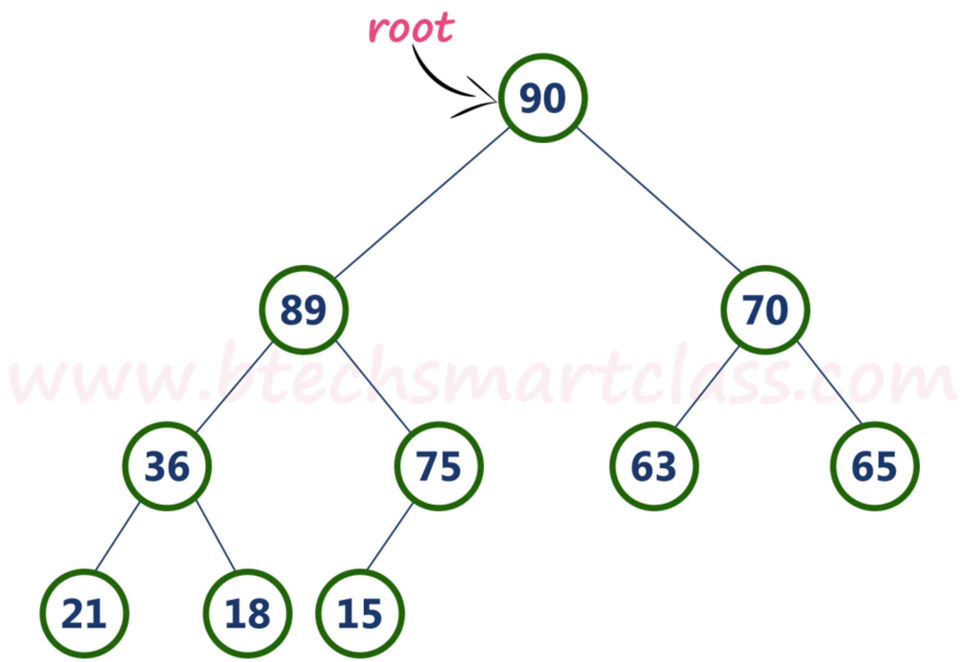
Heaps

Prof. Ki-Hoon Lee
School of Computer and Information Engineering
Kwangwoon University

Heap

- A *max tree* is a tree in which the key value in each node is no smaller than the key values in its children (if any)



http://www.btechsmartclass.com/data_structures/max-heap.html

Heap (cont.)

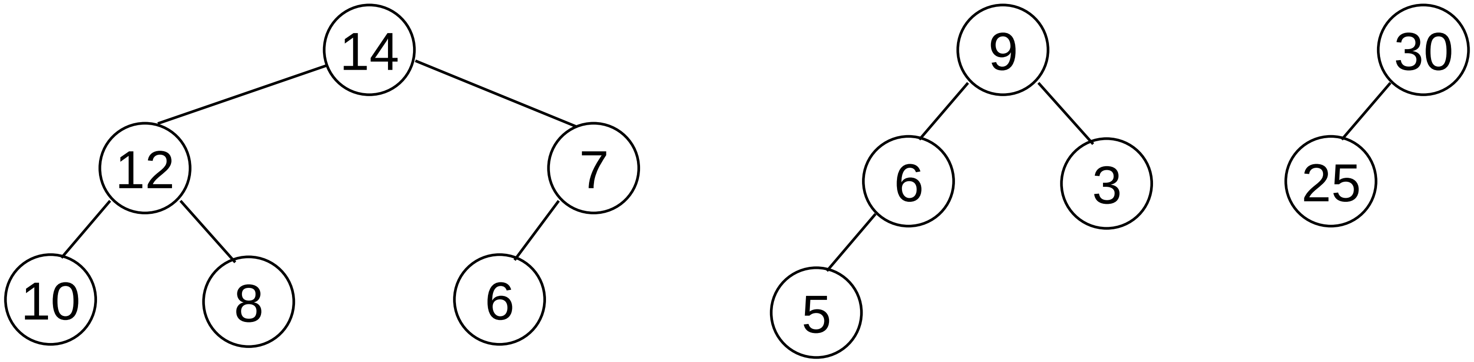
- A *min tree* is a tree in which the key value in each node is no larger than the key values in its children (if any)

- The key in the root of a max (min) tree is the largest (smallest) key in the tree

Heap (cont.)

- A *max heap* is a complete binary tree that is also a max tree

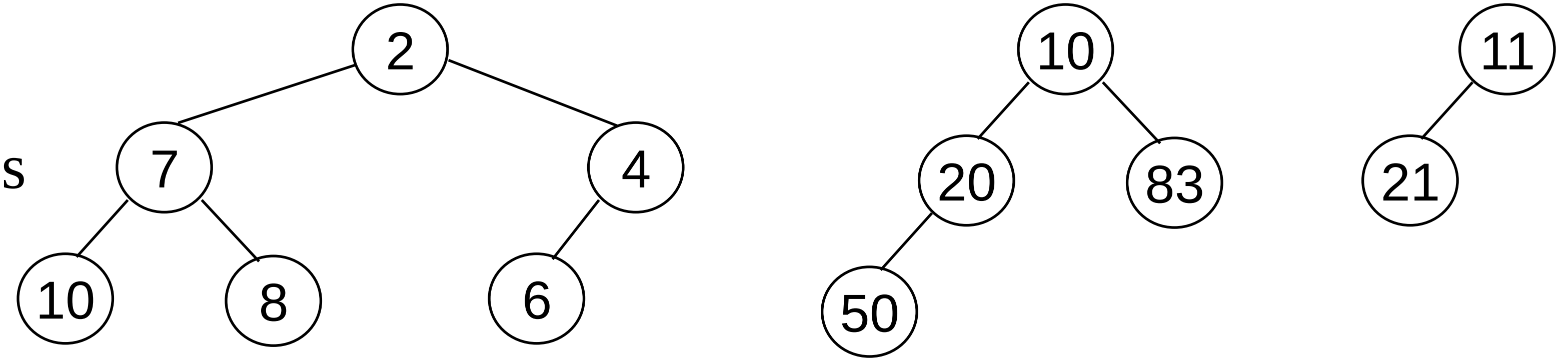
max heaps



Heap (cont.)

