CS202 - Algorithm Analysis

Tree Algorithms - Module 1

Aravind Mohan

Allegheny College

April 12, 2021

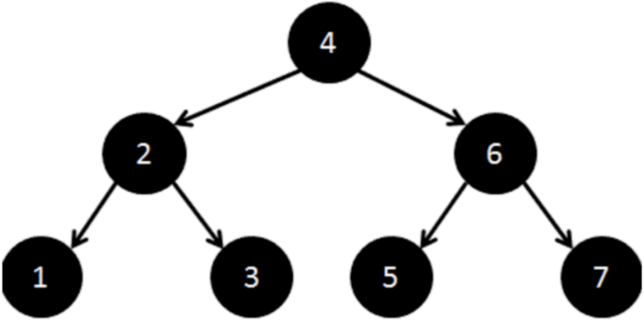
Discussion Based On ...

Sedgewick 2.4 Heap Sort

Data Structures - An overview

- So far we have seen linear structures:
  - linear: before and after relationship
  - Arrays, Stacks, and Queues
- Non-linear structure: **Trees**
  - probably the most fundamental structure in computing
  - hierarchical structure
  - Terminology: from family trees (genealogy)

Trees



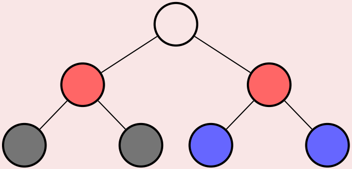
Trees More Formally

- **Definition:** A tree T is a set of nodes storing elements such that the nodes have a parent-child relationship that satisfies the following properties:
  - If T is nonempty, it has a special node, called the root of T, that has no parent.
  - Each node v of T different than the root has a unique **parent node** w; every node with parent w is a **child** of w

Trees More Formally

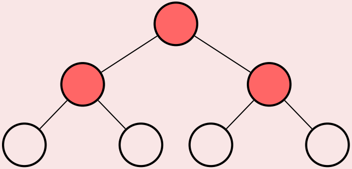
- **Recursive Definition:**
  - T is either empty
  - or consists of a node r, called the root of T, and a (possibly empty) set of trees whose roots are the children of r.

- **Siblings:** Two nodes that have the same parent are called siblings.



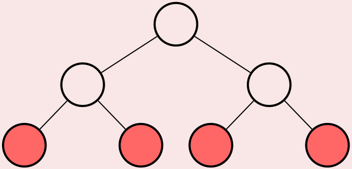
Trees - Some basic terminologies

- **Internal nodes:** Nodes that have one or more children(s).



Trees - Some basic terminologies

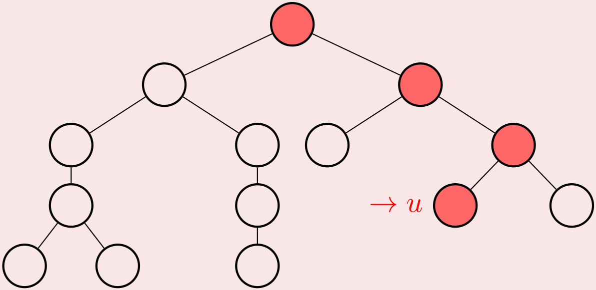
- **External nodes:** Nodes that don't have any children.



- **Ancestors:**

  Ancestors of a node u are u itself and the ancestors of its parent.

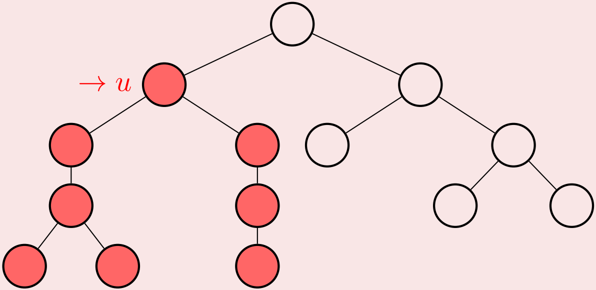
  (INCLUSIVE)



- **Descendants:**

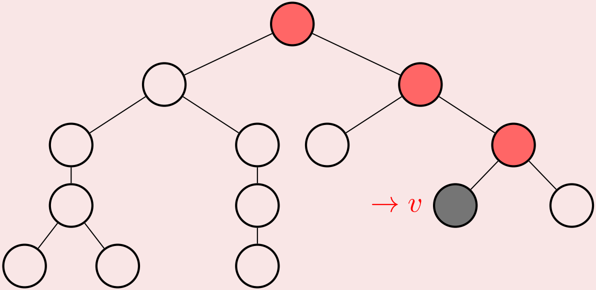
  v is a descendants of u if u is an ancestor of v.

  (INCLUSIVE)



- **Depth(T, v):**

  Number of ancestors of v, excluding v itself.

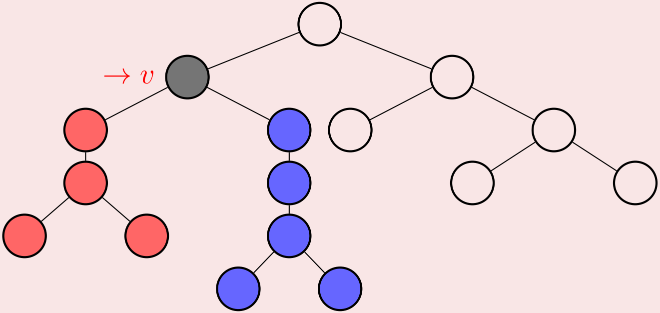


Depth(T, v) = 3

Trees - Some basic terminologies

- **Height(T, v):**

  Number of nodes in the longest path from v to any leaf, excluding v itself.



