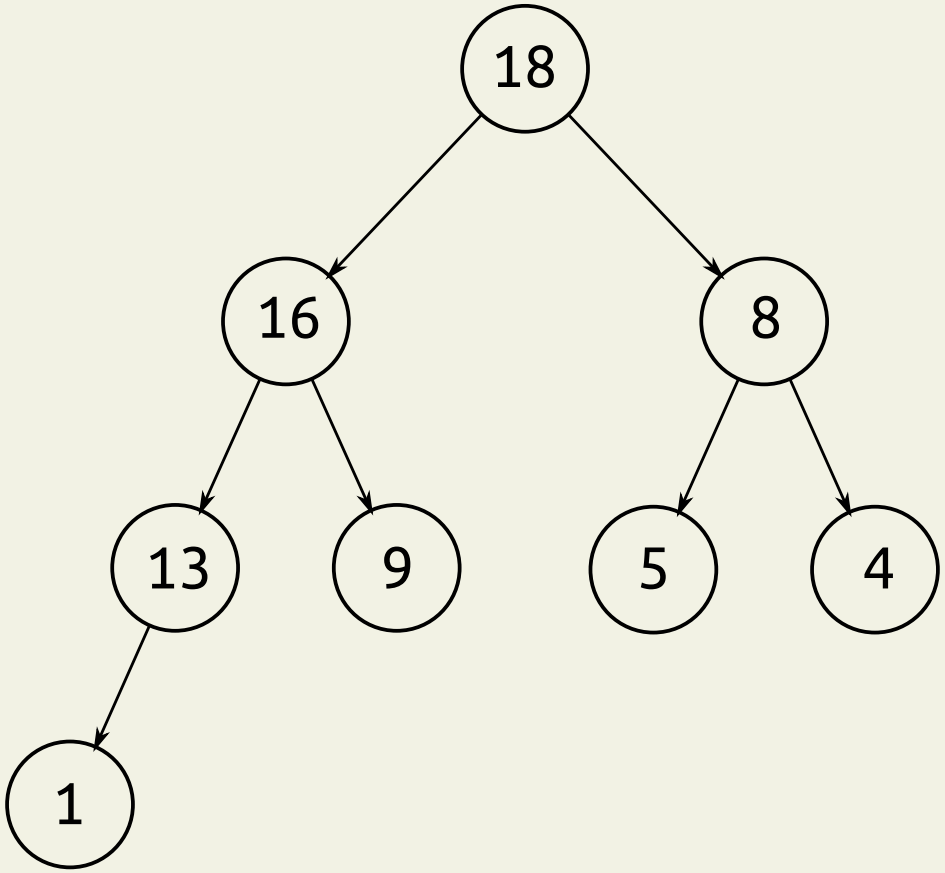
1. Is a complete binary tree except possibly for the lowest level.
2. The value of a node is greater than that of both its children.

A max-heap supports the following functions:

- Insert(*val*) – Inserts *val* into the heap.
- ExtractMax() – Returns and removes the maximum element from the heap.