- A *min heap* is a complete binary tree that is also a min tree

min heaps



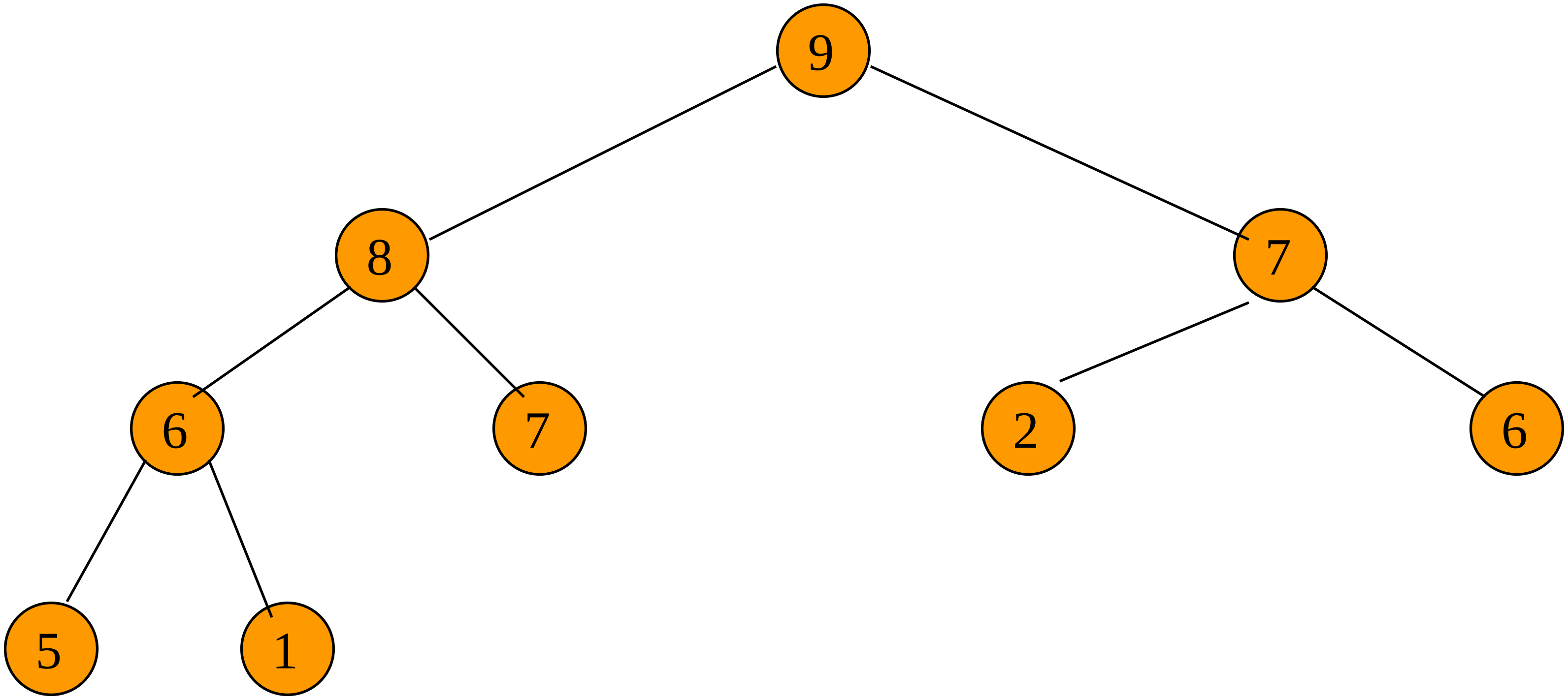
Heap Height

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

Heap (cont.)