Height(T, v) = 4

Trees - Some basic terminologies

- What is the height of the leaf node(s)?

- The height of a tree is the height of its root.
- Height and Depth are symmetrical.

Proposition: The **height** of a tree T is the **maximum depth** of one of its leaves.

Trees - Applications

- Scheduling and Priority Queue (Heap)
- Class Hierarchy in Java
- File System
- Storing hierarchies in organizations

Binary Trees More Formally

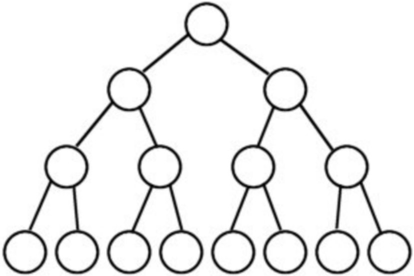
- **Definition:** A binary tree is a tree such that:
  - every node has at most 2 children
  - each node is labeled as being either a left chilld or a right child

Binary Trees More Formally

- **Recursive Definition:**
  - a binary tree is empty;
  - or it consists of
    - a node (the root) that stores an element
    - another binary tree, called the left subtree of T
    - another binary tree, called the right subtree of T

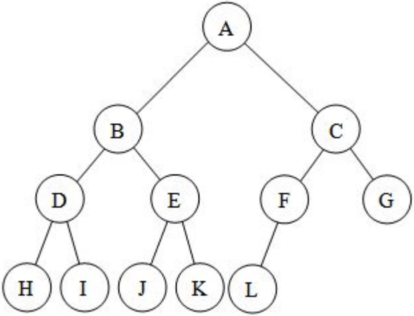
Binary Tree Examples

- A full binary tree (sometimes complete or proper binary tree or 2-tree) is a tree in which every node other than the leaves has two children.



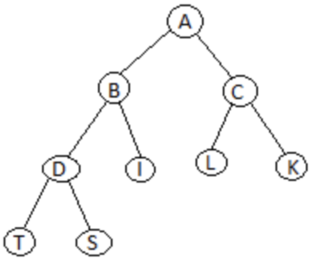
Binary Tree Examples

- An atmost complete binary tree is a binary tree in which every level, except possibly the last, is completely filled, and all nodes are as far left as possible.

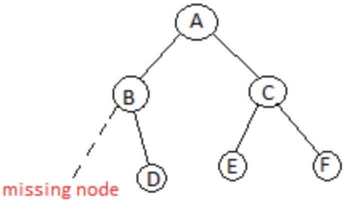


Binary Tree Examples

- An In-Complete binary tree is a binary tree in which the properties of the complete binary tree is not true.



Complete Binary Tree



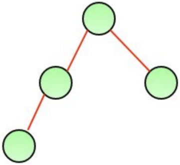
In-Complete Binary Tree

Binary Tree Examples

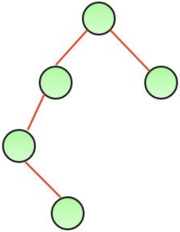
- **Balanced** : Difference between the height of the left and right subtree is atmost 1.
- **Unbalanced** : Difference between the height of the left and right subtree is greater than 1.

Depend on the balancing scheme. Later.

- **Balanced Vs Unbalanced**



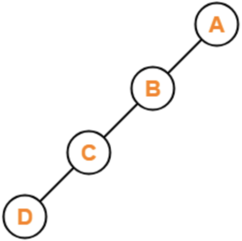
A height balanced tree



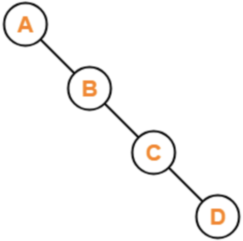
Not a height balanced tree

Binary Tree Examples

- A skewed binary tree is a binary tree that satisfies the following 2 properties:
  1. All the nodes except one node has one and only one child.
  2. The remaining node has no child.



Left Skewed Binary Tree



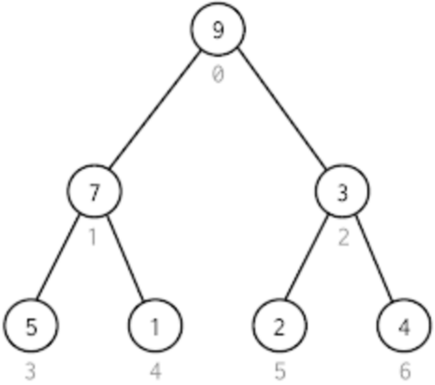
Right Skewed Binary Tree

Properties of Binary Trees

- In a binary tree
  - level 0 has $<= 1$ node
  - level 1 has $<= 2$ nodes
  - level 2 has $<= 4$ nodes
  - $\cdots$
  - level i has $<= 2^{i}$ nodes

How to store a binary tree in a program?

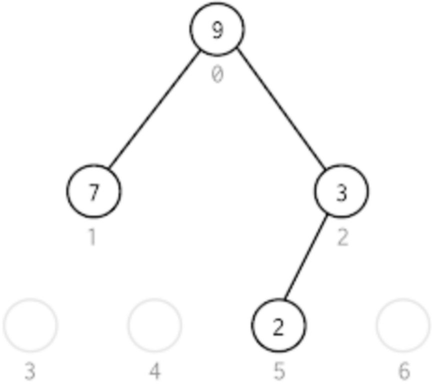
- An array can be used to represent the binary tree structure.