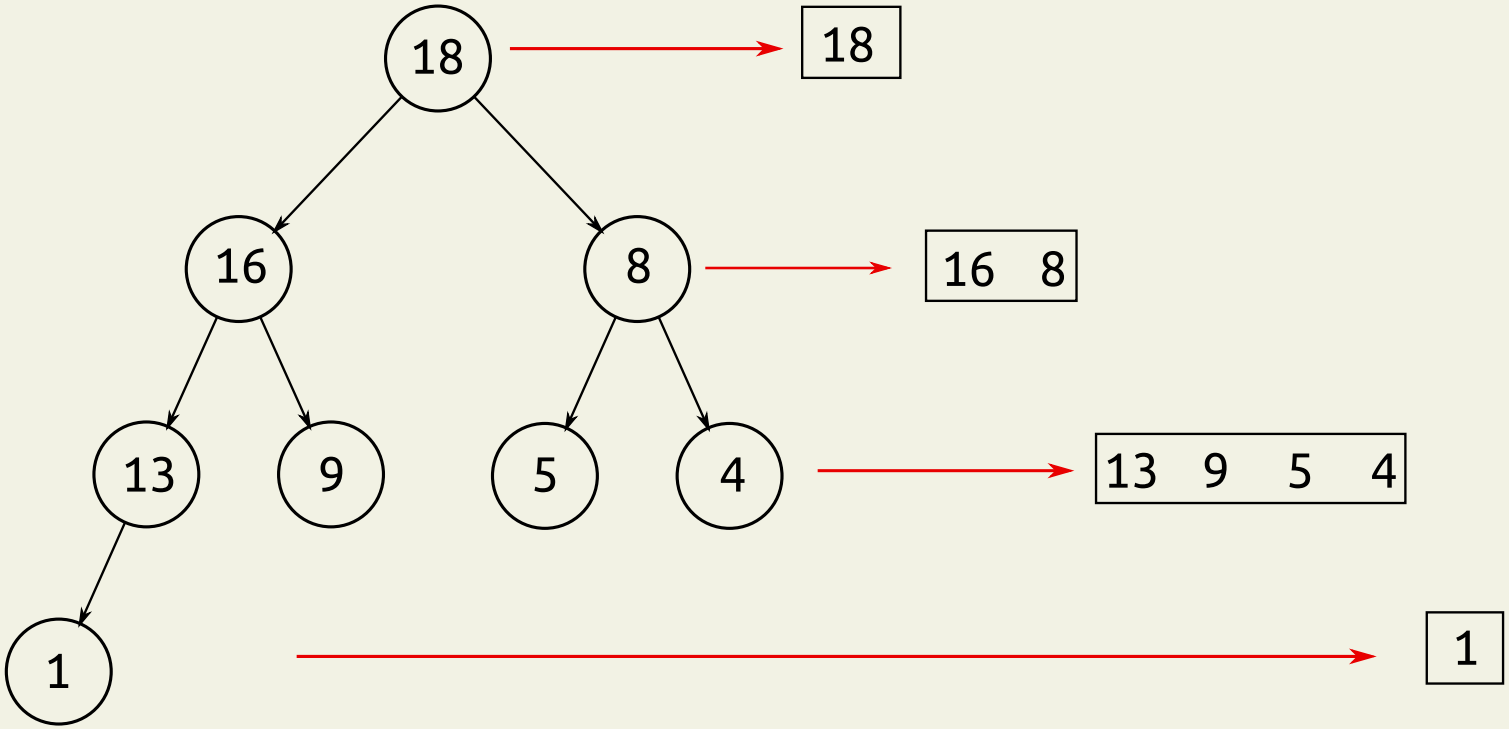
Data Structure

Heaps are usually implemented using arrays.



Data Structure

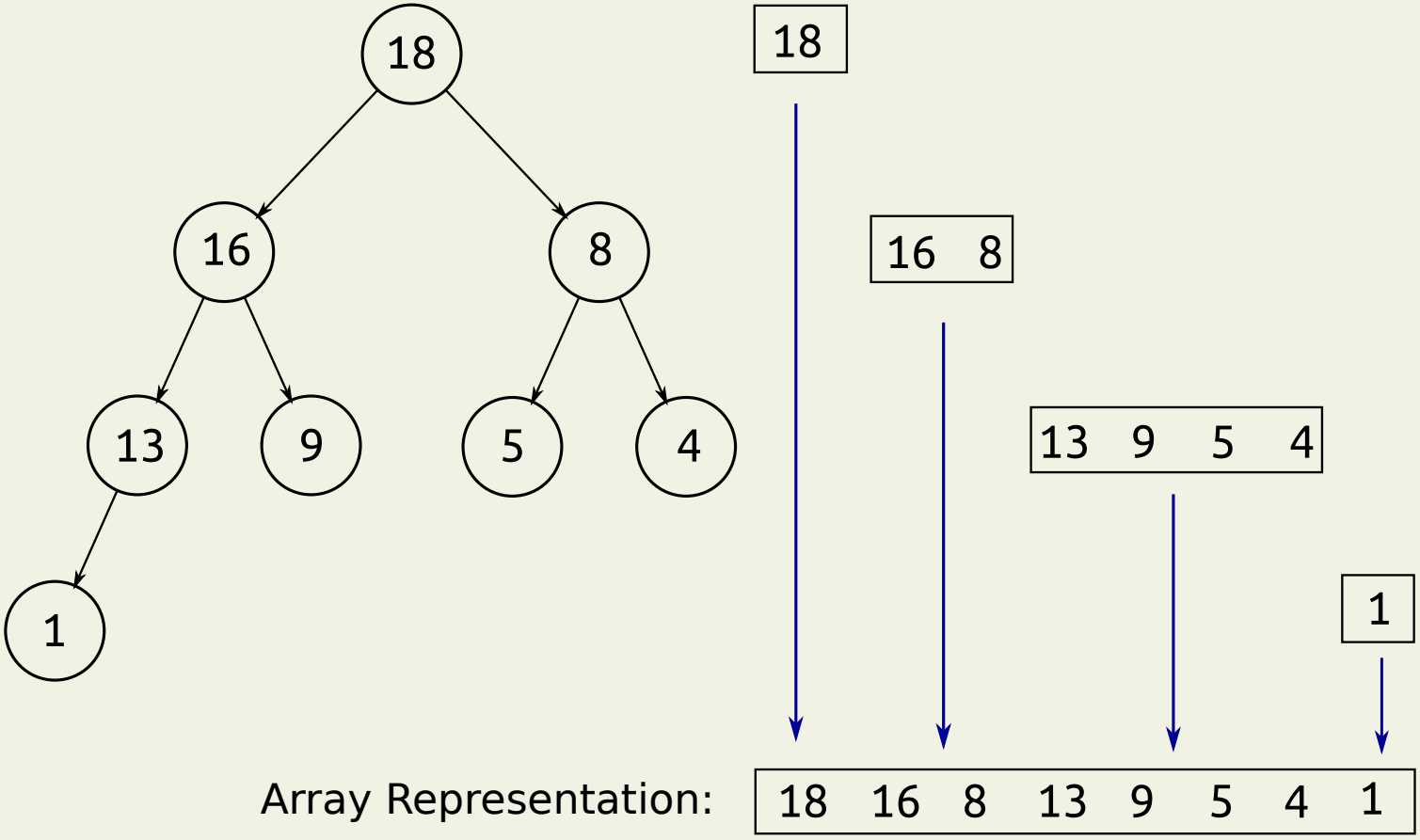
Read off from top to bottom, left to right.