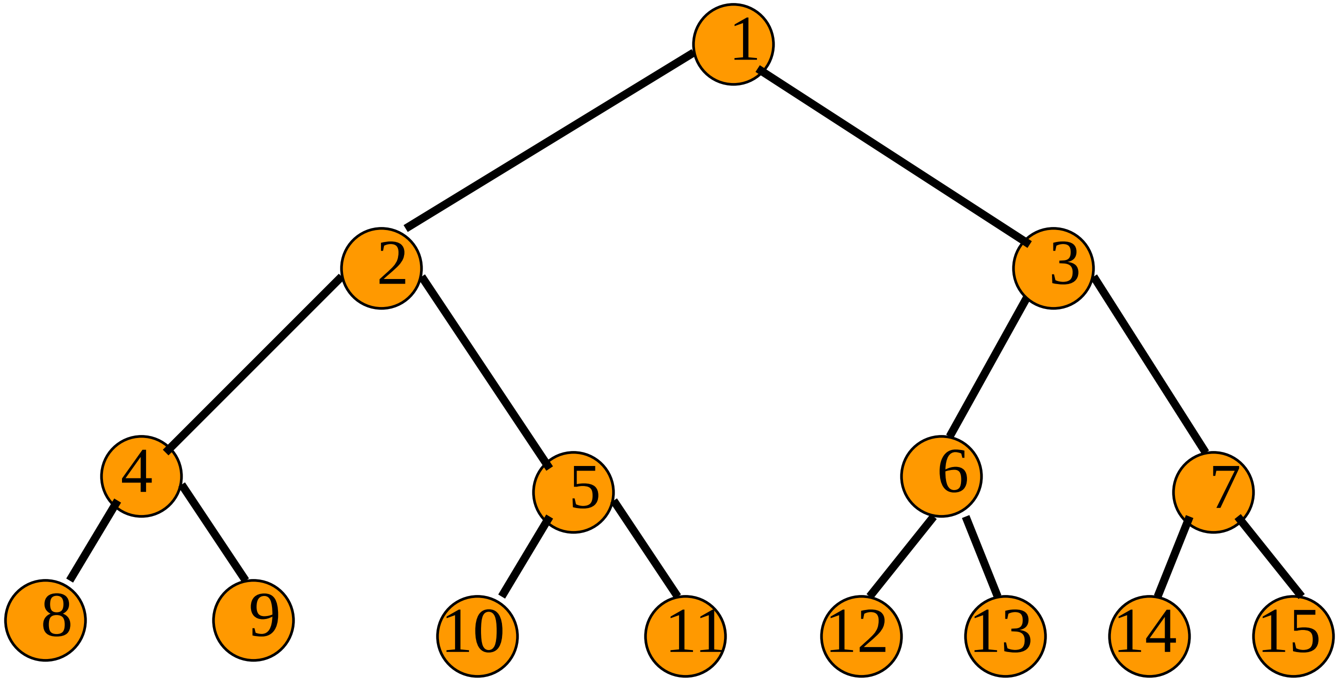
- Basic Operations
  - Creation of an empty heap
  - Insertion
  - Deletion of the root

