Heaps

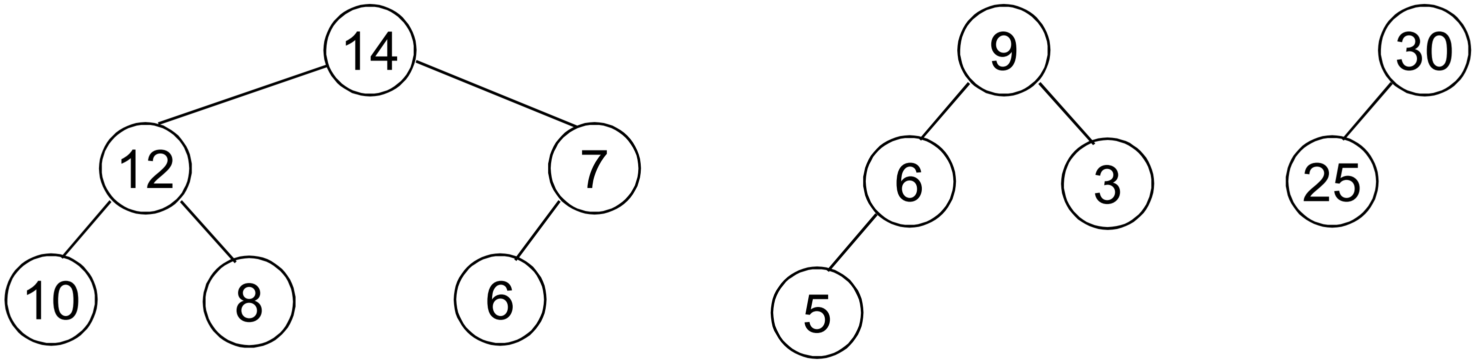
Prof. Ki-Hoon Lee
Dept. of Computer Engineering
Kwangwoon University

Heap

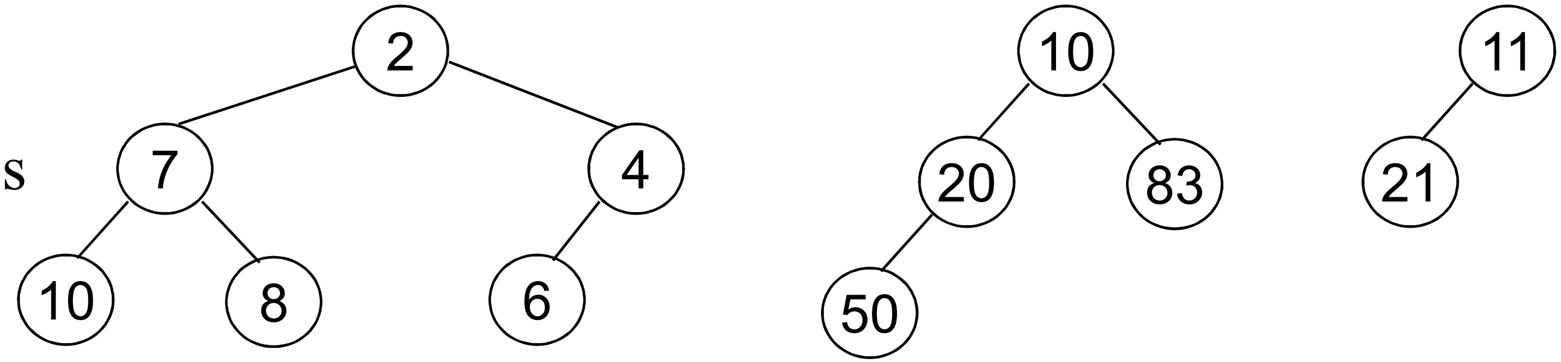
- A *max* (*min*) *tree* is a tree in which the key value in each node is no smaller (larger) than the key values in its children (if any)
- The key in the root of a max (min) tree is the largest (smallest) key in the tree
- A *max* (*min*) *heap* is a complete binary tree that is also a max (min) tree

Heap (cont.)

max heaps



min heaps



Heap (cont.)

- Basic Operations
  - Creation of an empty heap
  - Insertion
  - Deletion of the root

Heap Height

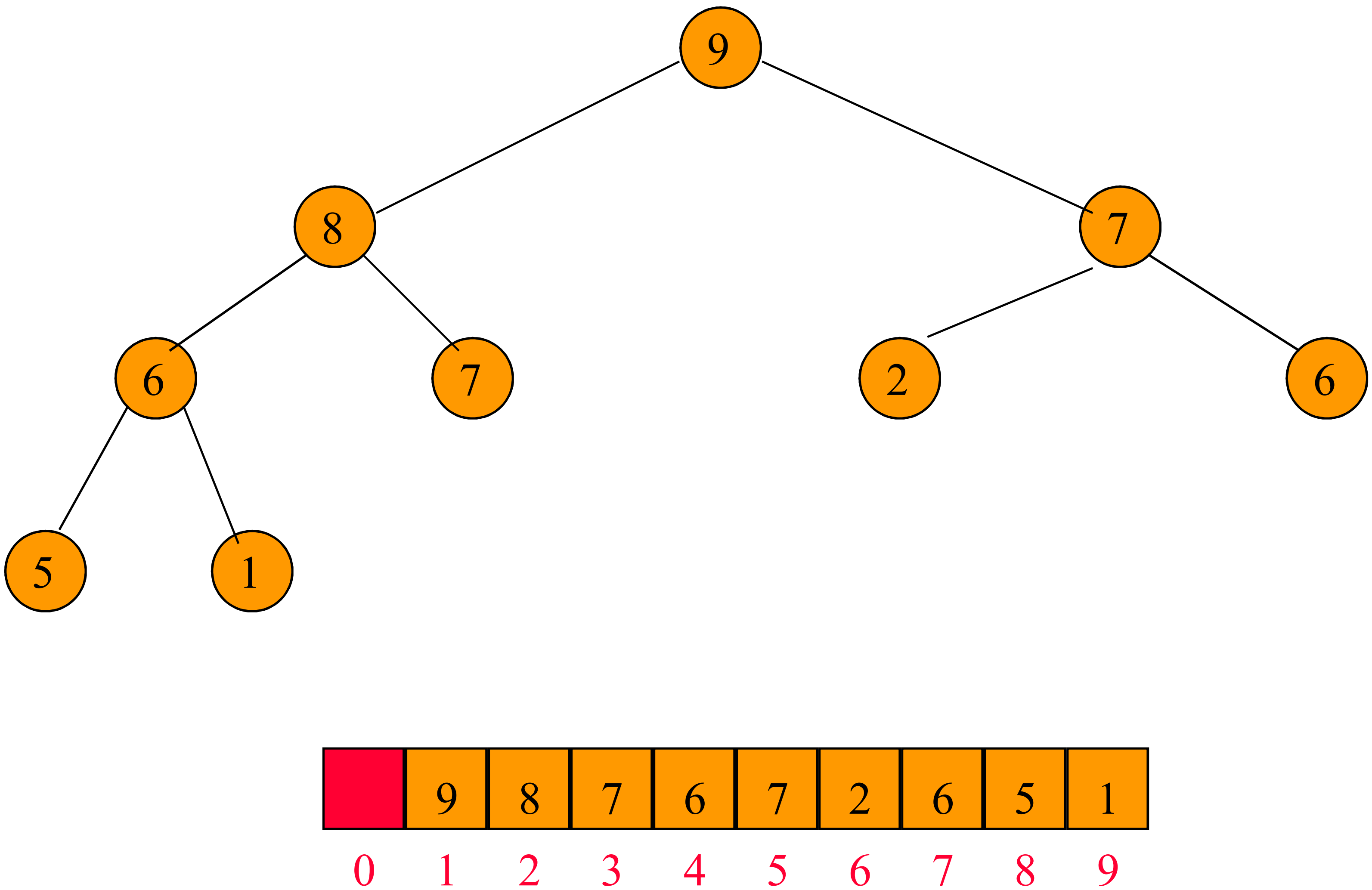
- Since a heap is a complete binary tree, the height of an *n* node heap is $\lceil \log_2(n+1) \rceil$