Array Representation:

Data Structure

Read off from top to bottom, left to right.



Questions

About Heaps:

1. How many nodes does a height h heap have? (both bounds)
2. What is the maximum height of a heap with n nodes?

About the array implementation:

1. What is the array index of the children of the node at $A[i]$?
2. What is the array index of the right sibling of the node at $A[i]$?

Heaps using arrays

Typically, a heap is built starting with an arbitrary array:

- Procedure BuildHeap(Array A) – Takes an array and rearranges the elements to form a heap.

In Object Oriented languages, BuildHeap is essentially the *Constructor* of class Heap.
The procedure BuildHeap works by using a method called Heapify(*node*).

Heapify

The HEAPIFY(*node*) procedure:

- If *node* violates the heap property:
  1. Swap value of *node* with the largest of its two children.
  2. Call HEAPIFY on the child replaced.
- Else, do nothing and return.

Heapify

The HEAPIFY(*node*) procedure:

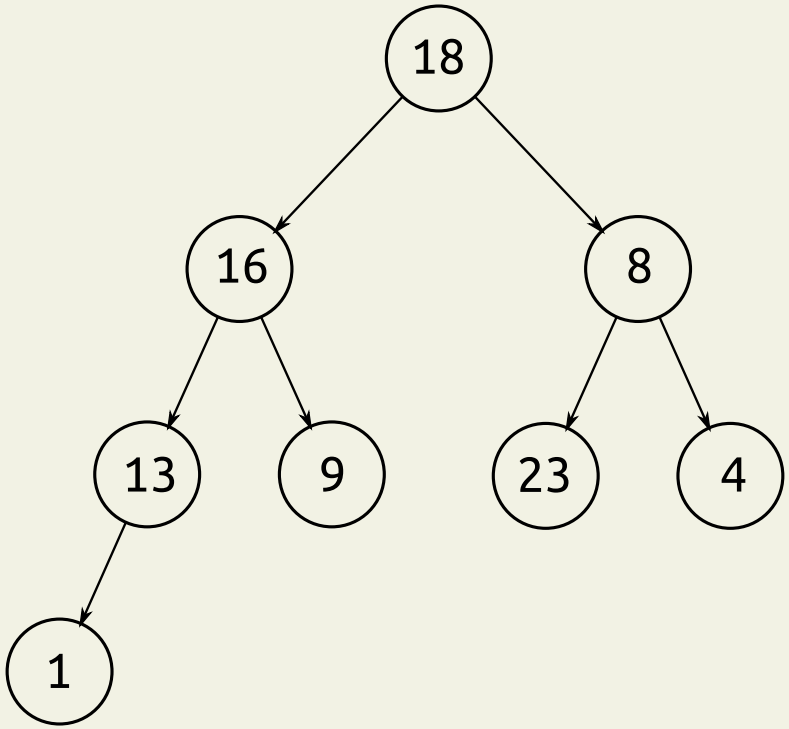
- If *node* violates the heap property:
  1. Swap value of *node* with the largest of its two children.
  2. Call HEAPIFY on the child replaced.
- Else, do nothing and return.