A Heap Represented as an Array



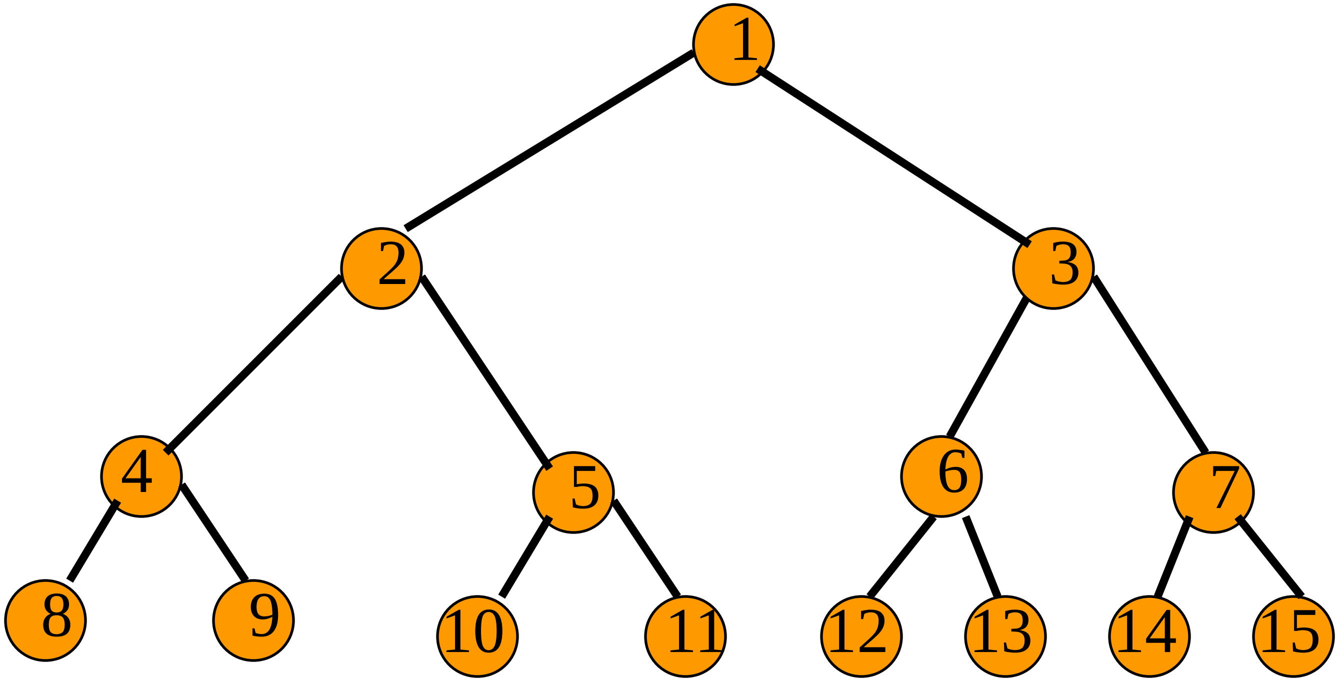
0	1	2	3	4	5	6	7	8	9
	9	8	7	6	7	2	6	5	1

Node Number Properties



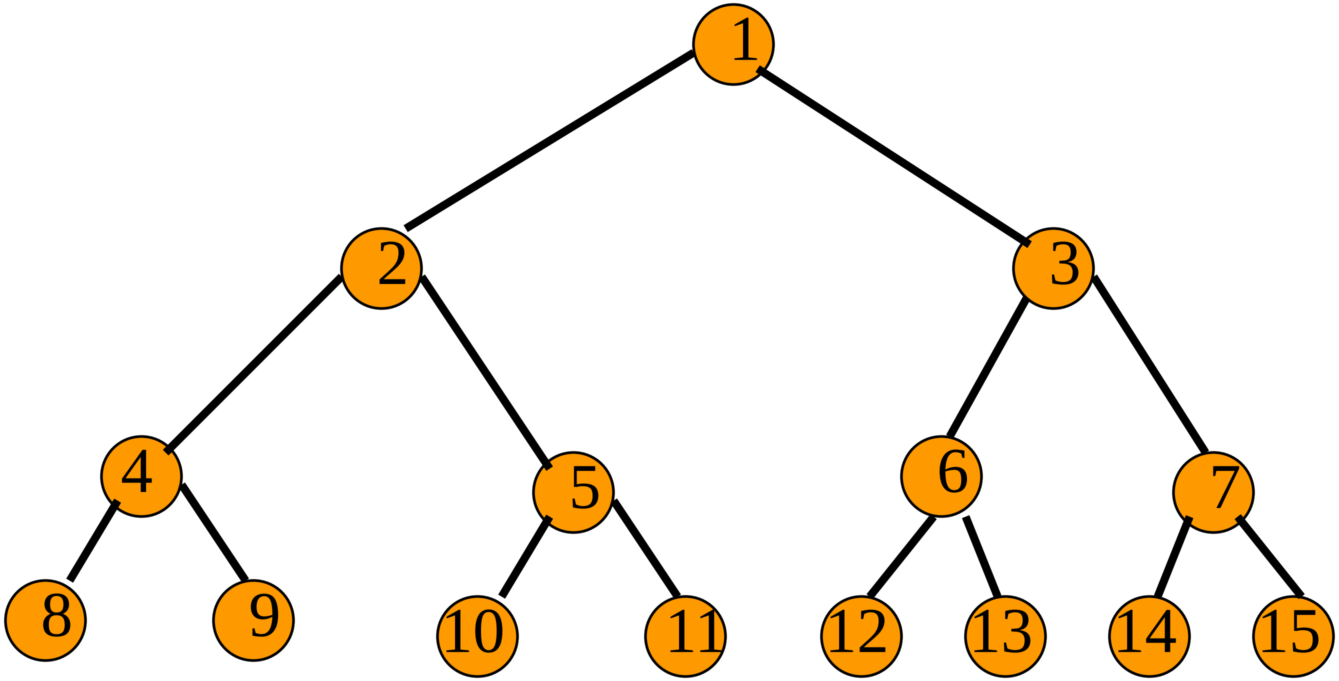
- Parent of node i is node $i / 2$, unless $i = 1$.
- Node 1 is the root and has no parent.