Maximum number of nodes

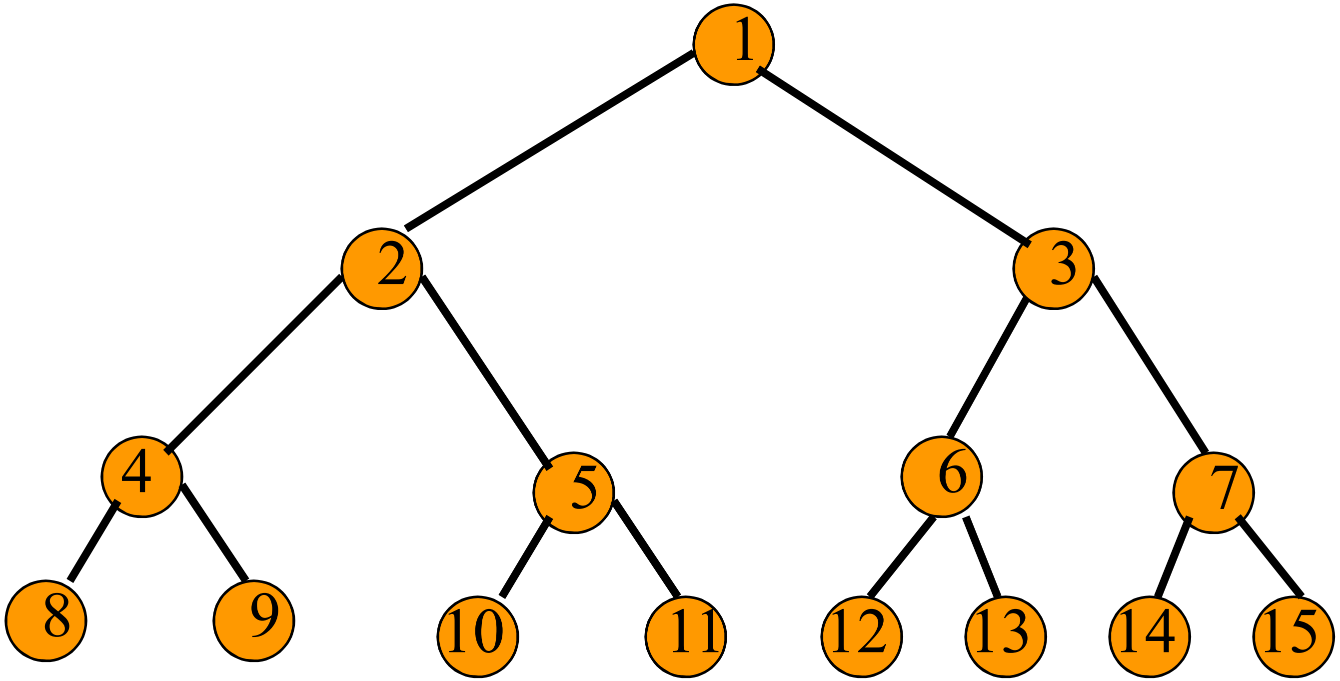
$= 1 + 2 + 4 + 8 + \ldots + 2^{h-1}$

$= 2^h - 1$

A Heap Represented as an Array

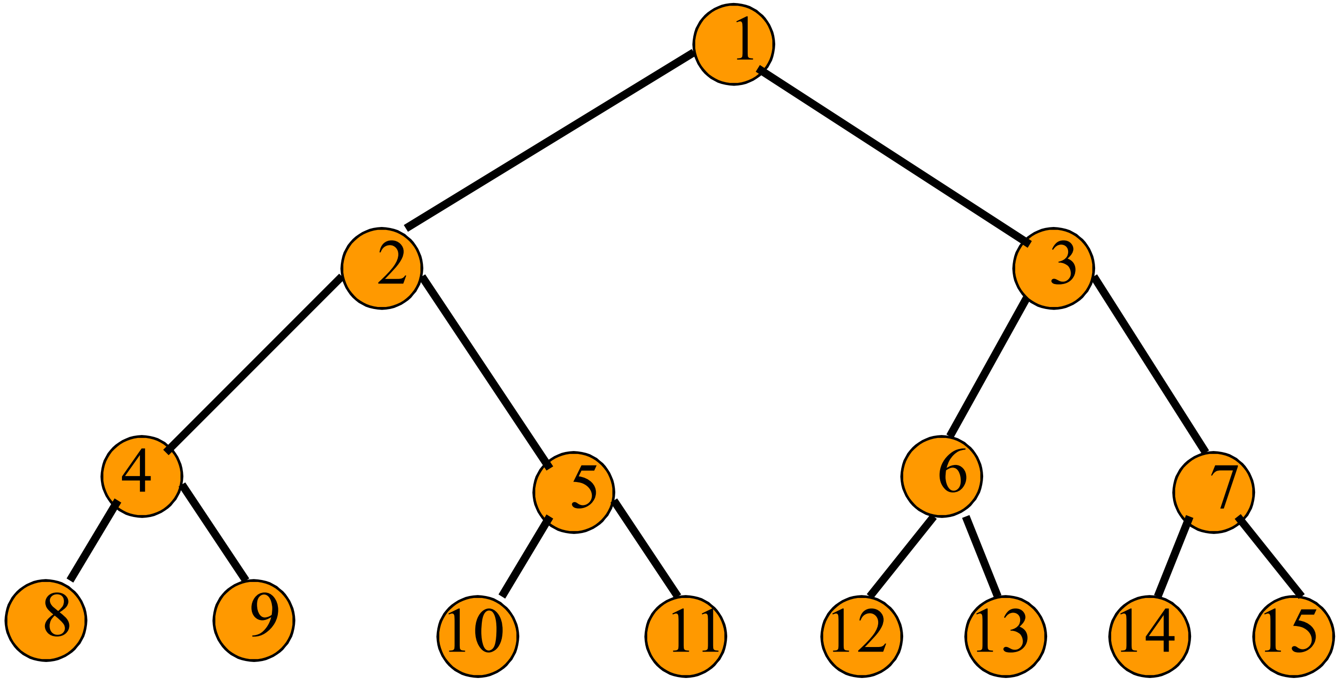


Node Number Properties



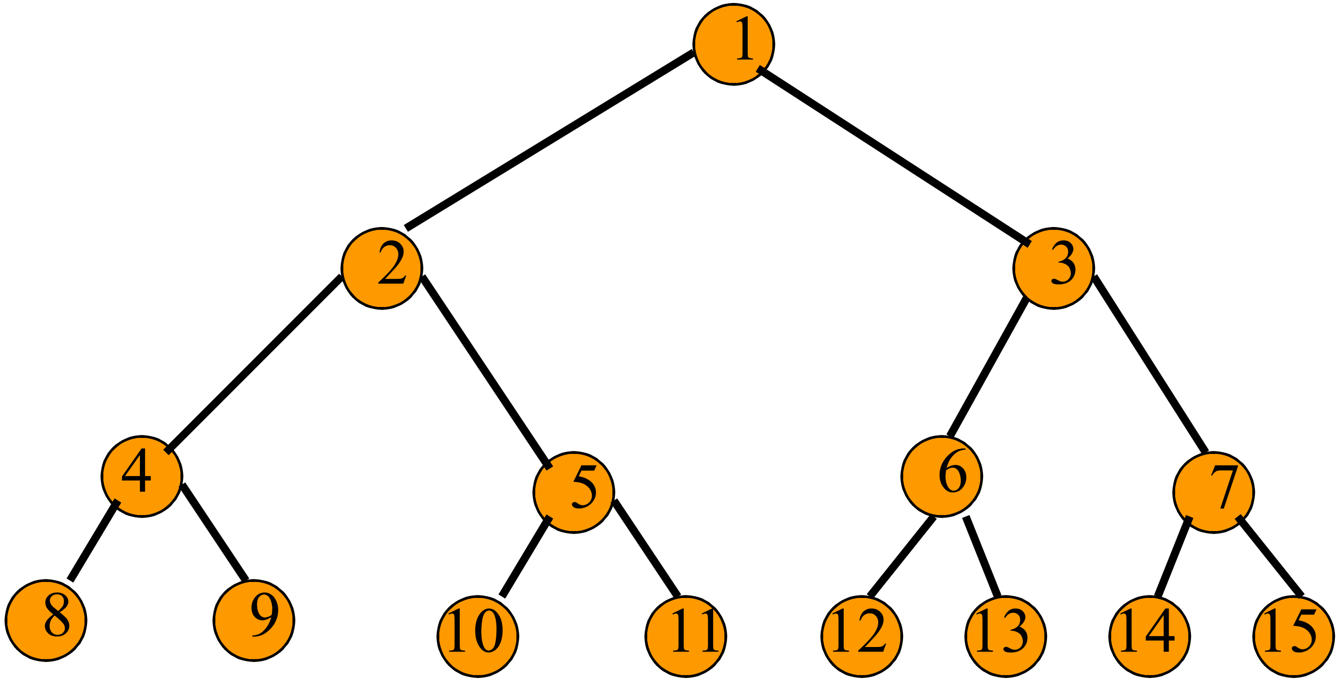
- Parent of node i is node i / 2, unless $i = 1$.
- Node 1 is the root and has no parent.

Node Number Properties (Cont.)



- Left child of node i is node $2i$, unless $2i > n$, where n is the number of nodes.
- If $2i > n$, node i has no left child.

Node Number Properties (Cont.)



- Right child of node i is node $2i+1$, unless $2i+1 > n$, where n is the number of nodes.
- If $2i+1 > n$, node i has no right child.

Template Class MaxHeap

```
private:
  T *heap;          // element array
  int heapSize; // number of elements in heap
  int capacity;  // size of the array heap
```

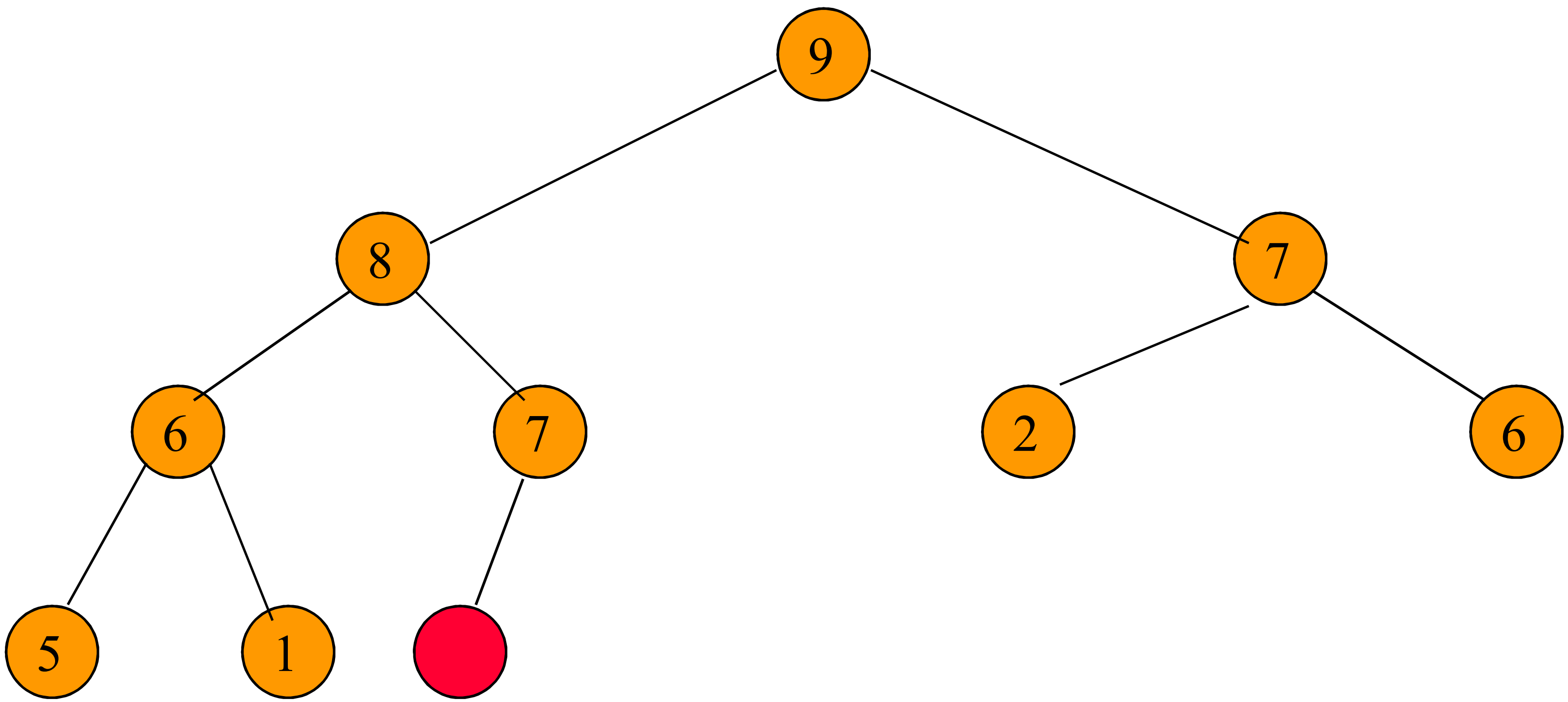
Template Class MaxHeap (cont.)

```
template <class T>
MaxHeap<T>::MaxHeap (int theCapacity = 10)
{
  if (theCapacity < 1) throw "Capacity must be >= 1.";
  capacity = theCapacity;
  heapSize = 0;
  heap = new T[capacity + 1]; // heap[0] is not used
}
```

Insertion

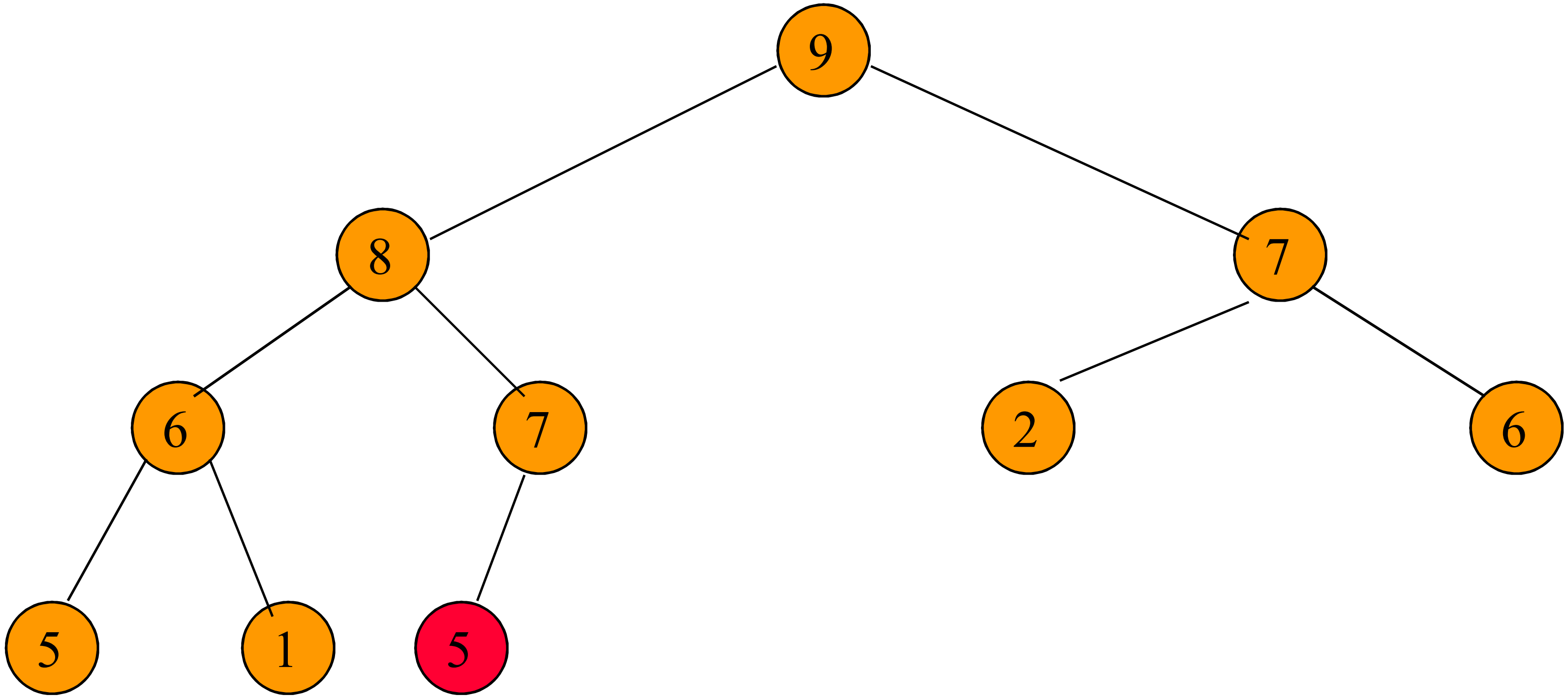
- To determine the correct place for the element being inserted, we use a *bubbling up* process
- The bubbling up process begins at a new leaf node and moves up toward the root
- The element to be inserted bubbles up as far as is necessary to ensure a max (min) heap