Note:

- The Heapify procedure assumes that both the subtrees under *node* are already heaps.
- It merely resolves the possible conflict between the value at *node* and its children and recurses.

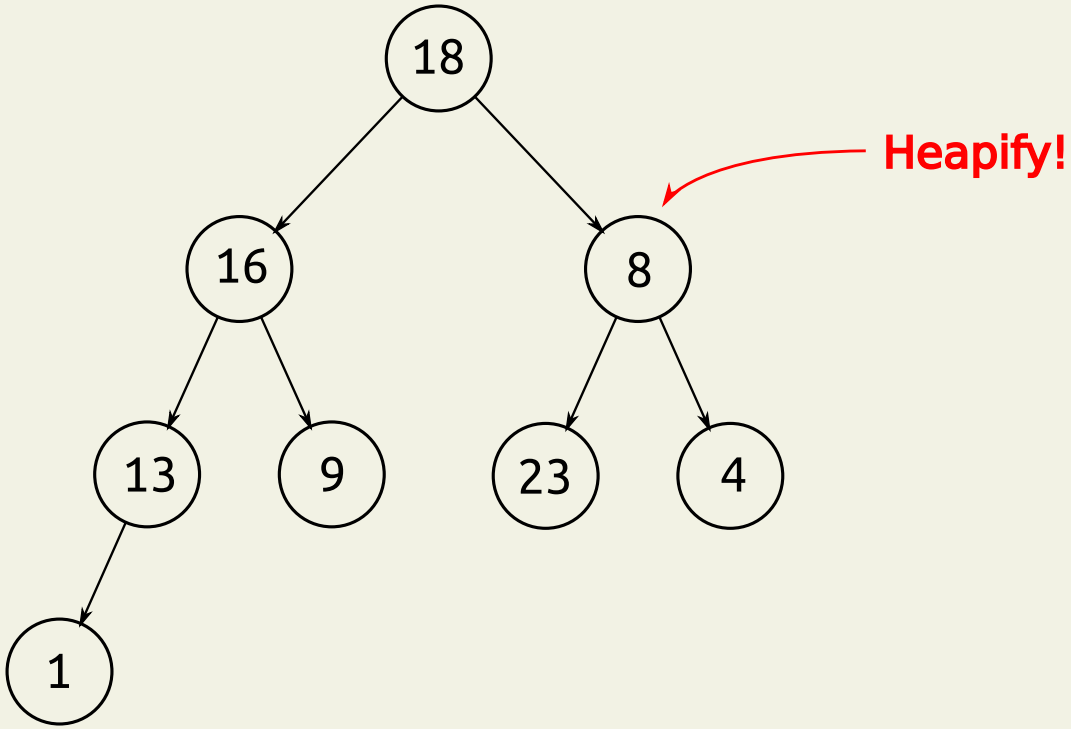
Heapify

Example:



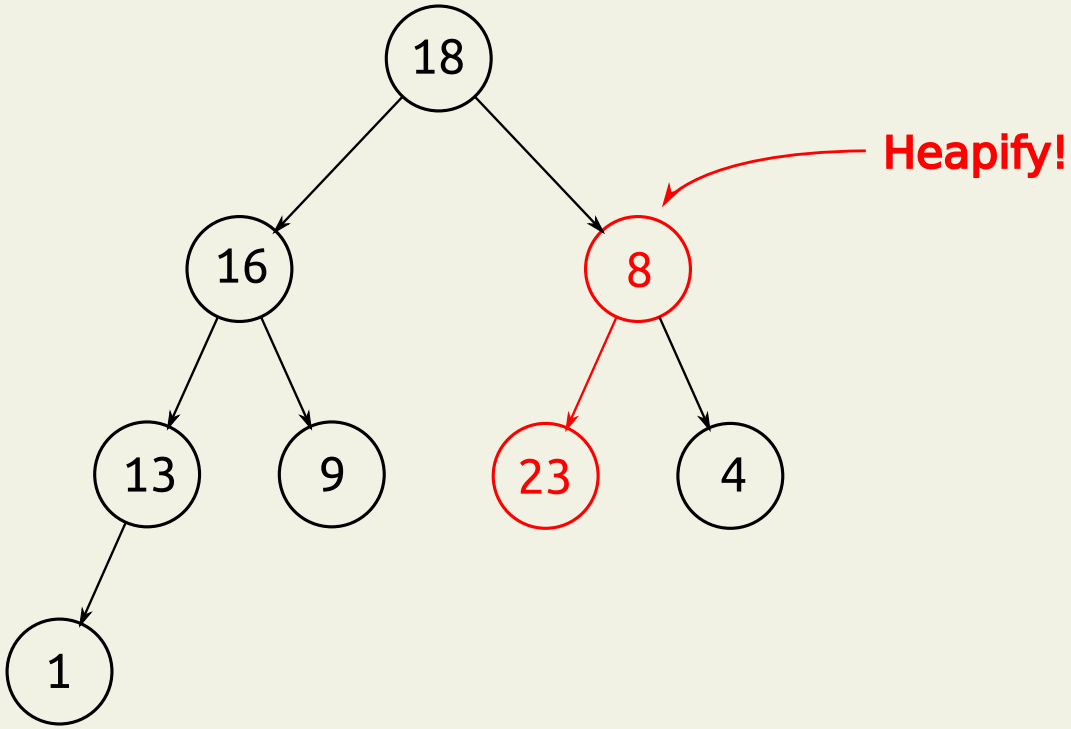
Heapify

Example:



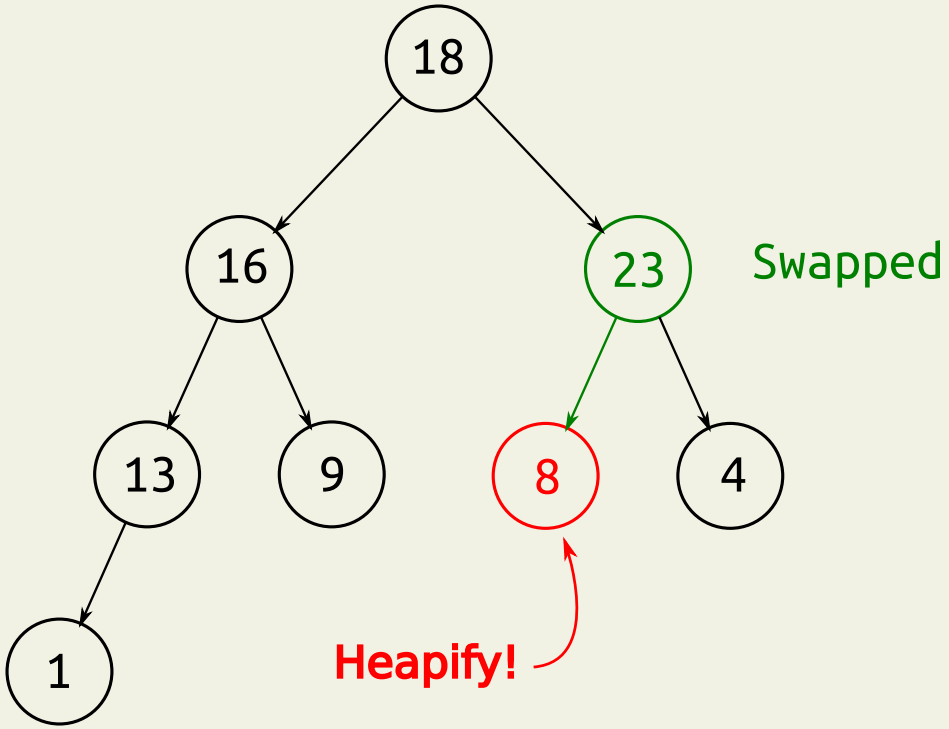
Heapify

Example:



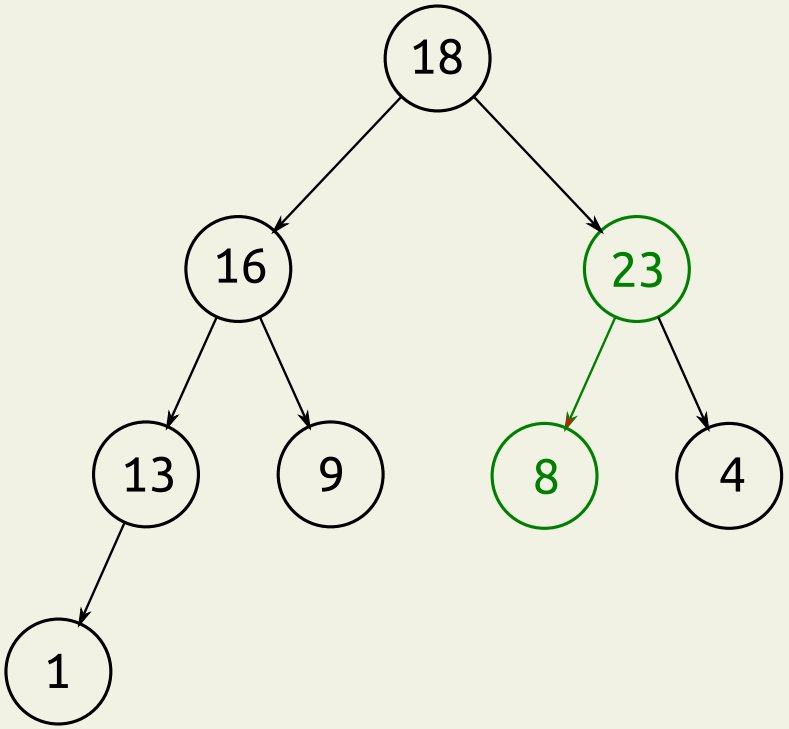
Heapify

Example:



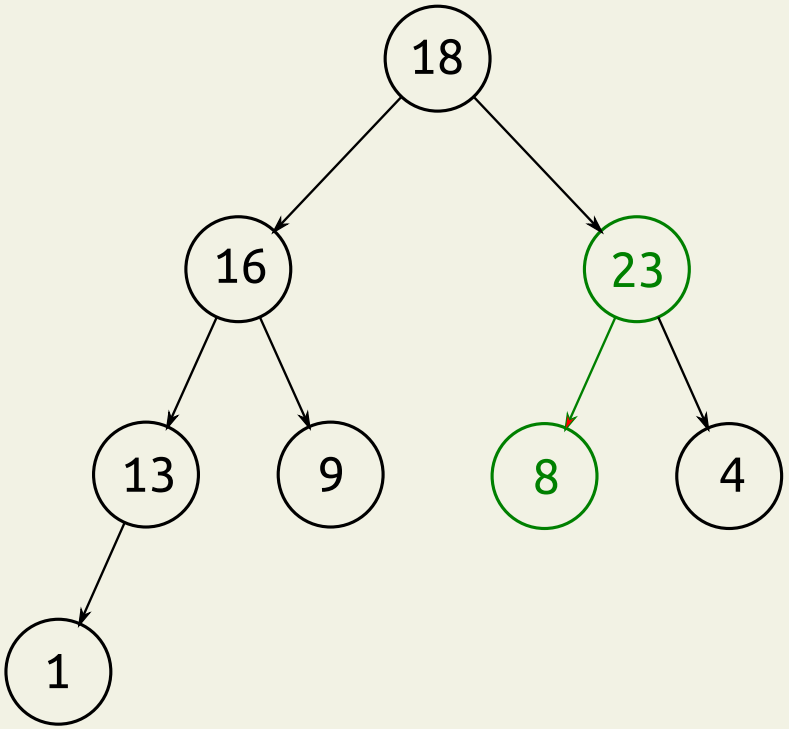
Heapify

Example:



Heapify

Note that Heapify only resolves conflicts downwards.



Building a Heap

Two ways:

- William's method: Take each element and use INSERT procedure.
- Floyd's method: Take all elements in an arbitrary array. Heapify repeatedly.

Building a Heap

The procedure BuildHeap(A) by Floyd is the following:

- For i from n to 1:
  - Heapify(i)

Building a Heap

The procedure BuildHeap(A) by Floyd is the following:

- For i from n to 1:
  - Heapify(i)

Note: Indices $n/2$ to n form leaves of the heap.
The leaves are already heaps (trivially).
Hence it suffices to run the above loop from $n/2$ to 1.