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
}
```

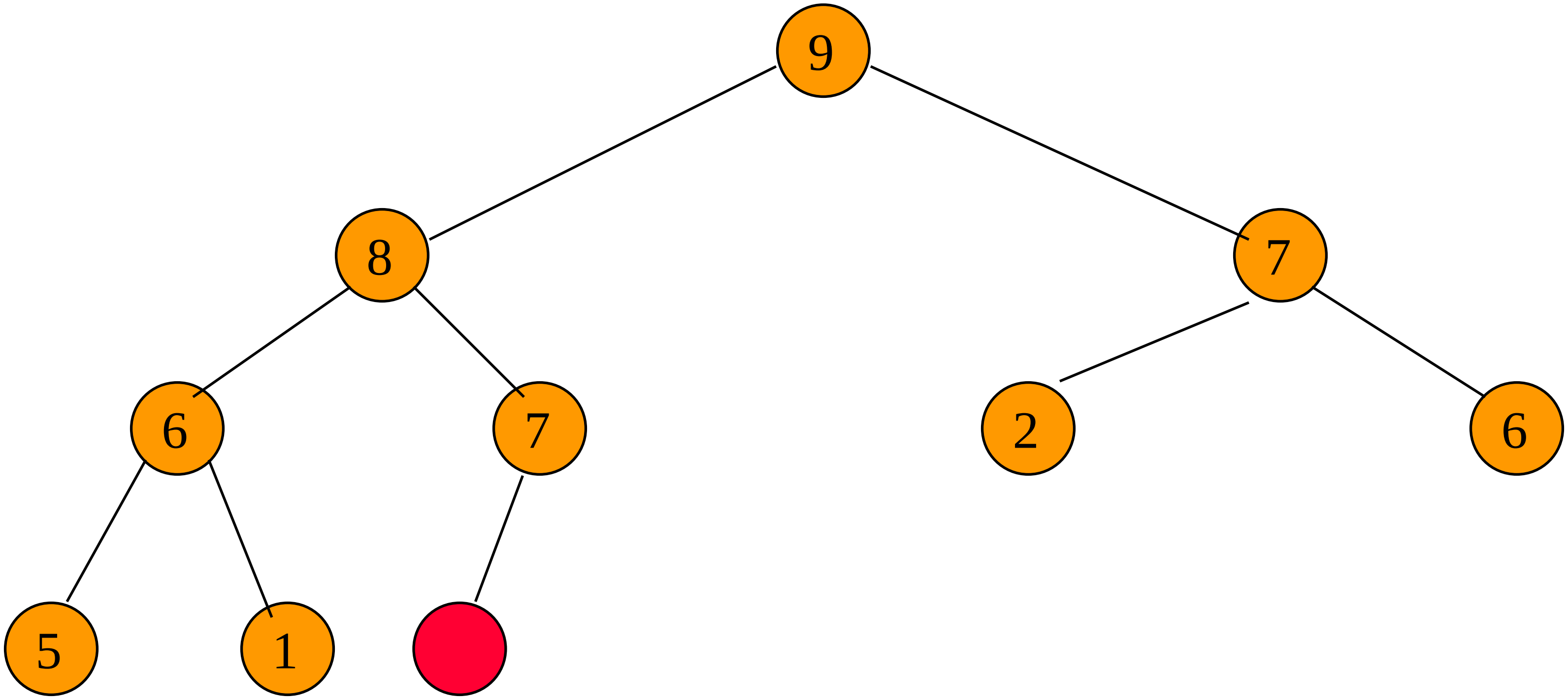
Insertion

- To determine the correct place for the element being inserted, we use a *bubbling up* process