Inserting an Element into a Max Heap



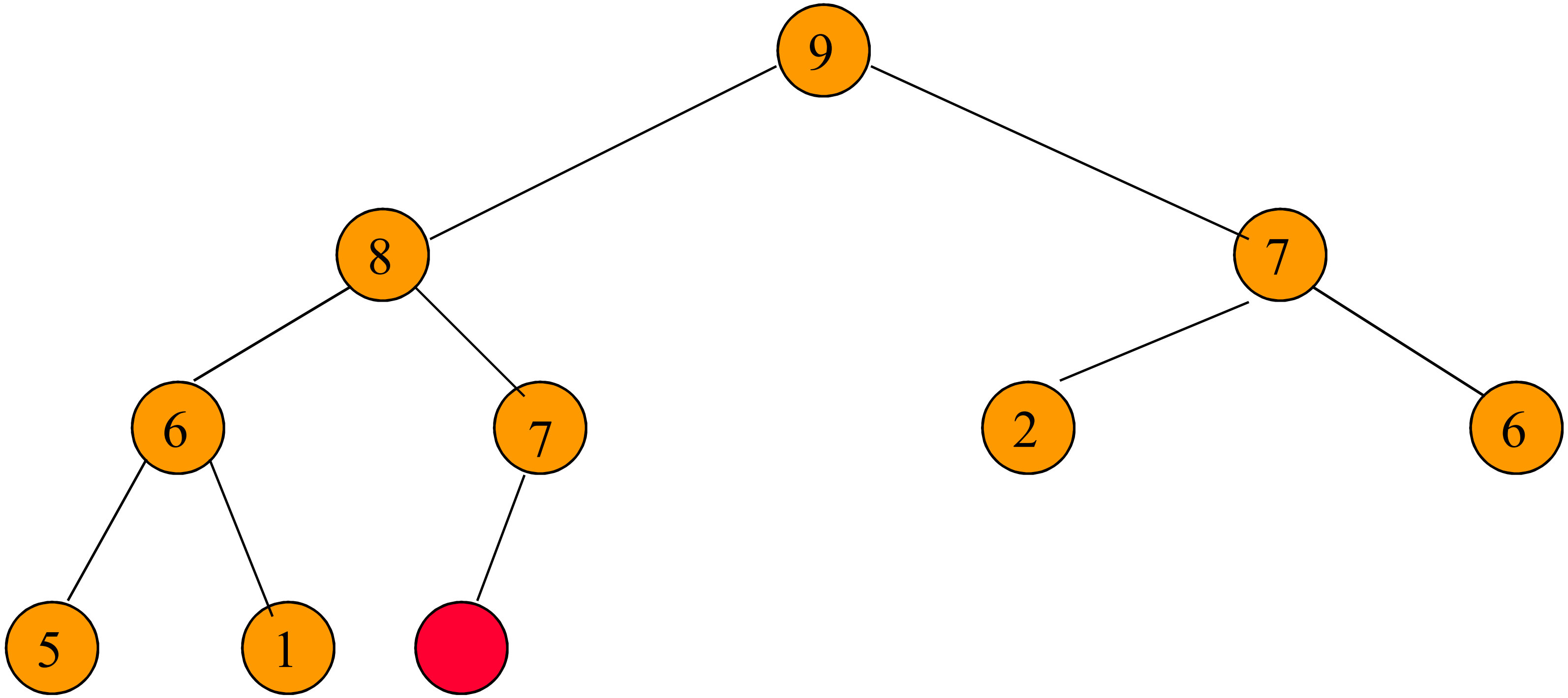
Complete binary tree with 10 nodes.

Inserting an Element into a Max Heap



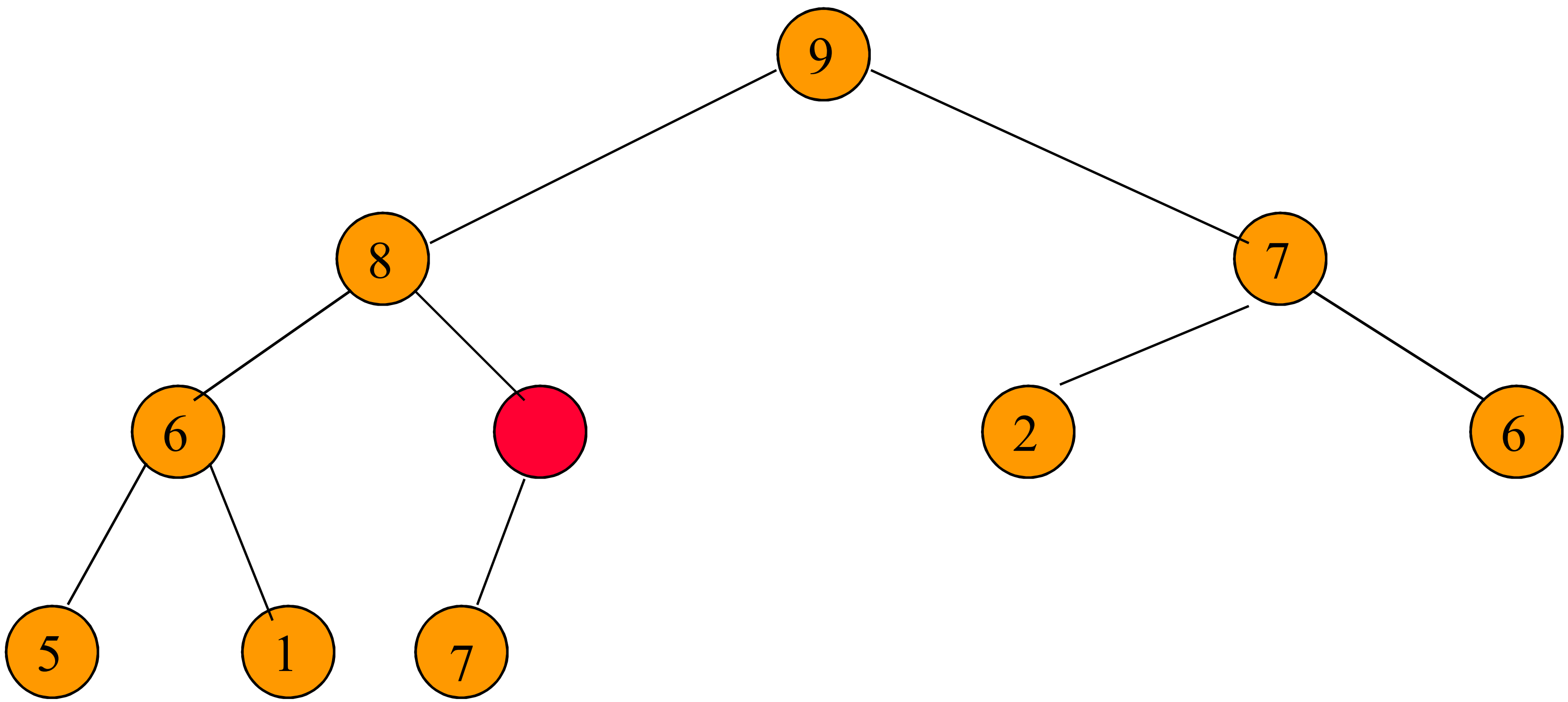
New element is 5.

Inserting an Element into a Max Heap



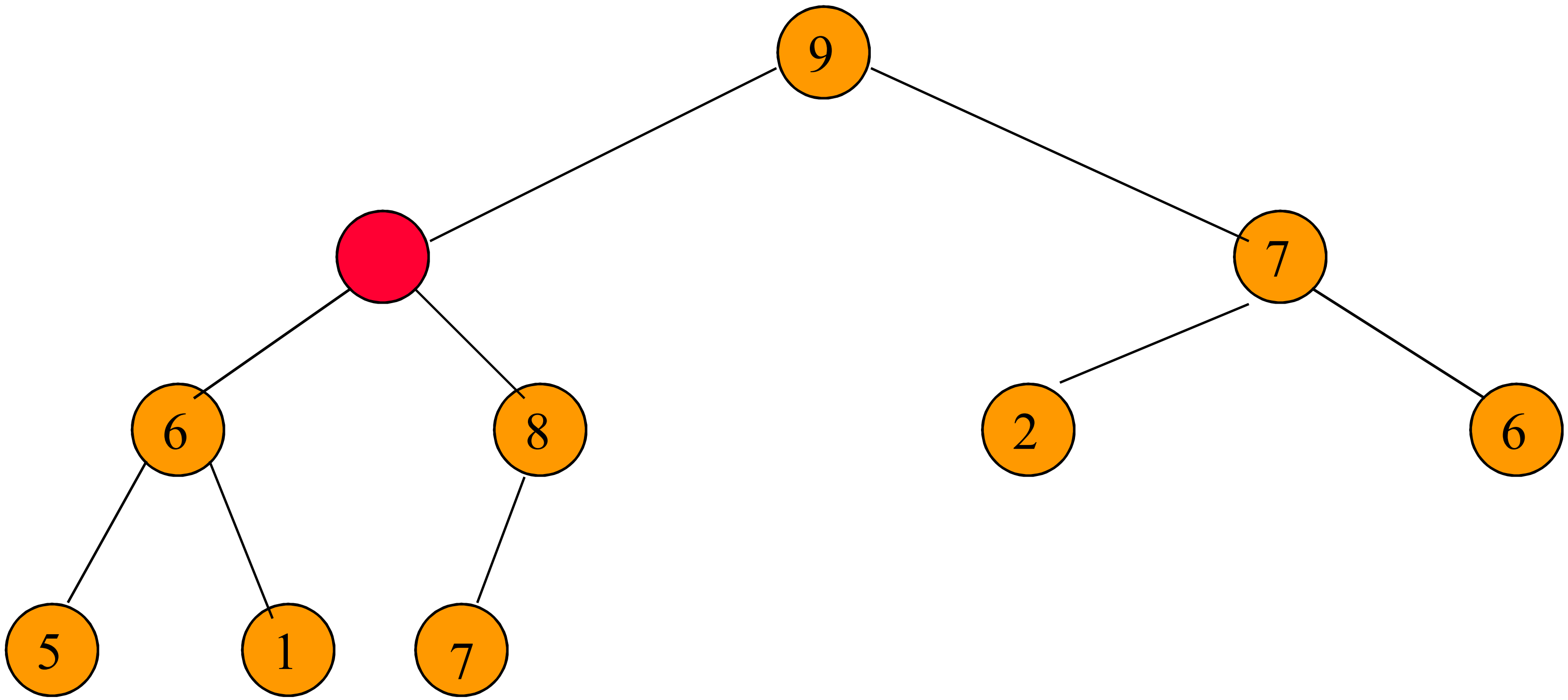
New element is 20.