9	7	3	5	1	2	4
0	1	2	3	4	5	6

How to store a binary tree in a program?

- An array can be used to represent the binary tree structure.

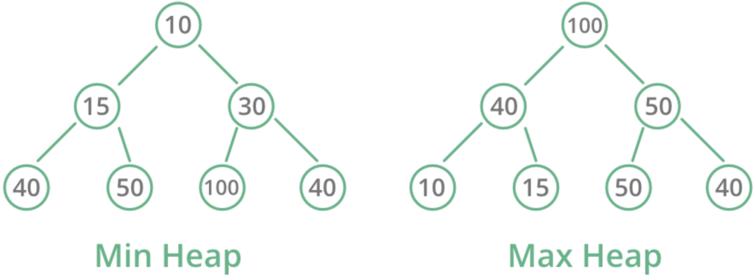


What is a Binary Heap?

- Each node has atmost two children.
- Complete binary tree or atmost complete binary tree qualified as binary heap.
- Node with no children is also qualified as heap.
- Left skewed or right skewed tree is not a heap.
- There are two types of heap, namely:

  Max heap and Min heap.

What is a Binary Heap?



Binary Heap Properties

- Binary Heap has two main properties:
  1. Order property
  2. Shape property

Binary Heap Properties

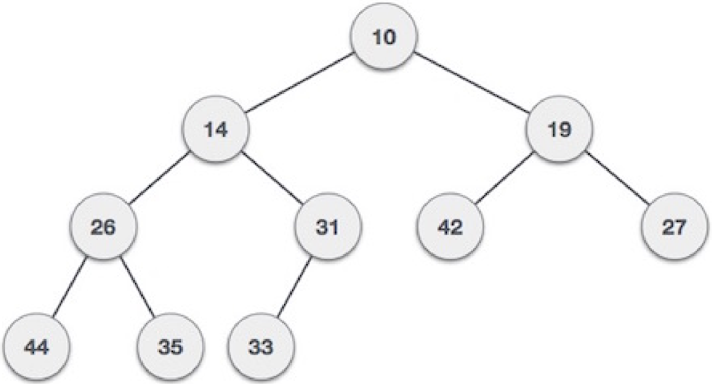
- **Order** property: The value in node n is $\geq$ the values in its children, for every node n (MAX heap).
- How about MIN heap?

Binary Heap Properties

Shape Property:

- All leaves are either at depth d or $d-1$ for some d
- All of the leaves at depth $d-1$ are to the right of the leaves at depth d
- And the following:
  1. There is at most 1 node with just 1 child v.
  2. v is the left child of its parent.
  3. v is the rightmost leaf at depth d.

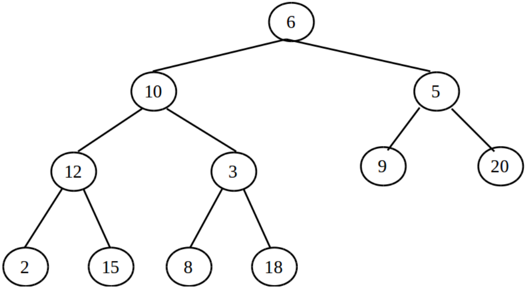
Binary Heap Example



- Phase 1: convert the array into an n-element heap
- Phase 2: repeatedly remove maximum element from the heap, and place that element in its proper position in the array
  - swap element at 0th position with element at $(n-1)$th position and then "reheapify" considering only the first $n-1$ elements
  - repeat this process until heap size is reduced to 1 (minimum element remains, at 0th position)

Heap Sort

6 10 5 12 3 9 20 2 15 8 18

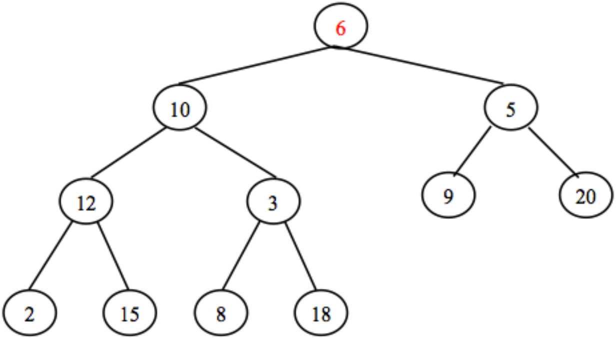


Heap Sort: Phase 1 - build the heap

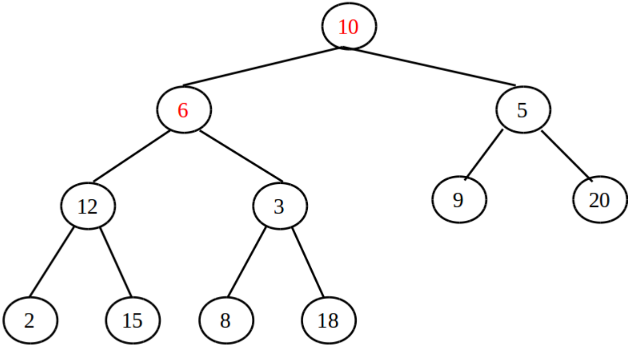
```
for i = 1 to n-1 do
  insert element s[i] into the heap consisting
   of the elements s[0]...s[i-1]
```