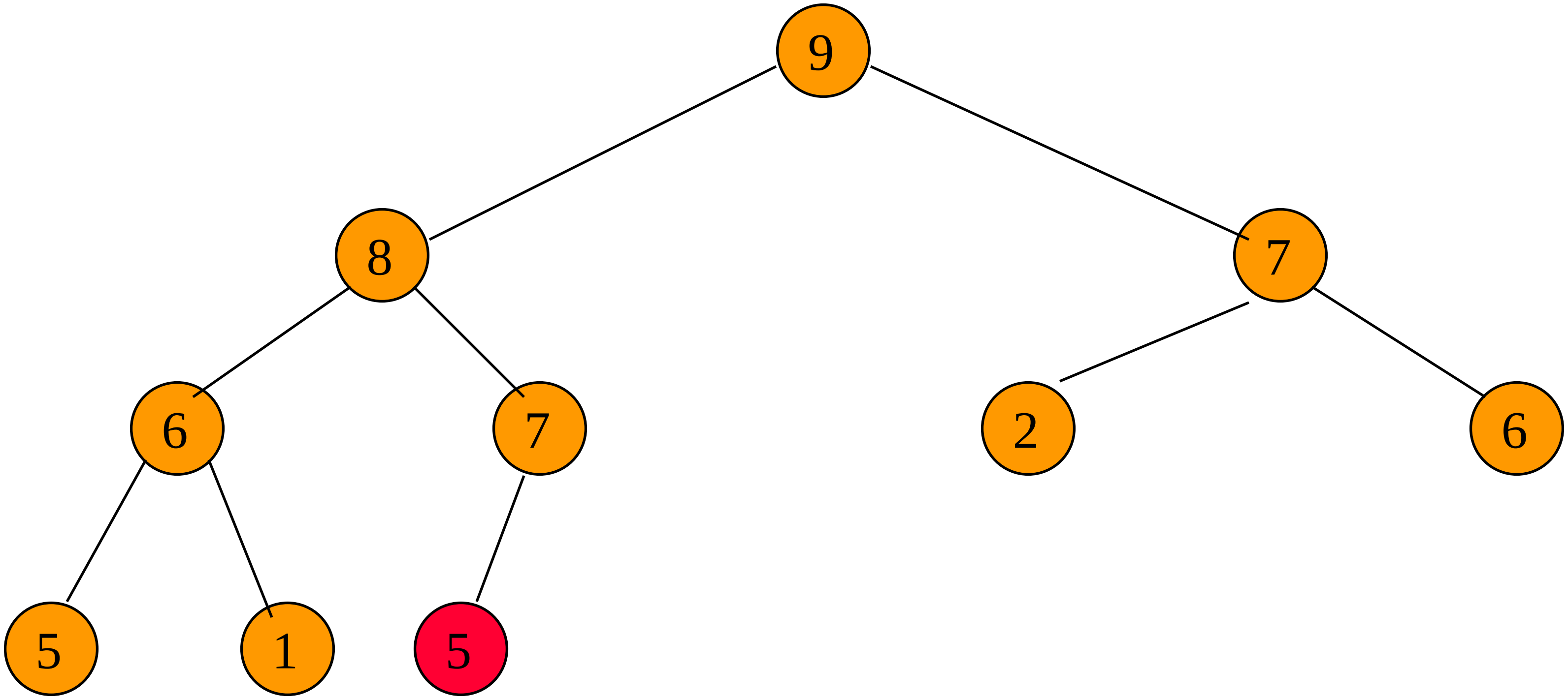
- The bubbling up process begins at a new leaf node and moves up toward the root

- 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

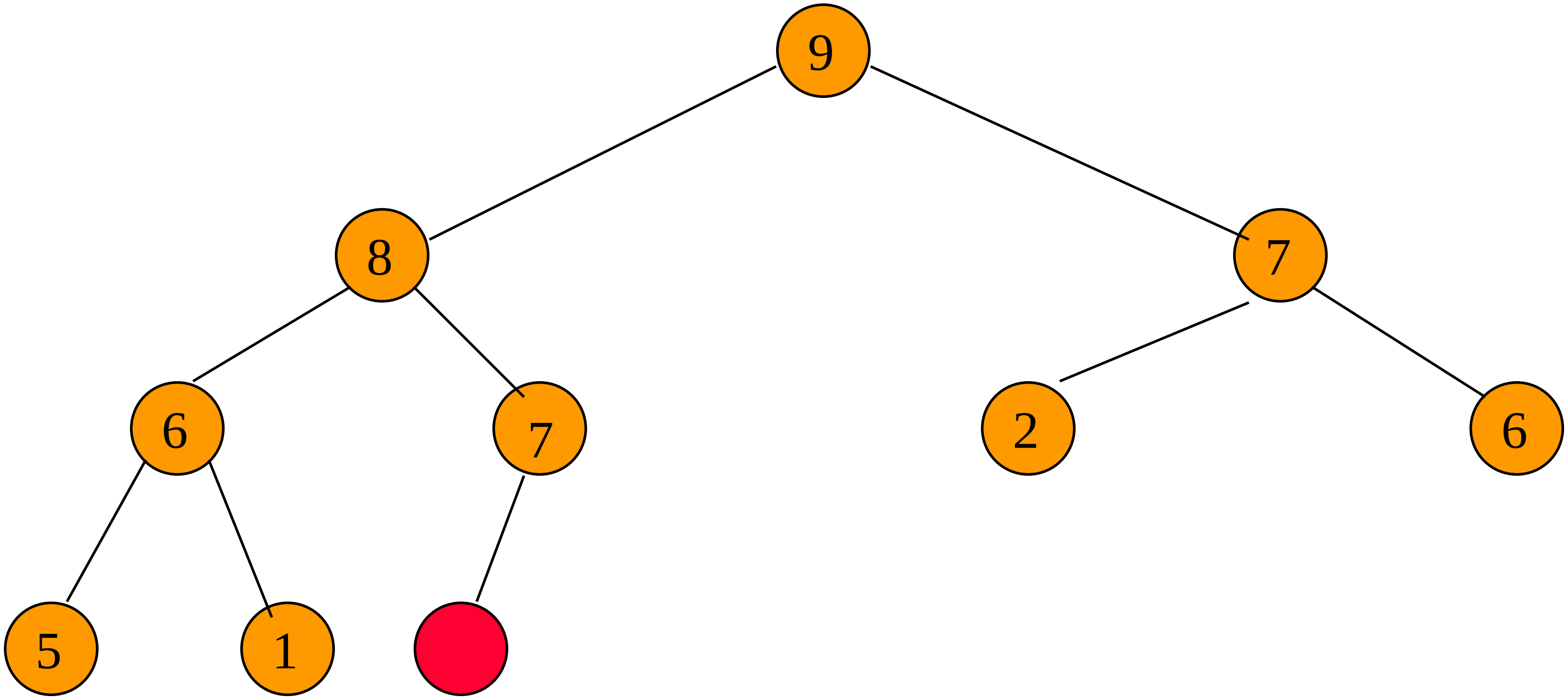