Inserting an Element into a Max Heap



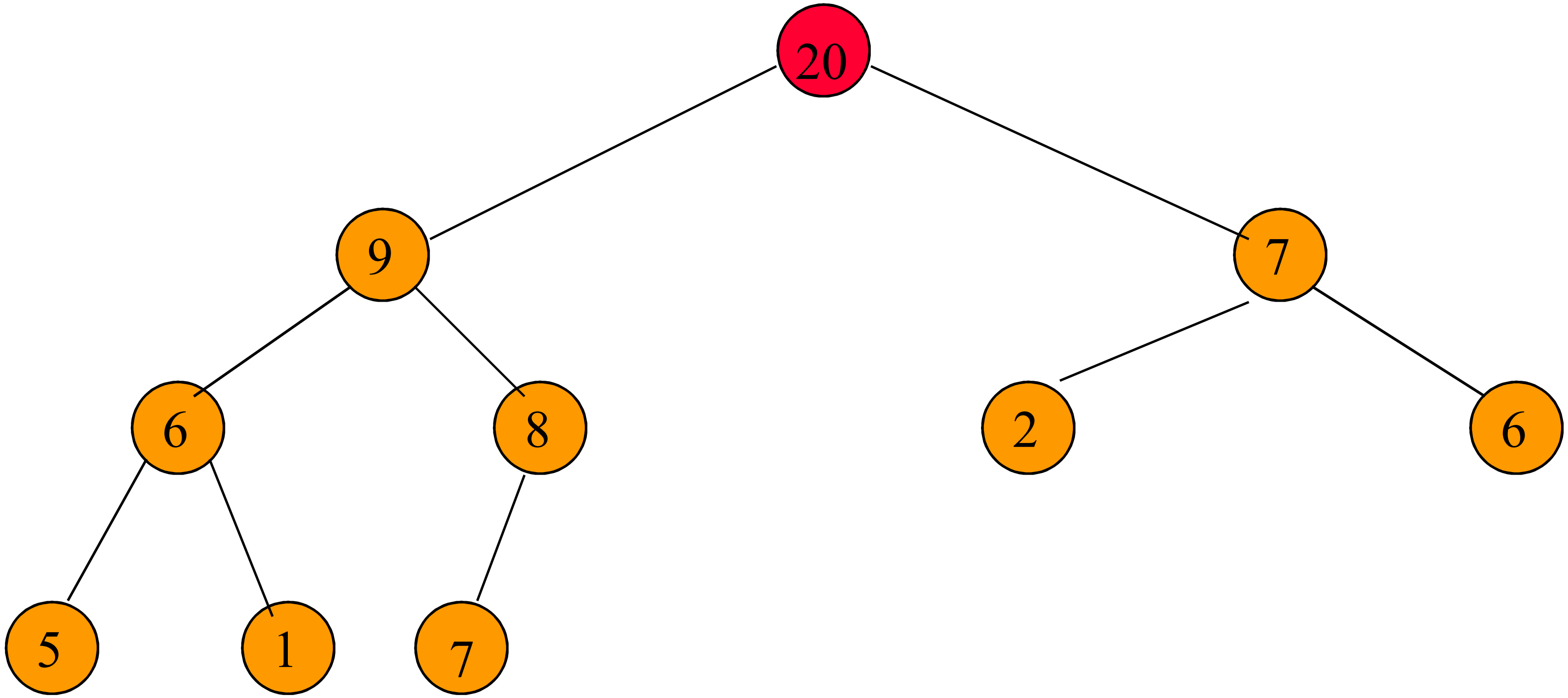
New element is 20.

Inserting an Element into a Max Heap



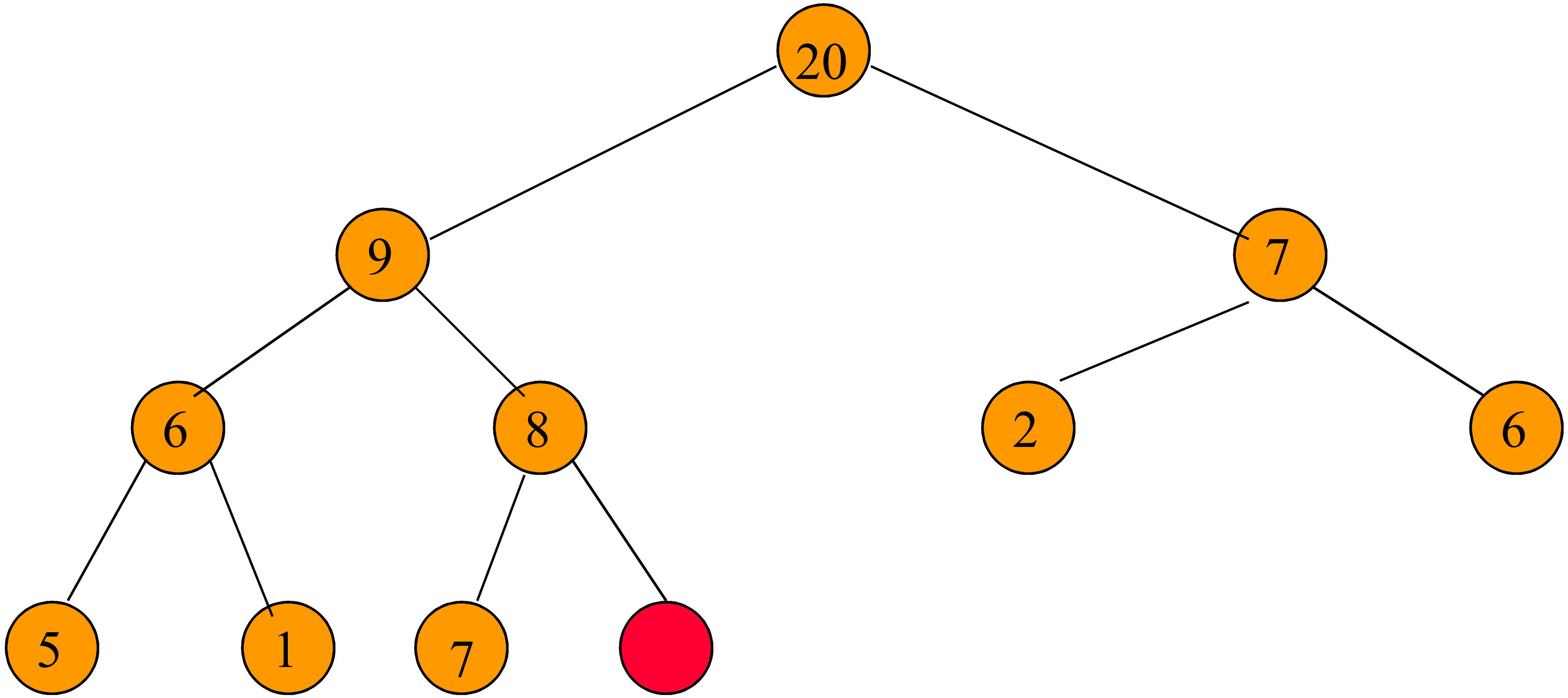
New element is 20.

Inserting an Element into a Max Heap



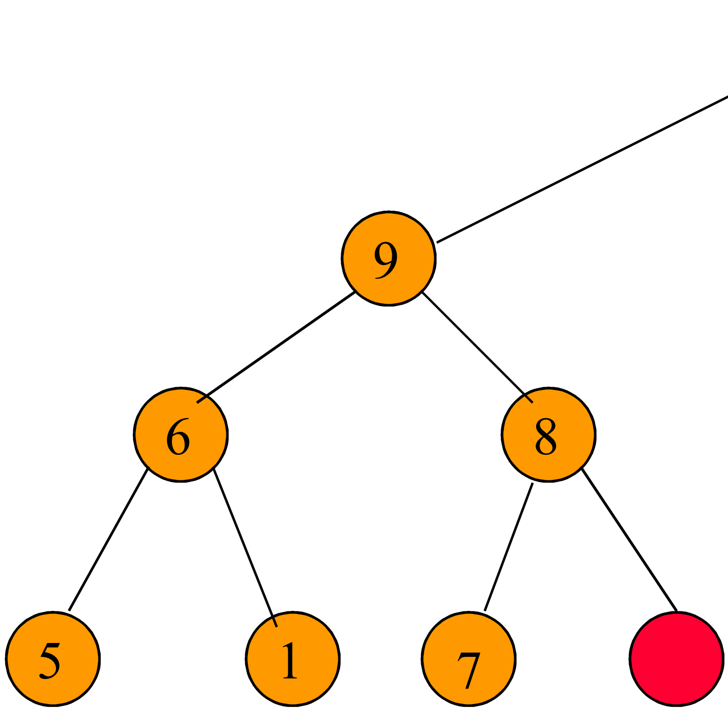
New element is 20.

Inserting an Element into a Max Heap



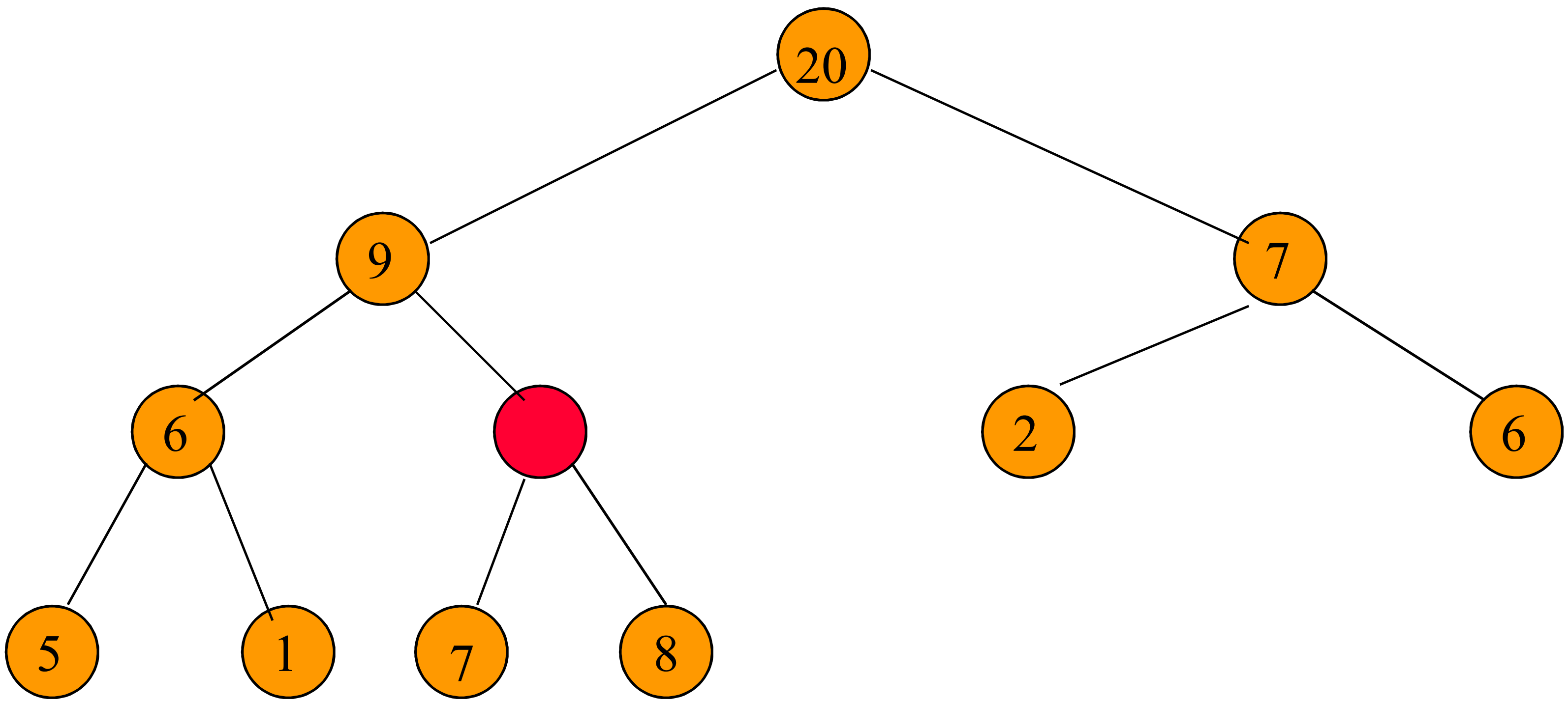
Complete binary tree with 11 nodes.

Inserting an Element into a Max Heap



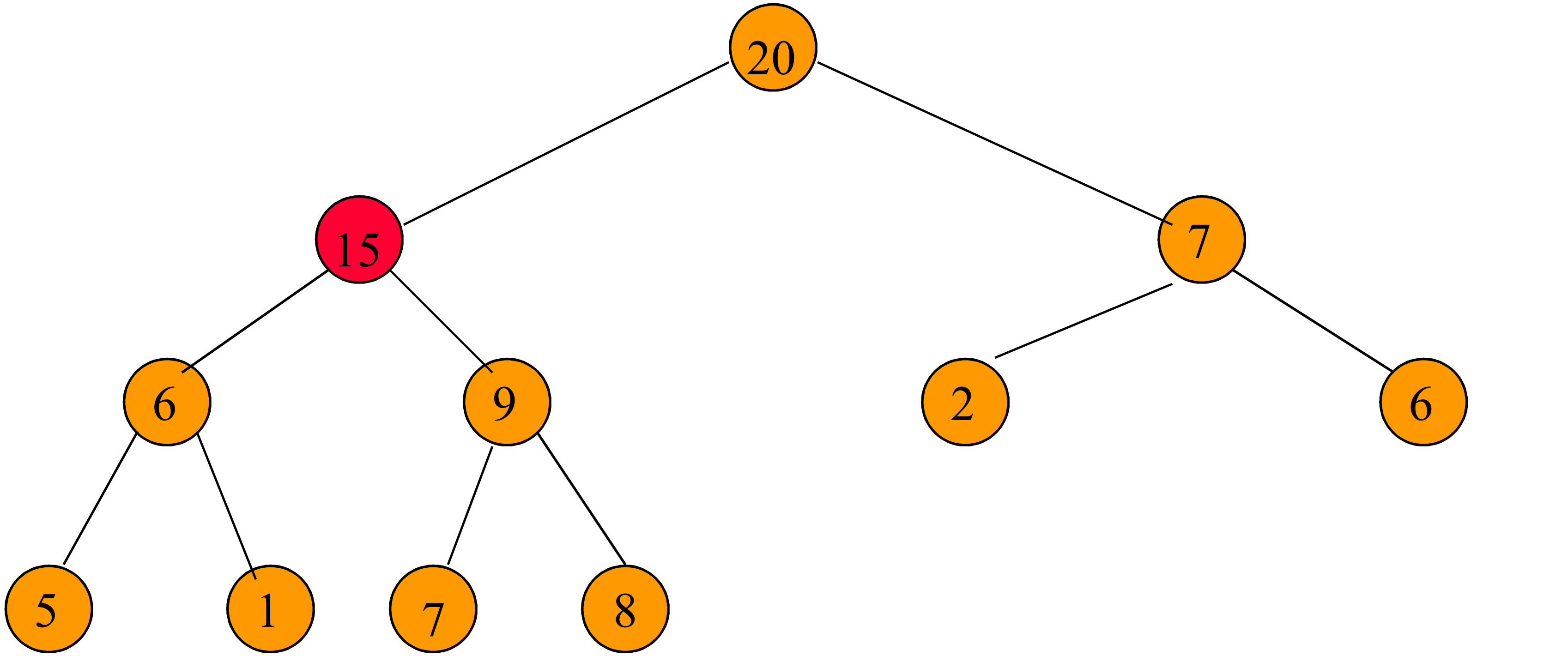
New element is 15.