Once the heap is built, $s[0]$ will contain the maximum element

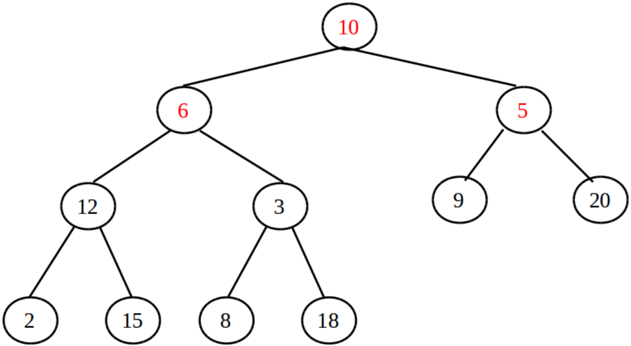
Phase 1 - build the Heap



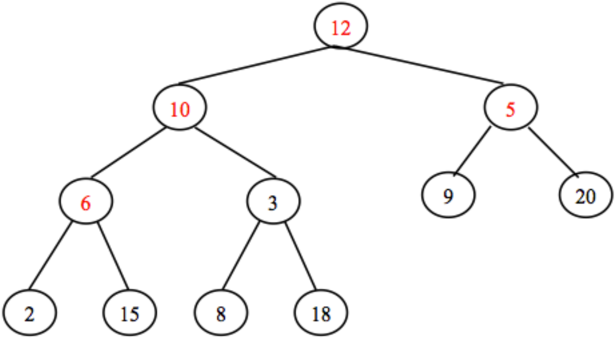
Phase 1 - build the Heap



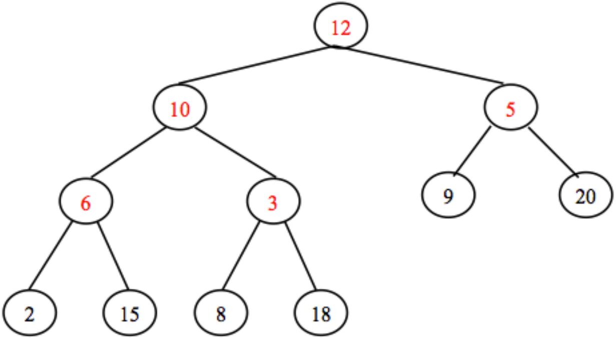
Phase 1 - build the Heap



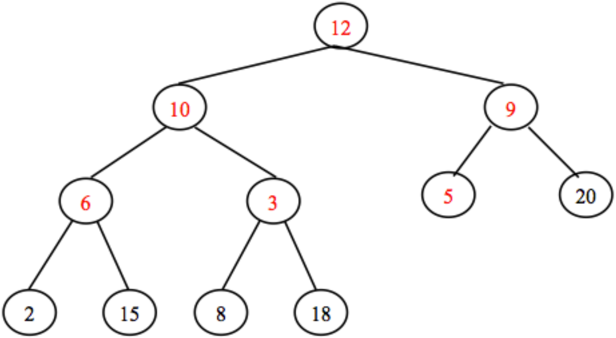
Phase 1 - build the Heap



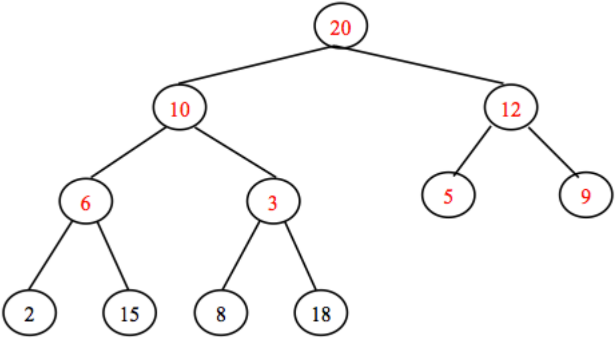
Phase 1 - build the Heap



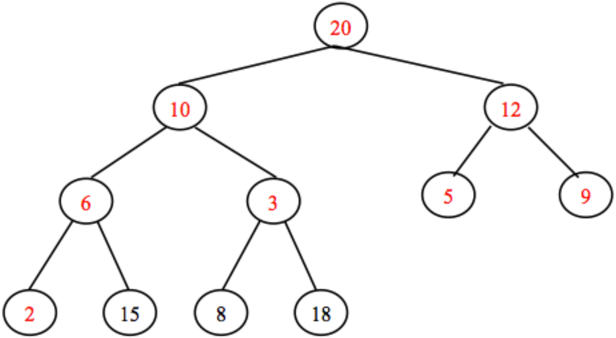
Phase 1 - build the Heap