Inserting an Element into a Max Heap



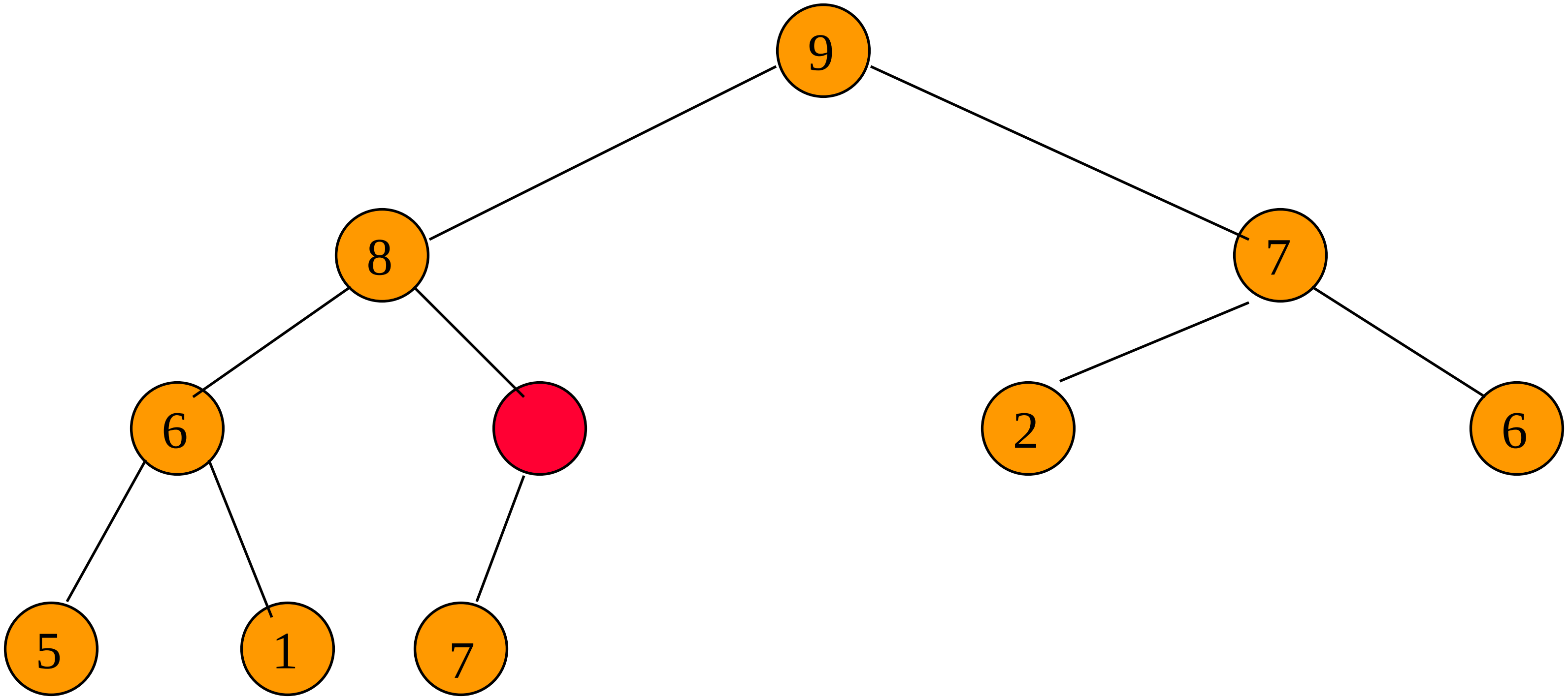
New element is 5.

Inserting an Element into a Max Heap



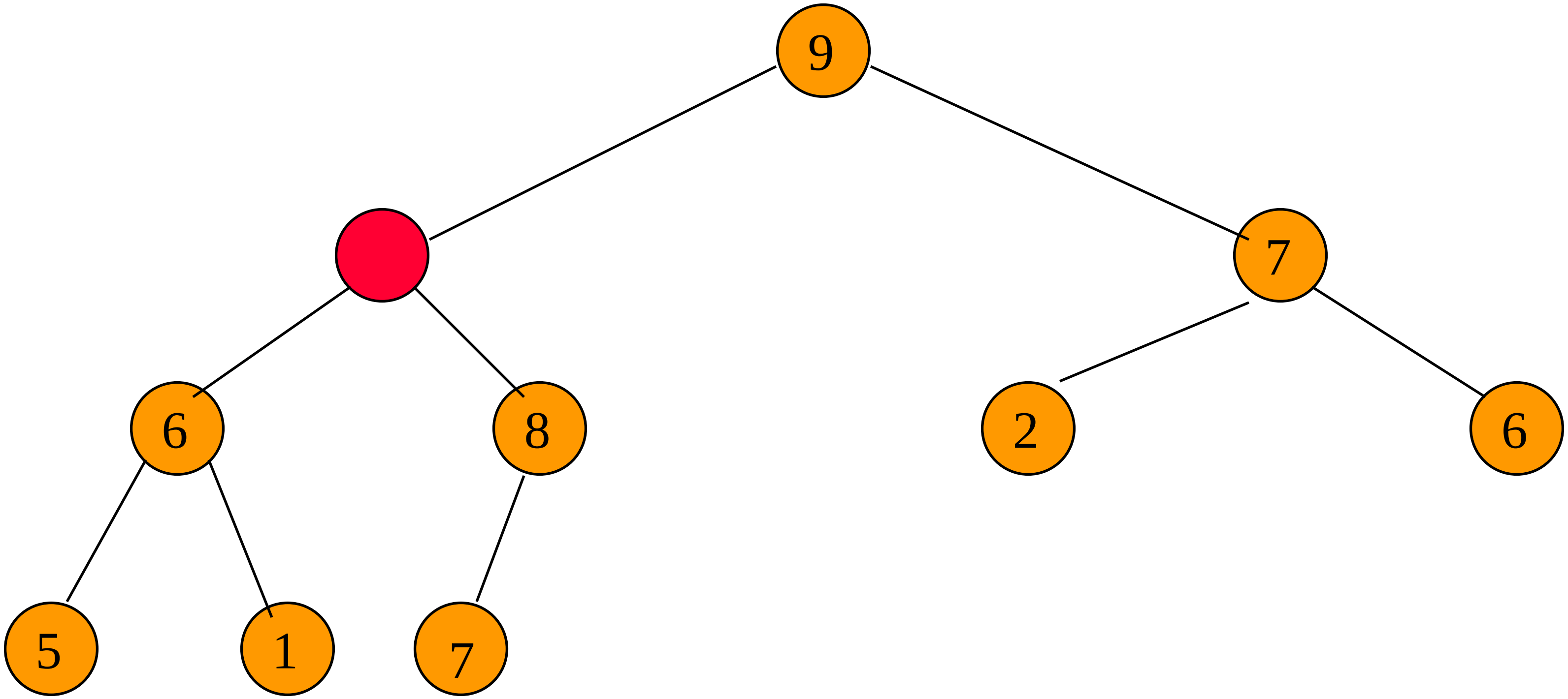
New element is 20.

Inserting an Element into a Max Heap