Inserting an Element into a Max Heap



New element is 15.

Inserting an Element into a Max Heap



New element is 15.

Inserting an Element into a Max Heap

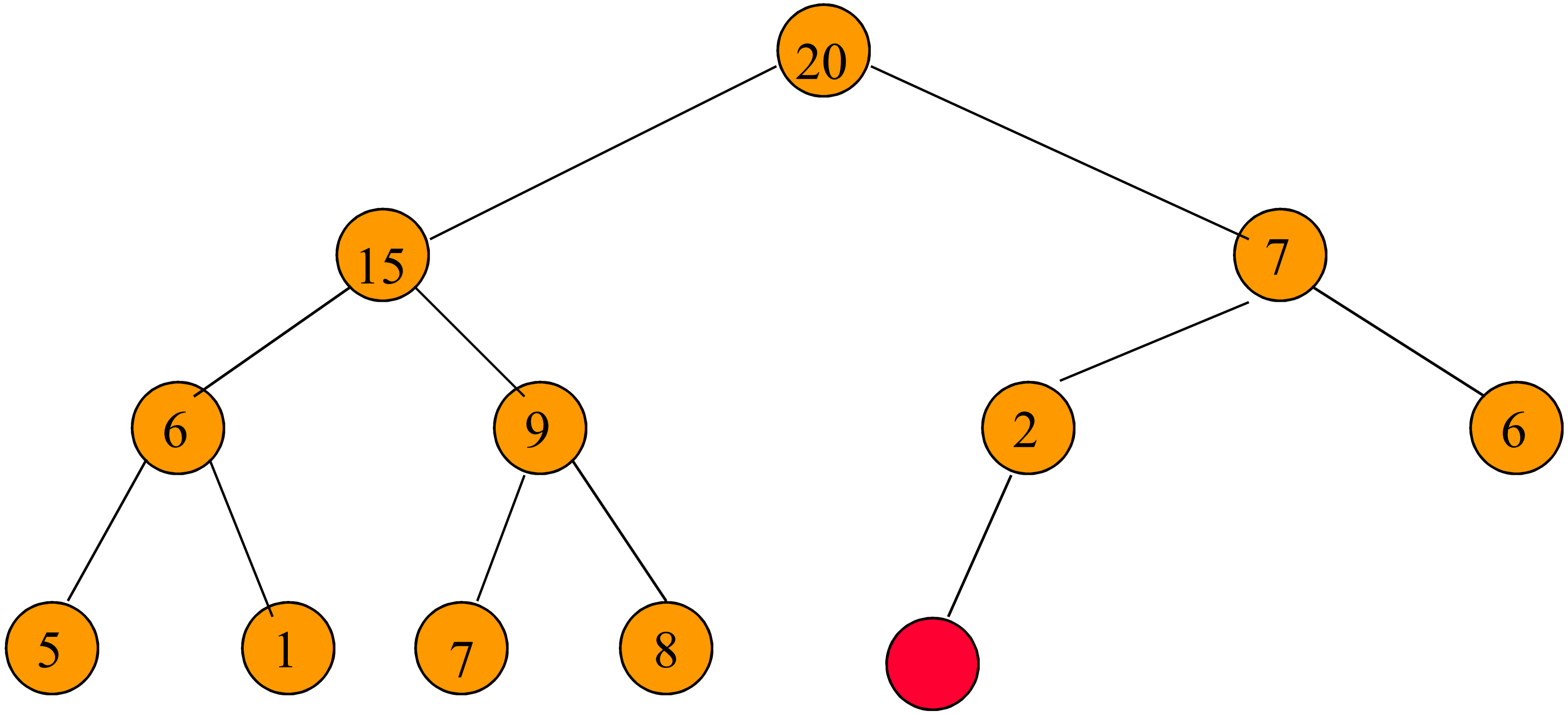
```
template <class T>
void MaxHeap<T>::Push(const T& e)
{// Insert e into the max heap.
  if (heapSize == capacity) {// double the capacity
    ChangeSize1D(heap, capacity, 2 * capacity);
    capacity *=2;
  }
  int currentNode = ++heapSize;
  while (currentNode != 1 && heap[currentNode / 2] < e)
  {// bubble up
    heap[currentNode] = heap[currentNode/2];  // move parent down
    currentNode /= 2; // move to parent
  }
  heap[currentNode] = e;
}
```

```cpp
#pragma warning(disable:4996)
#include <algorithm>
using namespace std;
```

```cpp
template <class T>
void ChangeSize1D(T*& a, const int oldSize, const int newSize)
{
    if (newSize < 0) throw "New length must be >= 0";

    T* temp = new T[newSize];                   // new array
    int number = min(oldSize, newSize);  // number to copy
    copy(a, a + number, temp);
    delete [] a;                                // deallocate old memory
    a = temp;
}
```

http://www.cplusplus.com/reference/algorithm/copy/

Complexity of Insert

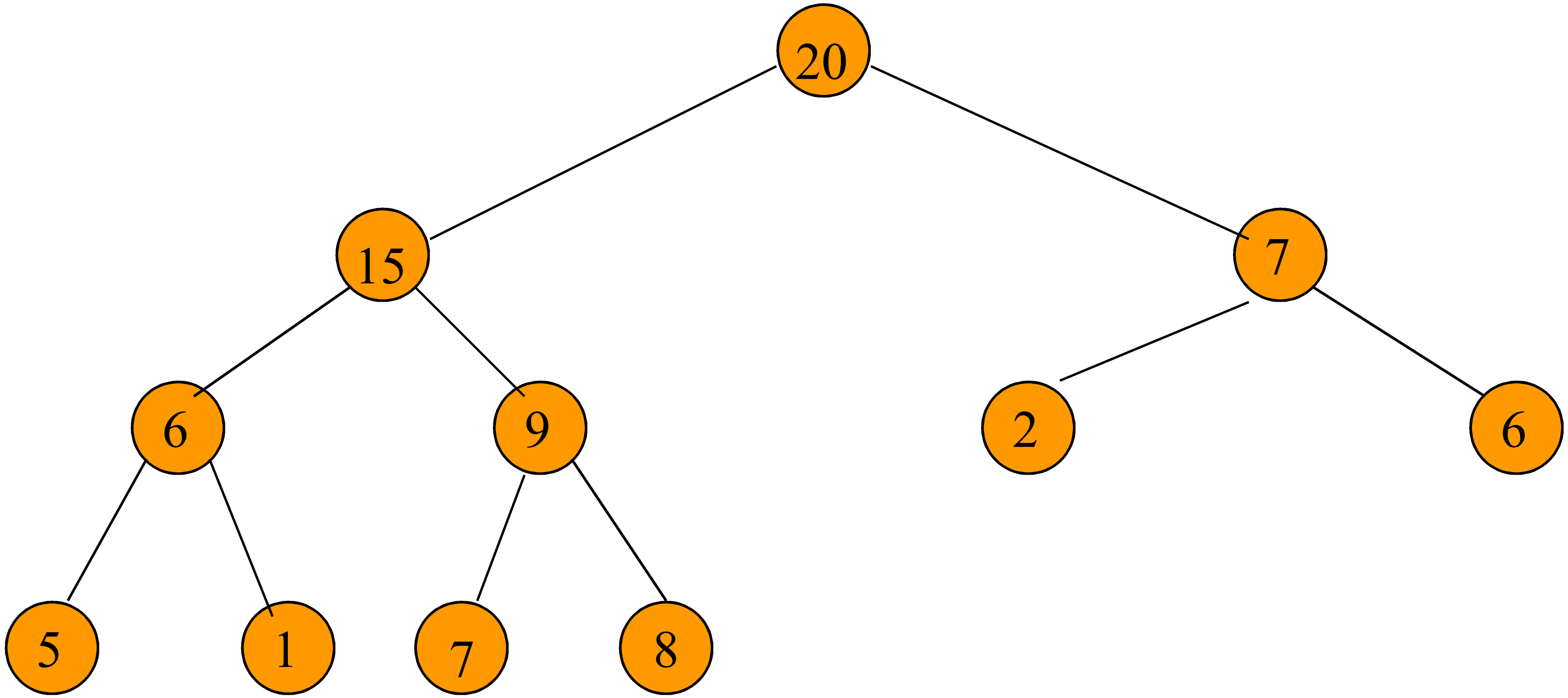


Complexity is O(log n), where n is heap size.

Deletion of the Root from a Max Heap

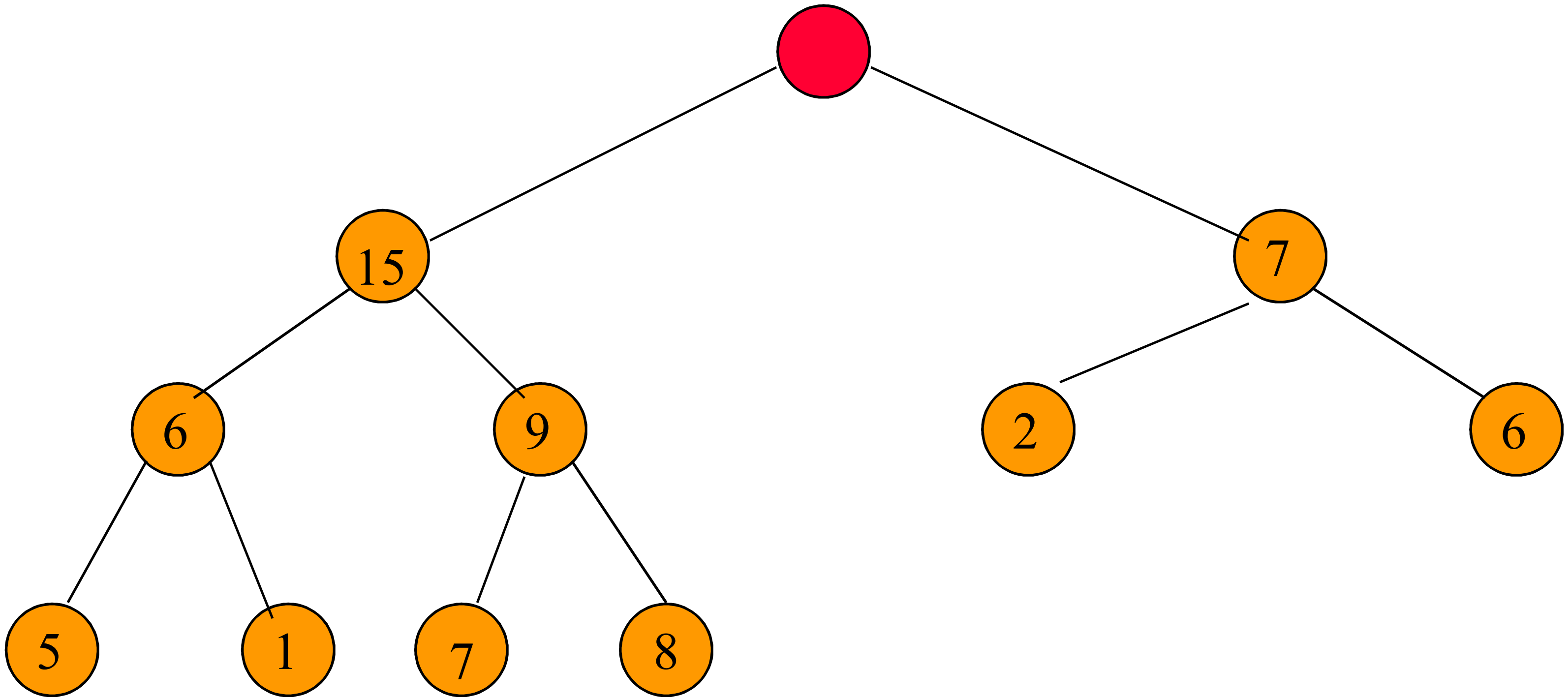
1. Replace the root of the heap with the last element on the last level
2. Compare the new root with its children; if they are in the correct order, stop
3. If not, swap the element with one of its children and return to the previous step
   – Swap with its smaller child in a min-heap and its larger child in a max-heap.