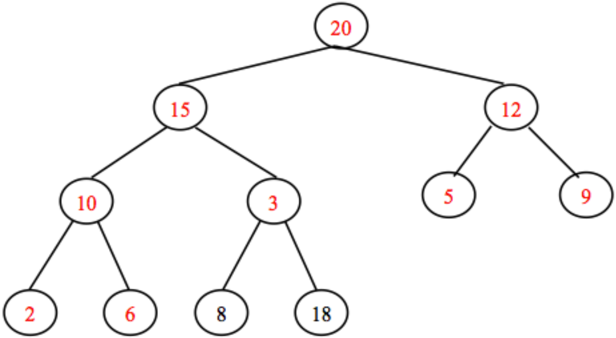
Phase 1 - build the Heap



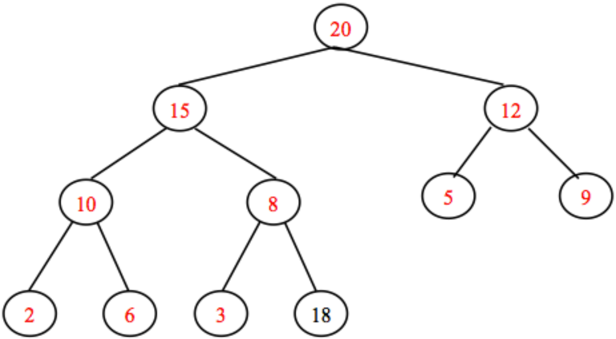
Phase 1 - build the Heap



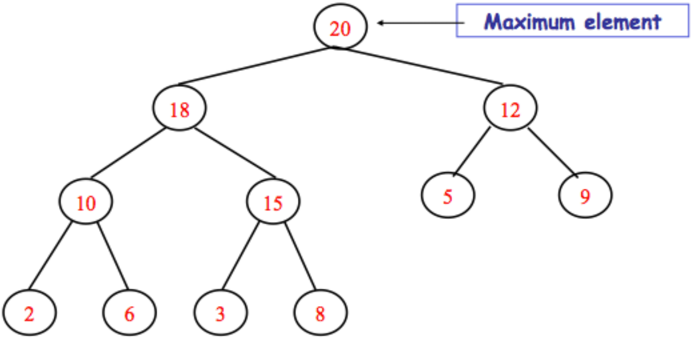
Phase 1 - build the Heap



Phase 1 - build the Heap



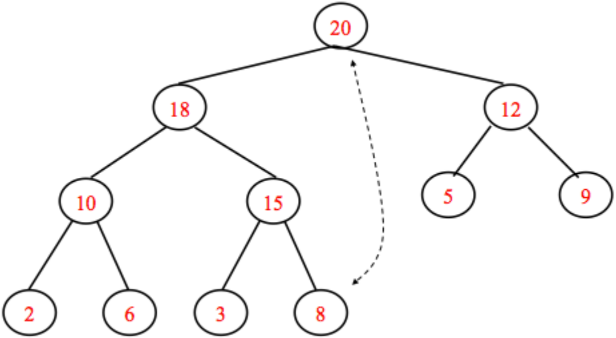
Phase 1 - build the Heap



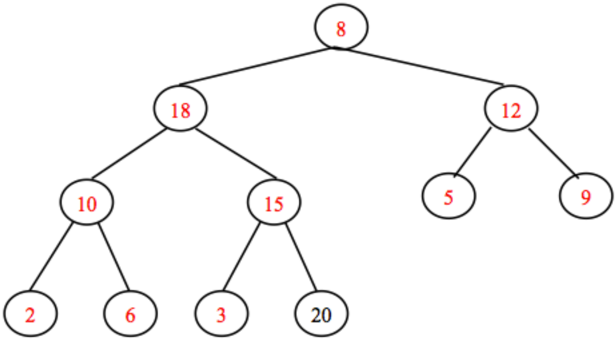
Phase 1 - build the Heap

```
for i = n − 1 to 1 do
  swap s[0] and s[i]
  demote s[0] to its proper place
  in the heap consisting of the elements
s[0]...s[i − 1]
```