Exercises

Write the following procedures:

- Insert(*val*):
  - Insert new value as the last element in the array.
  - Repeatedly Heapify *upwards* from the new element.
- ExtractMax(): Swap positions of root with last leaf. Heapfiy at new root.

Binary Search Tree

A Binary Search Tree (BST) is a tree that satisfies the following:

- For every node X in the BST, we have:

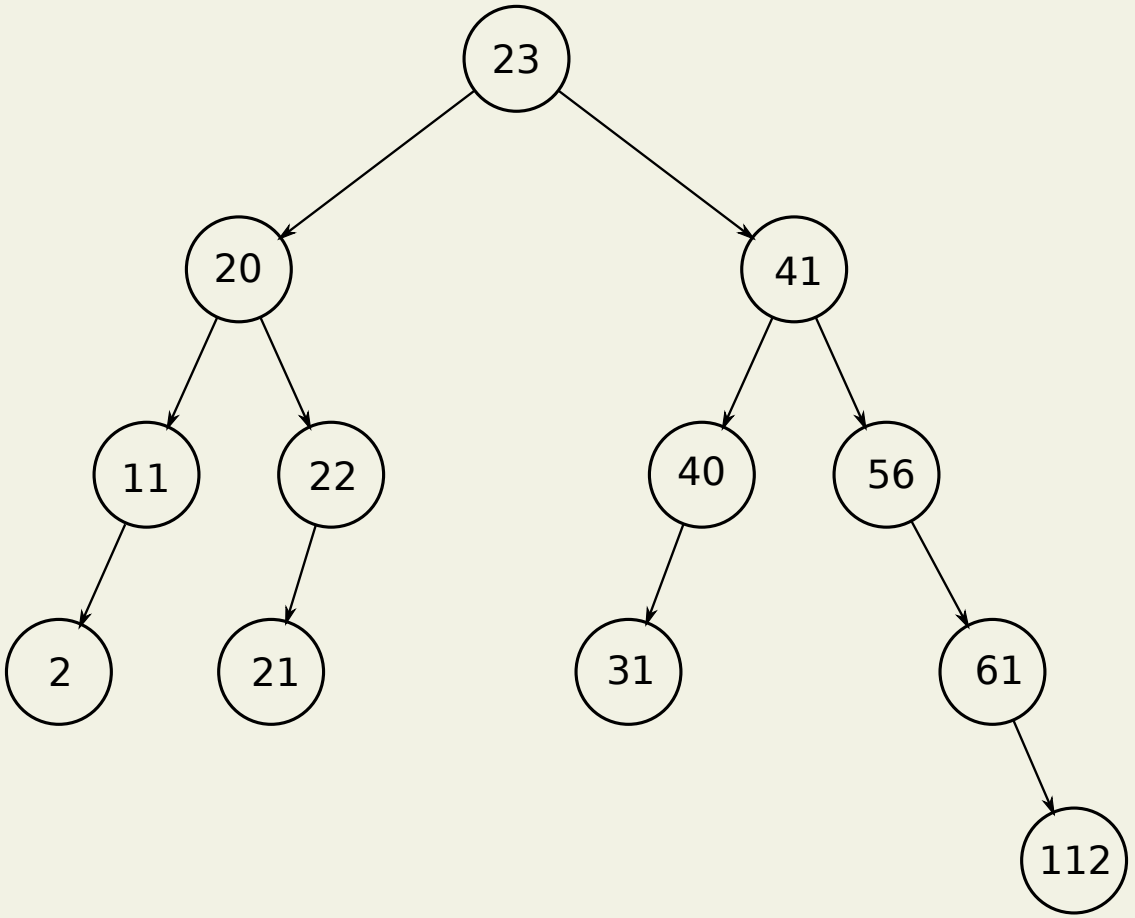
  Values in left subtree $\leq$ value(X) $\leq$ Value in right subtree

Abstract Data Type - BST

A BST supports the following functions:

- Insert(*node*, *val*) – Inserts *val* into the BST rooted at *node*.
- Search(*node*, *val*) – Returns True of *val* exists in the BST rooted at *node*. False otherwise.
- Succ(*val*) – Returns the smallest element greater than *val* in the BST.
- Pred(*val*) – Returns the largest element lesser than *val* in the BST.
- Delete(*val*) – Deletes *val* from the BST.