Removing the Max Element



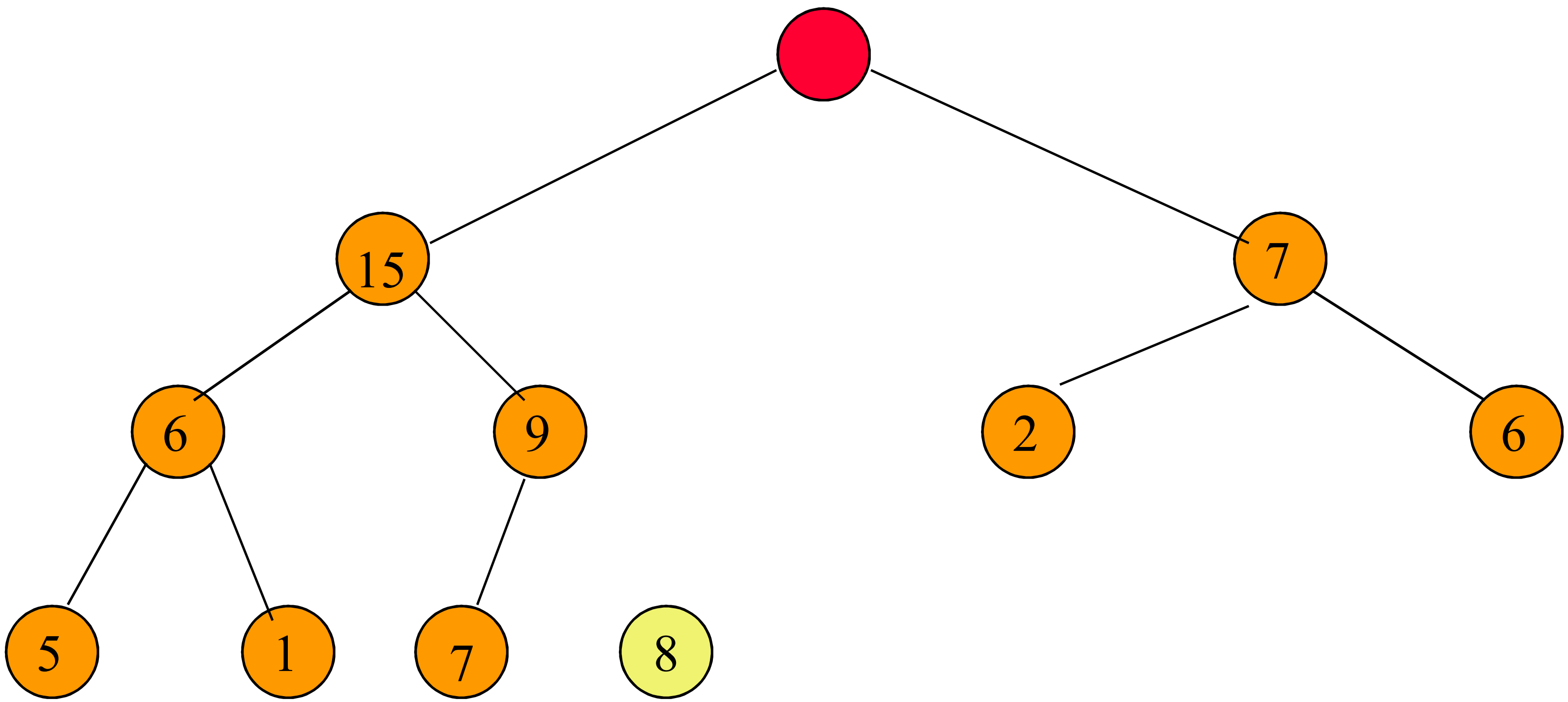
Max element is in the root.

Removing the Max Element



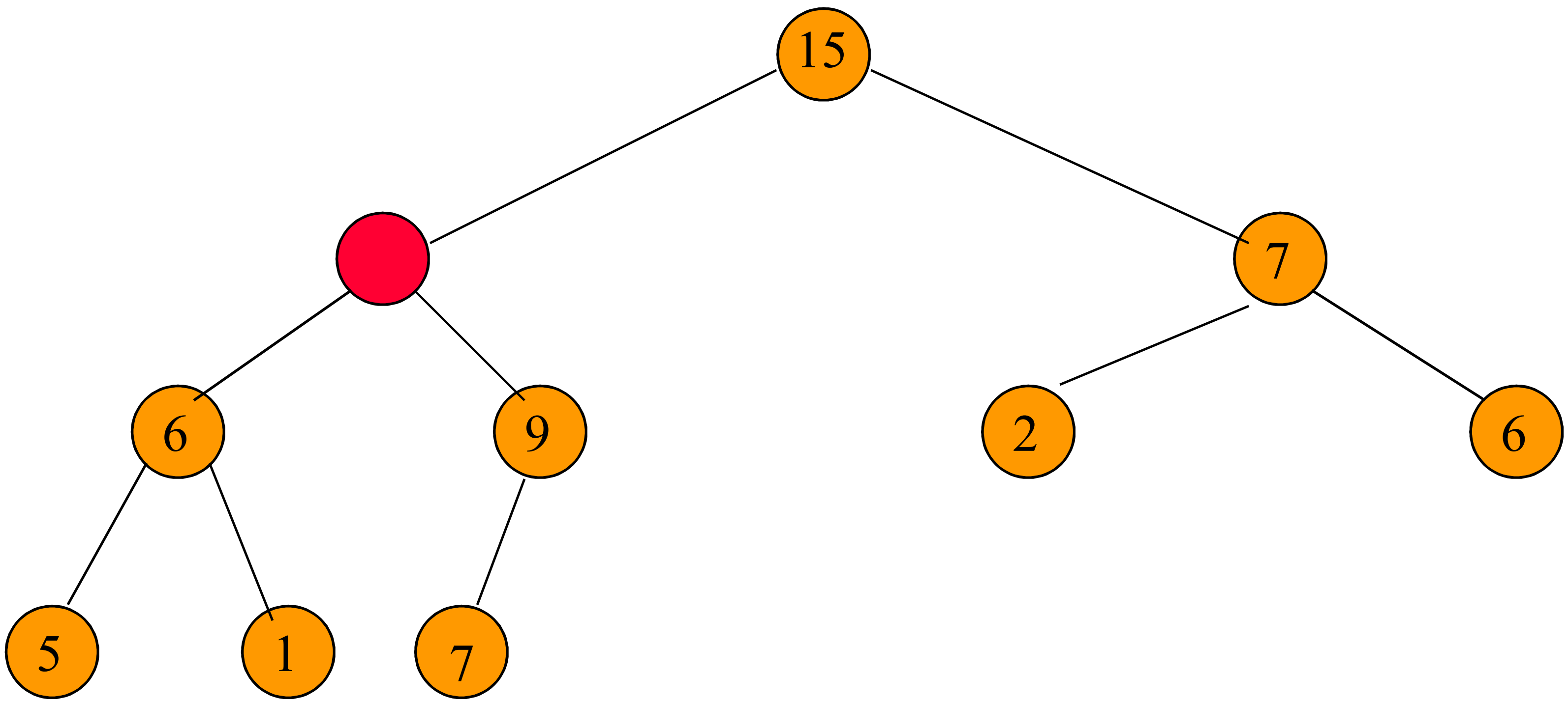
After max element is removed.

Removing the Max Element



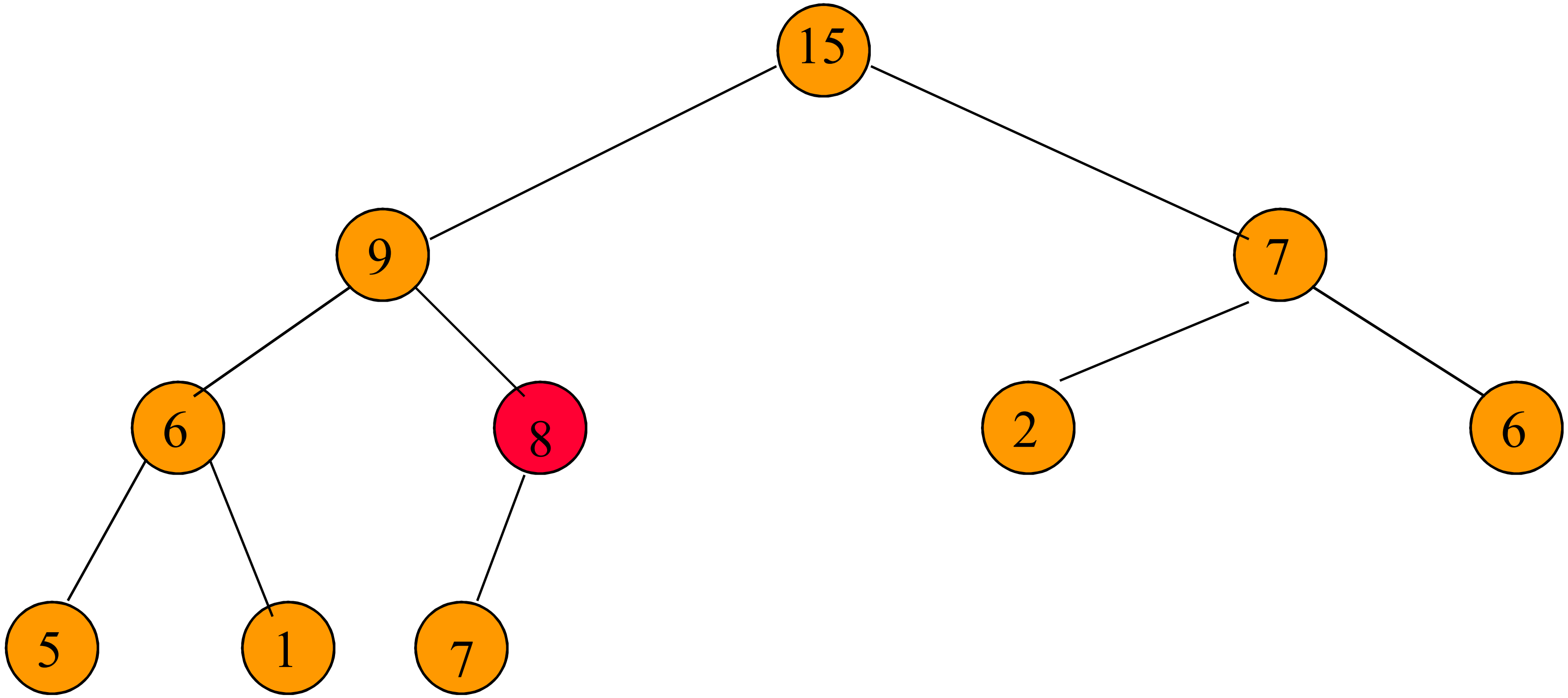
Reinsert 8 into the heap.