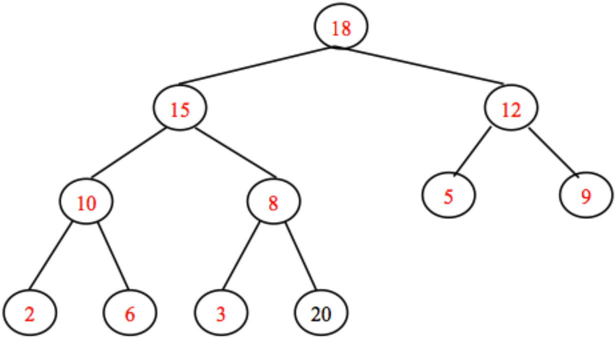
Phase 2 - select max



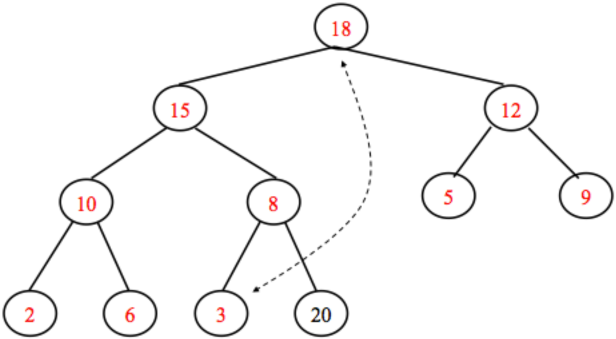
Phase 2 - select max



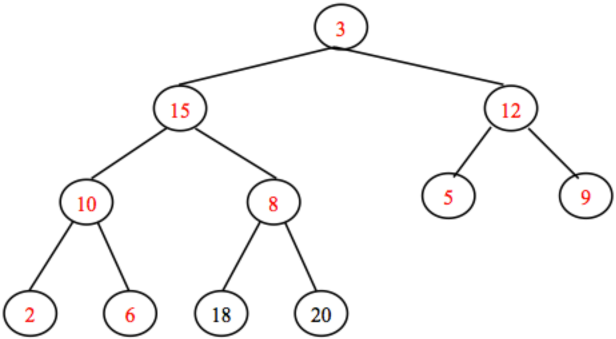
Phase 2 - select max



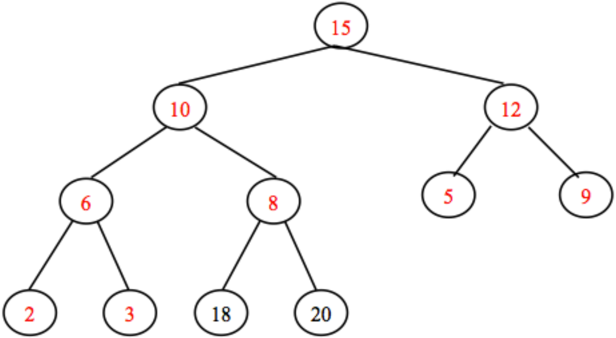
Phase 2 - select max



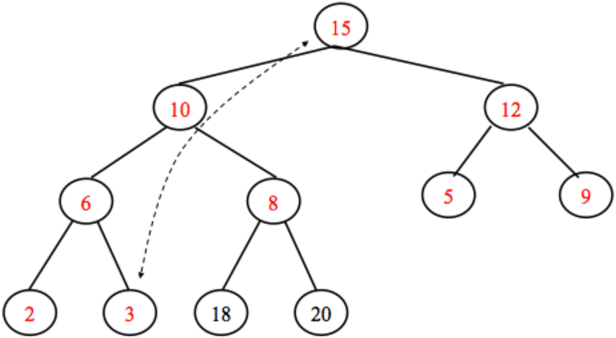
Phase 2 - select max



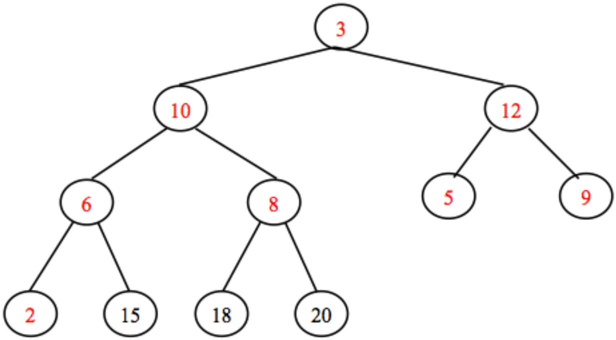
Phase 2 - select max



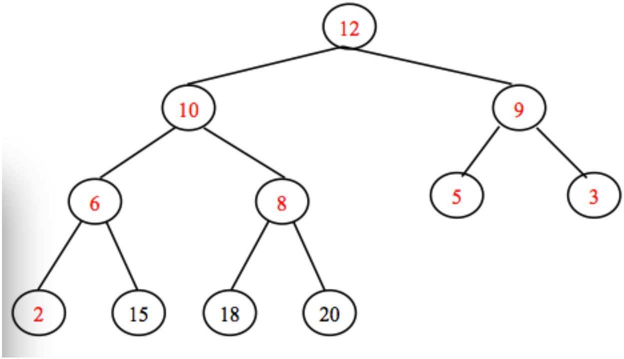
Phase 2 - select max



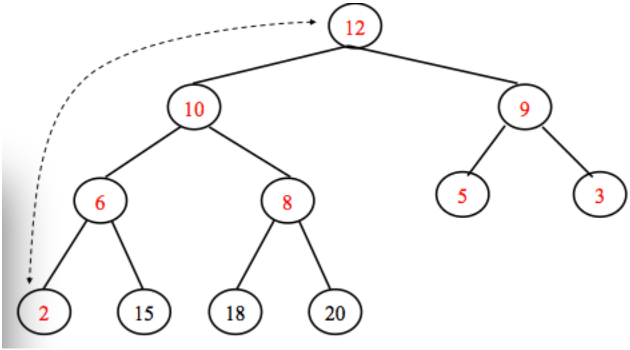
Phase 2 - select max



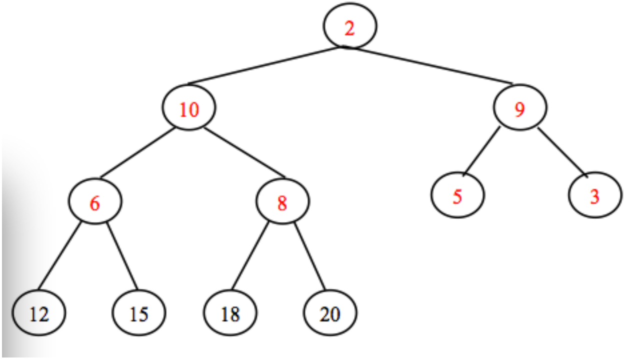
Phase 2 - select max



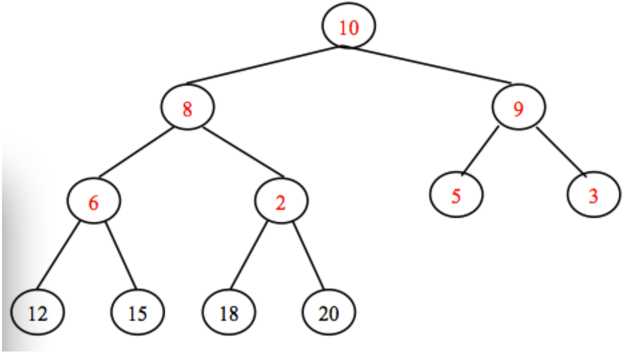
Phase 2 - select max



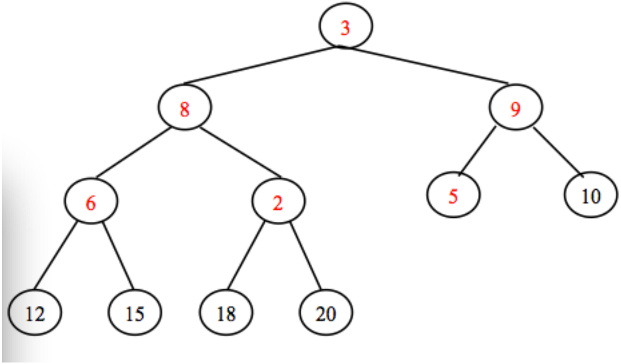
Phase 2 - select max



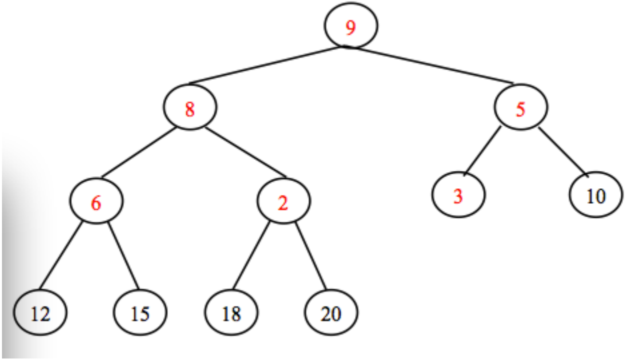
Phase 2 - select max