Example BST



Example BST

The order in which elements are inserted makes a difference!
Consider two different sequences of values:

- **Sequence A**: 23, 11, 20, 21, 2, 56, 40, 41
- **Sequence B**: 2, 11, 20, 21, 23, 40, 41

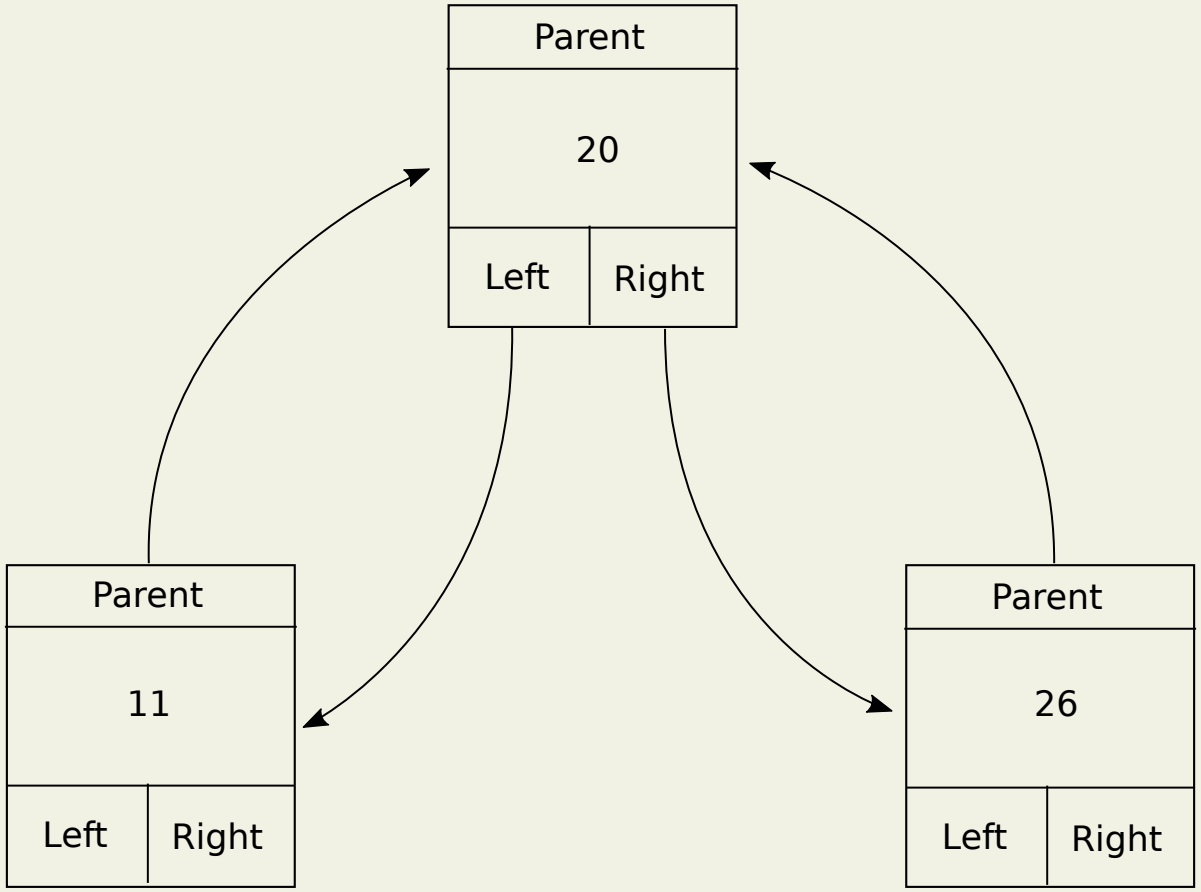
(See whiteboard).

Data Structure

BSTs are implemented using the Tree datastructure.
Similar to a node in a linked list, each node in the Tree has the following:

- int *val* – holds the data/value of the node.
- Left child pointer.
- Right child pointer.
- Parent node pointer.

Data Structure



Questions

1. What is the maximum height of a BST with n nodes?
2. What is the minimum height of a BST with n nodes?

Insert procedure

The Insert($node, x$) procedure:

- If $node = \text{NULL}$, create new node with x and attach to parent.
- Else If $x <$ value($node$),
  - Insert($node \rightarrow left, x$)
- Else If $x >$ value($node$) Then,
  - Insert($node \rightarrow right, x$)