Removing the Max Element



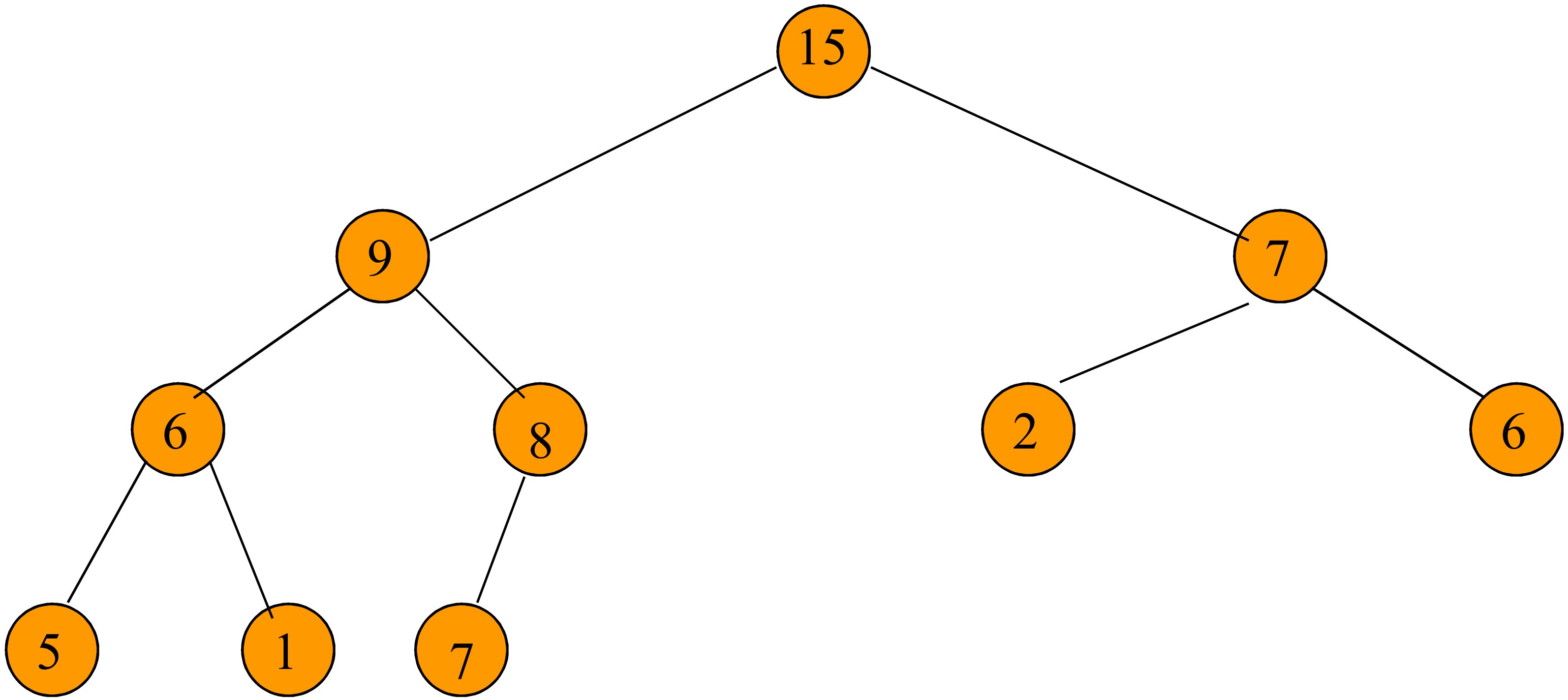
Reinsert 8 into the heap.

Removing the Max Element



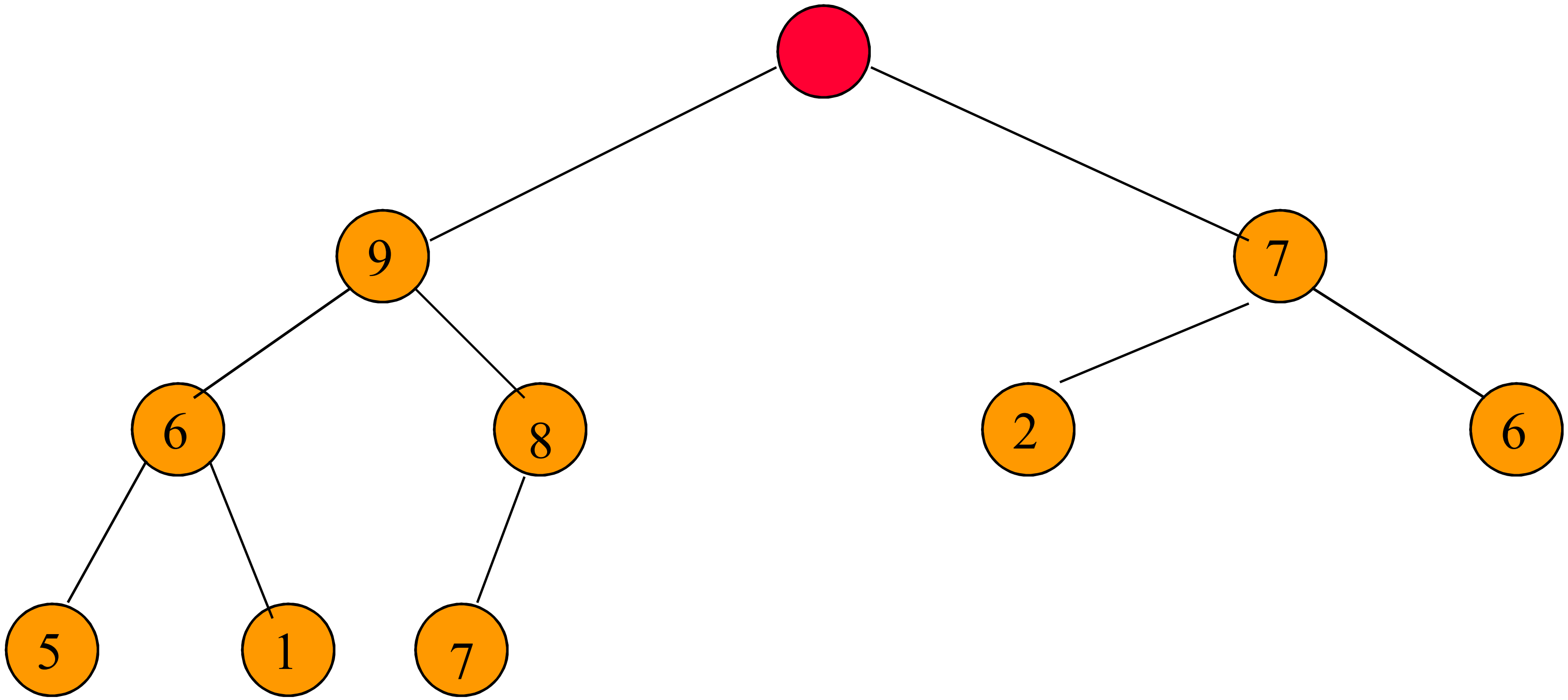
Reinsert 8 into the heap.

Removing the Max Element



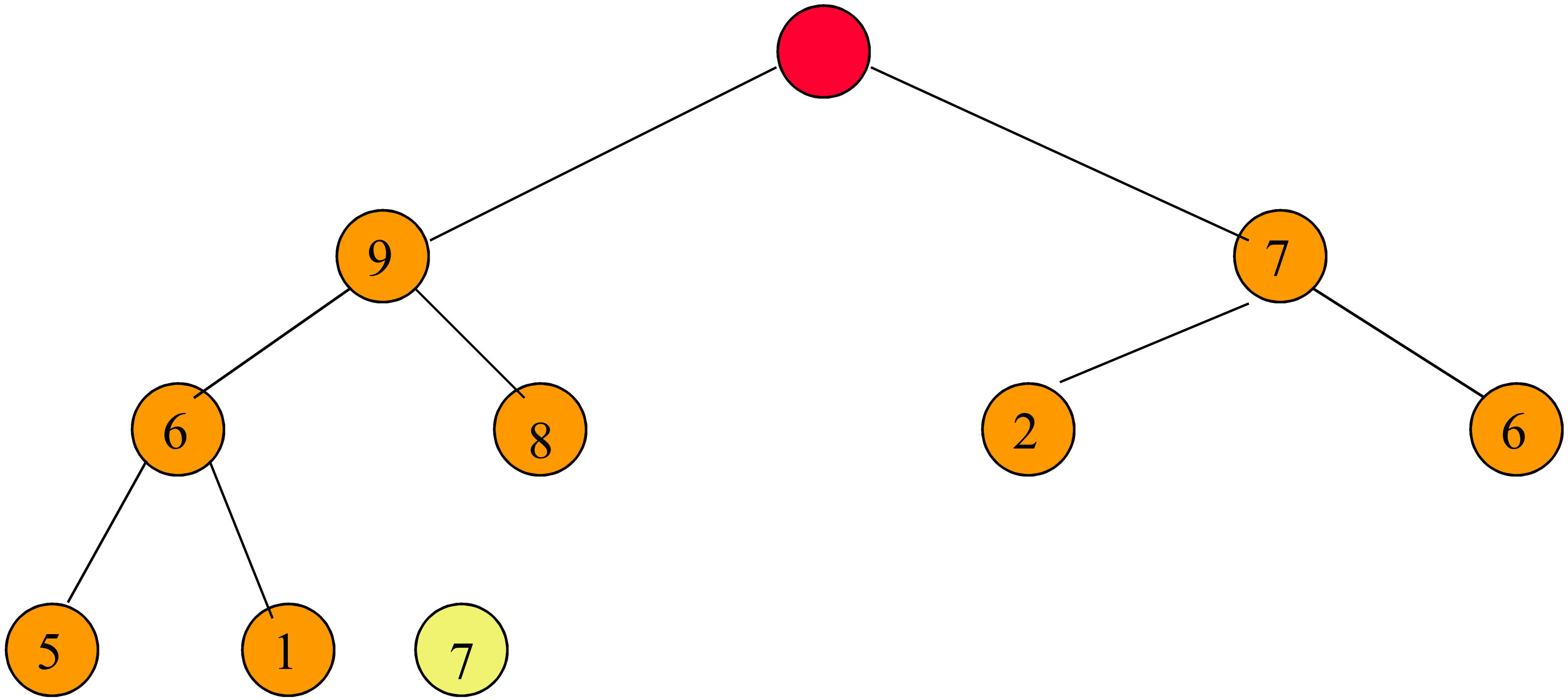
Max element is 15.

Removing the Max Element



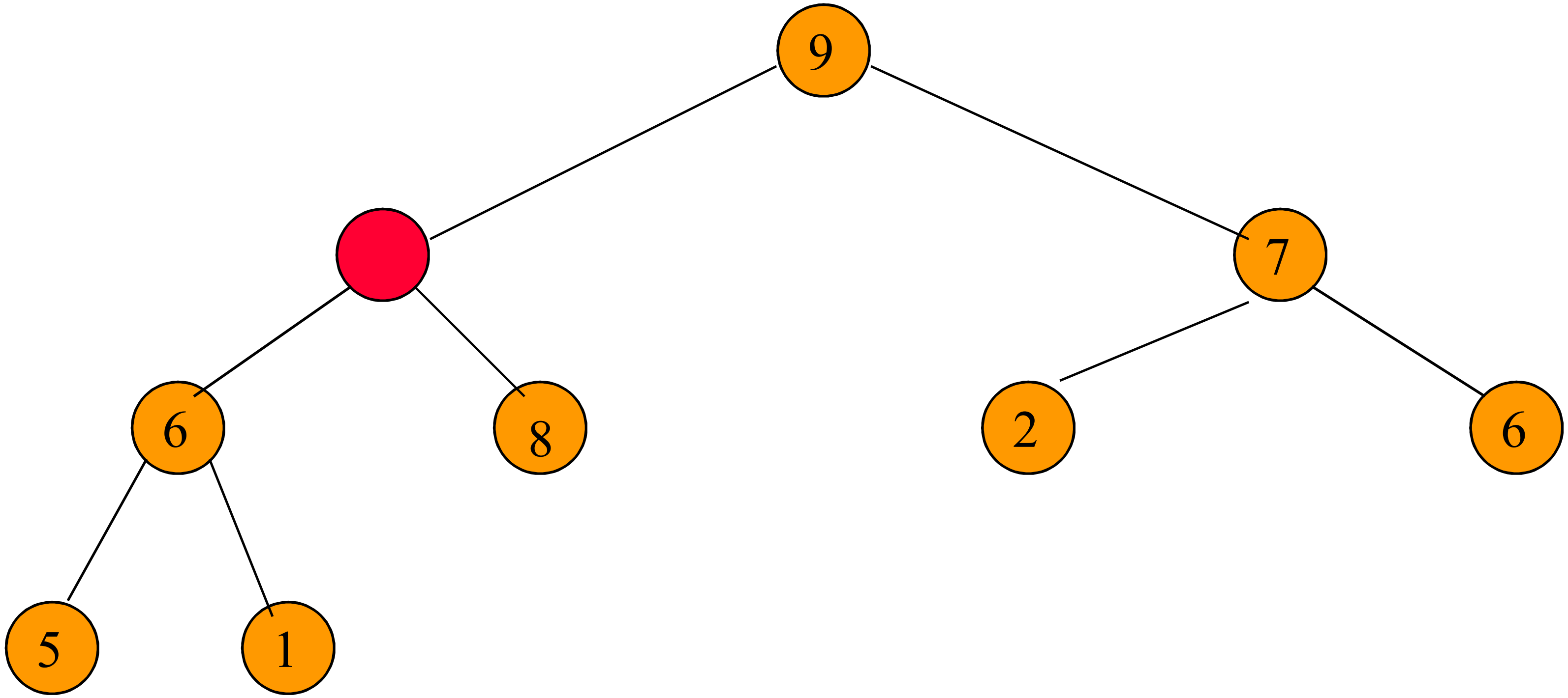
After max element is removed.