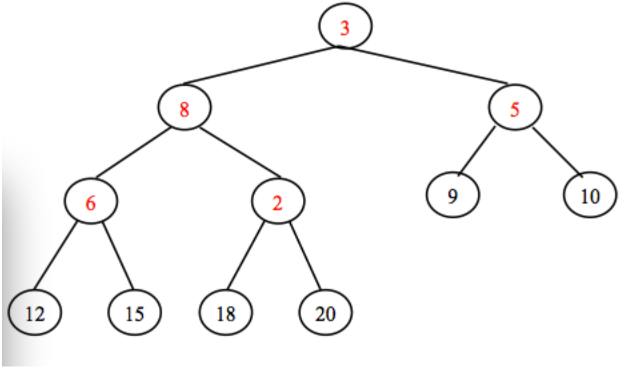
Phase 2 - select max



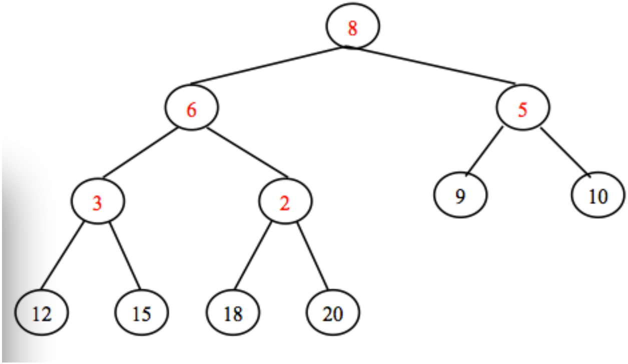
Phase 2 - select max



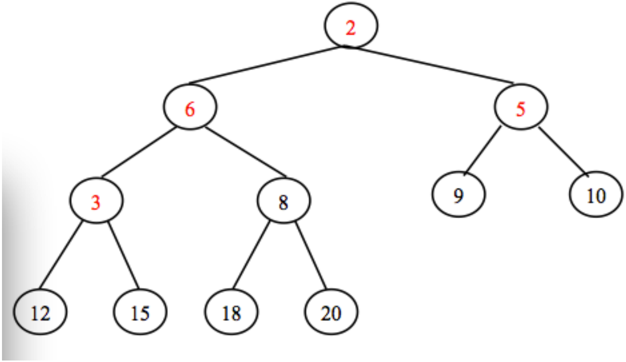
Phase 2 - select max



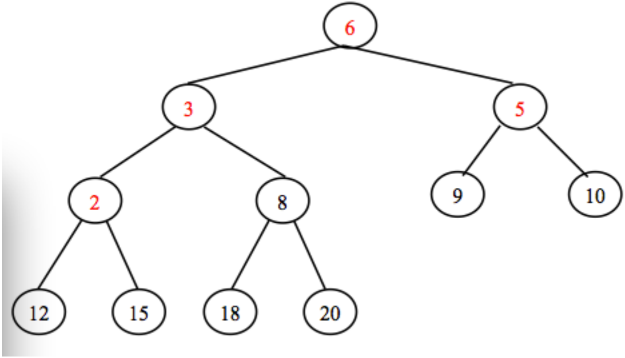
Phase 2 - select max



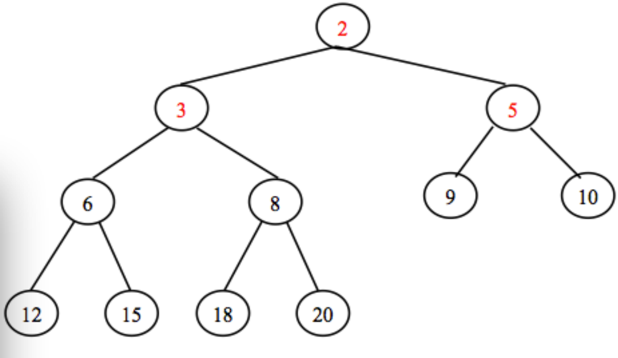
Phase 2 - select max



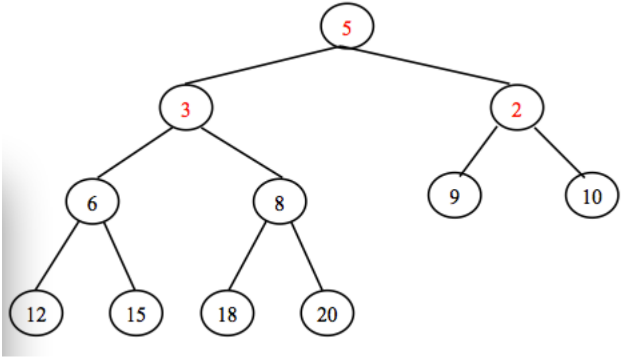
Phase 2 - select max



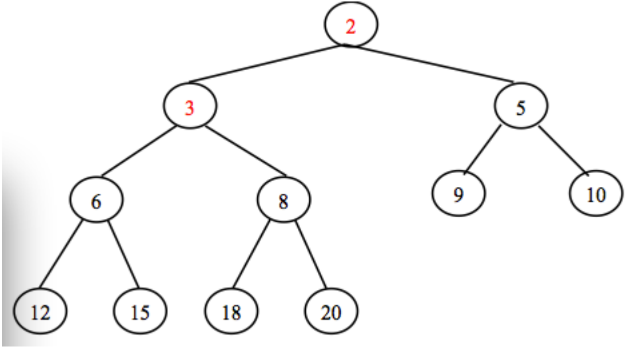
Phase 2 - select max



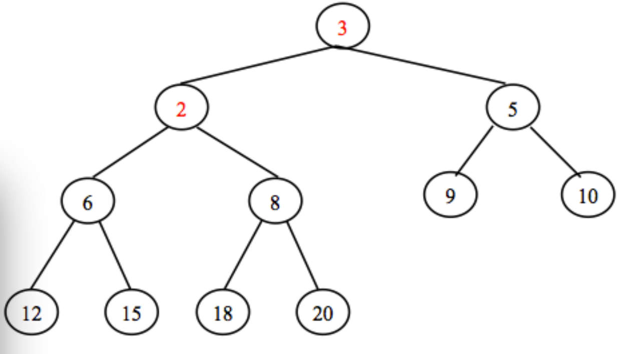
Phase 2 - select max



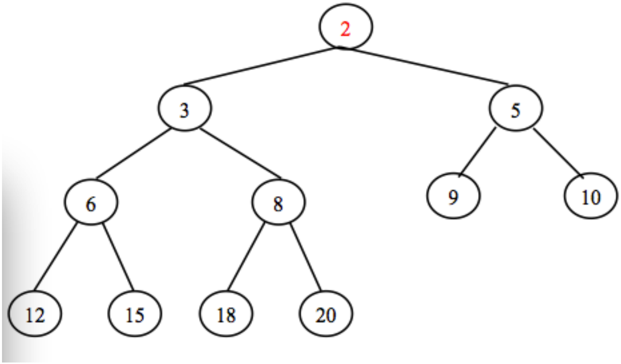
Phase 2 - select max



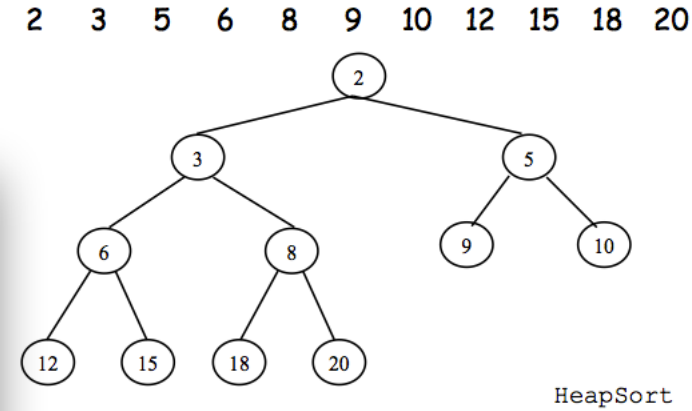
Phase 2 - select max



Phase 2 - select max



Heap Sort Completed



Heap Sort Complexity

```
for i ← 1 to n-1 do
   insert element s[i] into the
     heap consisting of the
     elements s[0]...s[i-1]
```

O(n log n)

O(log n) operations

```
for i ← n-1 down to 1 do
   swap s[0] and s[i]
   "demote" s[0] to its proper place
     in the heap consisting of the
     elements s[0]...s[i-1]
```

O(n log n)

Heap Sort

Note that heap sort is just a more clever version of selection sort since a maximum is repeatedly selected and placed in its proper position

Sorting Algorithms - Comparison

Algorithm	Time	Notes
selection-sort	$O(n^2)$	slow in-place for small data sets (< 1K)
insertion-sort	$O(n^2)$	slow in-place for small data sets (< 1K)
heap-sort	$O(n \log n)$	fast in-place for large data sets (1K — 1M)
merge-sort	$O(n \log n)$	fast sequential data access for huge data sets (> 1M)

Reading Assignment

Sedgewick 2.4 Heap Sort

Questions?

Please ask if there are any Questions!