Removing the Max Element



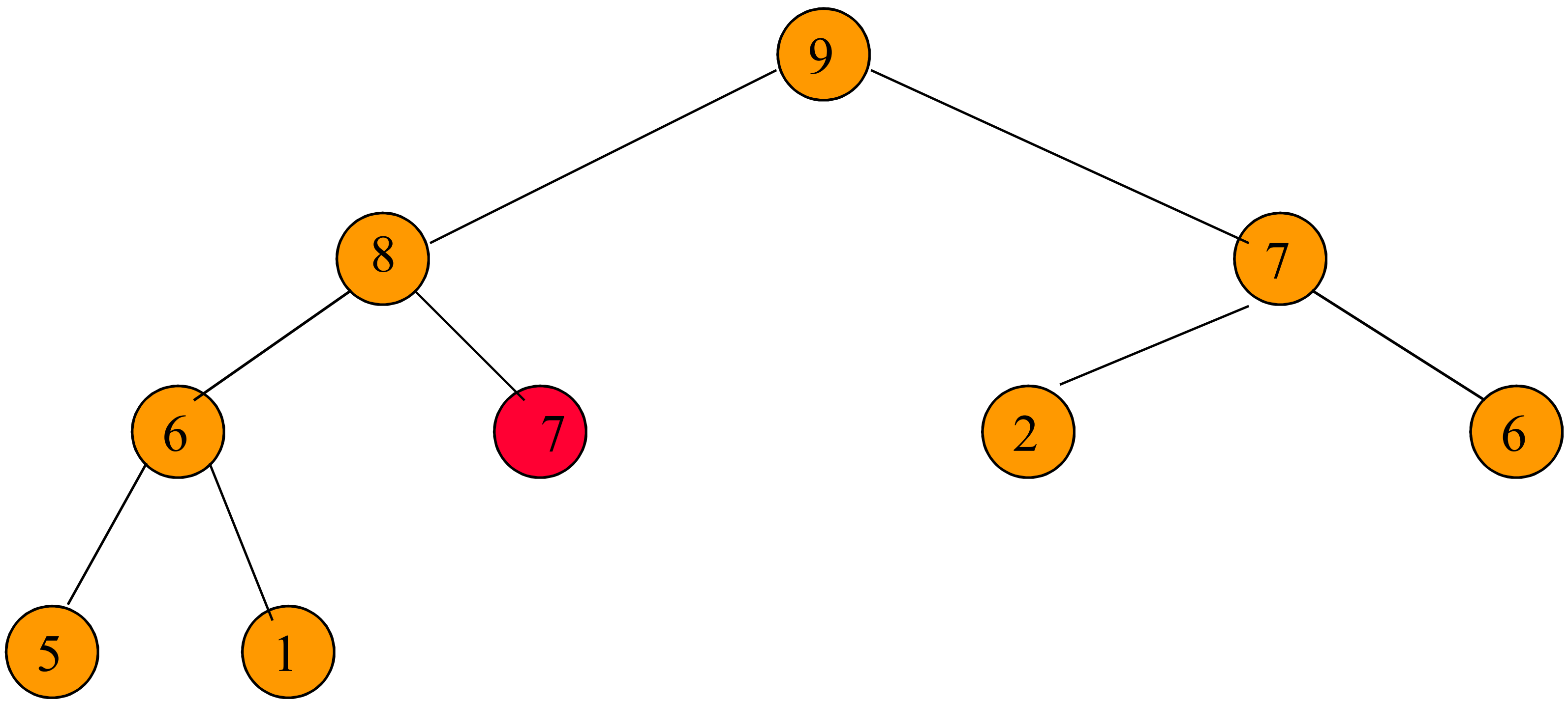
Reinsert 7.

Removing the Max Element



Reinsert 7.

Removing the Max Element

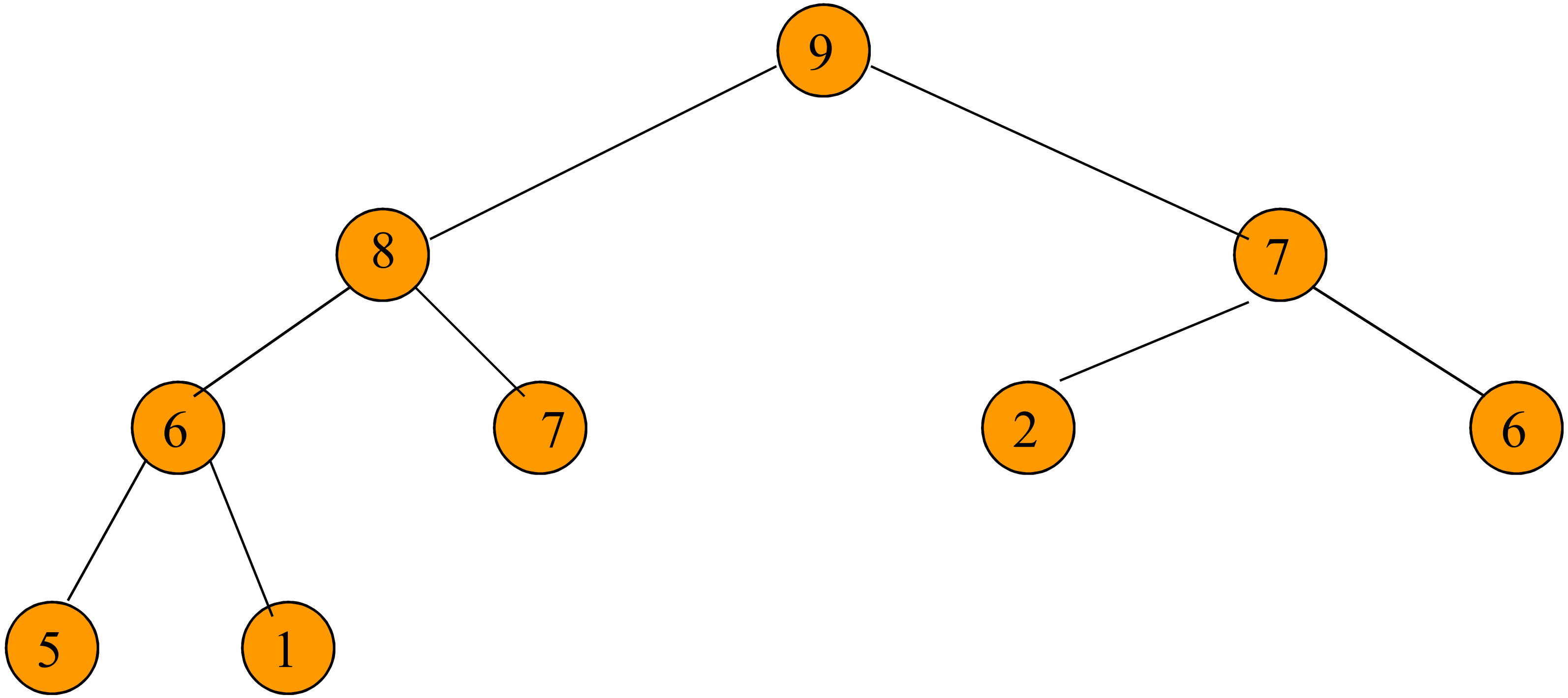


Reinsert 7.

Deletion from a max heap

```cpp
template <class T>
void MaxHeap<T>::Pop()
{// Delete max element.
  if (IsEmpty()) throw "Heap is empty. Cannot delete.";
  heap[1].~T();    // delete max element

  // remove last element from heap
  T lastE = heap[heapSize--];

  // trickle down
  int currentNode = 1;   // root
  int child = 2;         // a child of currentNode
  while (child <= heapSize)
  {
    // set child to larger child of currentNode
    if (child < heapSize && heap[child] < heap[child+1]) child++;

    // can we put lastE in currentNode?
    if (lastE >= heap[child]) break;   // yes

    // no
    heap[currentNode] = heap[child];   // move child up
    currentNode = child; child *= 2;   // move down a level
  }
  heap[currentNode] = lastE;
}
```

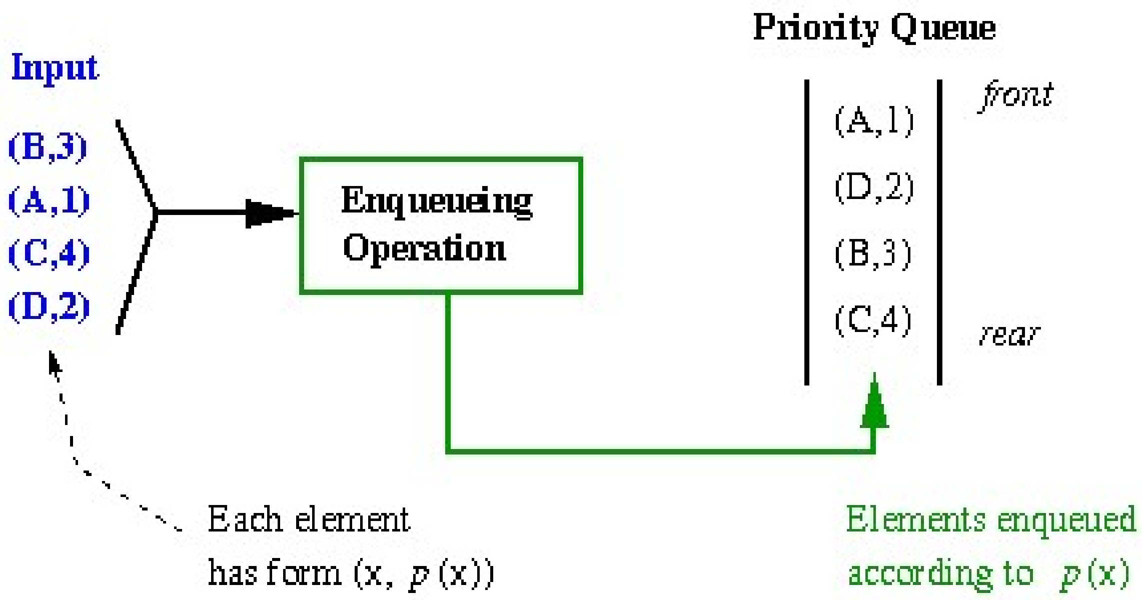
Complexity of Deletion



Complexity is O(log n).

Priority Queues

- The element to be deleted is the one with highest (or lowest) priority
- At any time, an element with arbitrary priority can be inserted into the queue



Priority Queues (cont.)

Two kinds of priority queues:

- Min priority queue
- Max priority queue

Max Priority Queue

- Collection of elements
- Each element has a priority
- Supports following operations:
  - empty
  - size
  - insert an element into the priority queue (push)
  - get element with max priority (top)
  - remove element with max priority (pop)

Complexity of Operations

Use a heap

empty, size, and top => O(1) time

insert (push) and remove (pop) =>
O(log n) time where n is the size of the priority queue

Applications of Priority Queues

- Sorting
  - use element key as priority
  - insert elements to be sorted into a priority queue
  - remove/pop elements in priority order
    - if a min priority queue is used, elements are extracted in ascending order of priority (or key)
    - if a max priority queue is used, elements are extracted in descending order of priority (or key)
- Scheduler of an OS

Heap Sort

- Uses a max (or min) priority queue that is implemented as a heap

- Complexity of sorting n elements.
  - n insert operations => O(n log n) time.
  - n remove max operations => O(n log n) time.
  - total time is O(n log n).
  - compare with $O(n^2)$ for insertion or bubble sort

Homework #3

Due: 10/1일(월) 자정까지

1. Implement and test
   - Programs 5.15, 5.16, 5.17
2. 예외처리(try, throw, catch)에 대해 공부하고 예를 들어 설명할 것