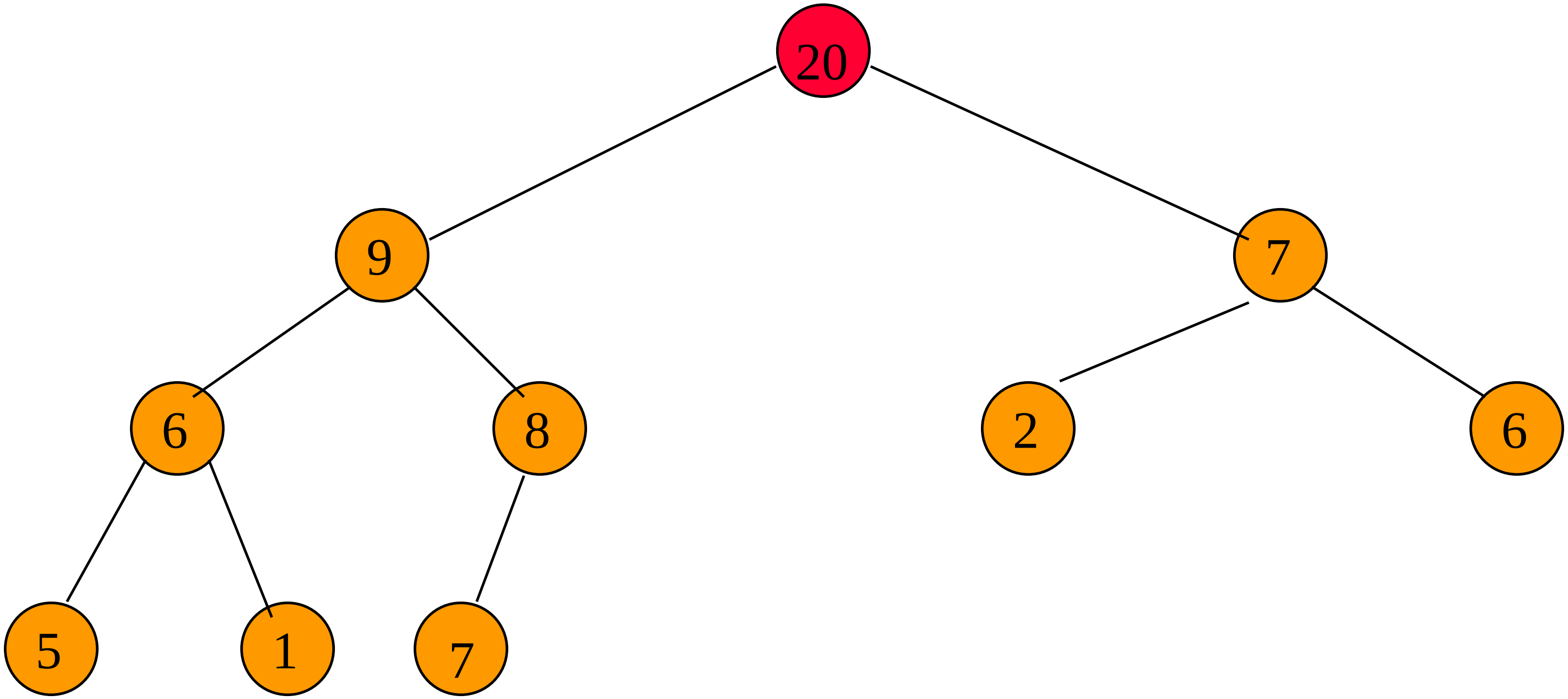
New element is 20.

Inserting an Element into a Max Heap



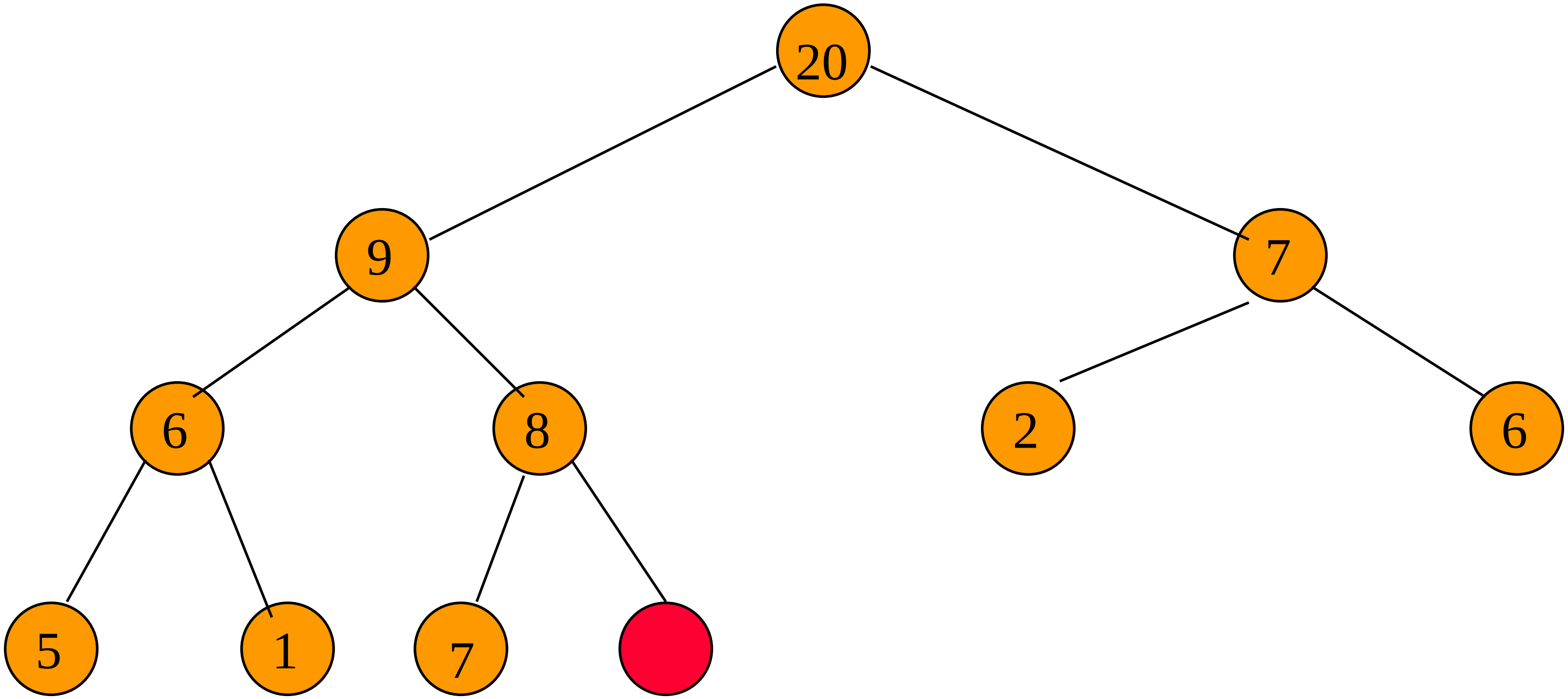
New element is 20.

Inserting an Element into a Max Heap



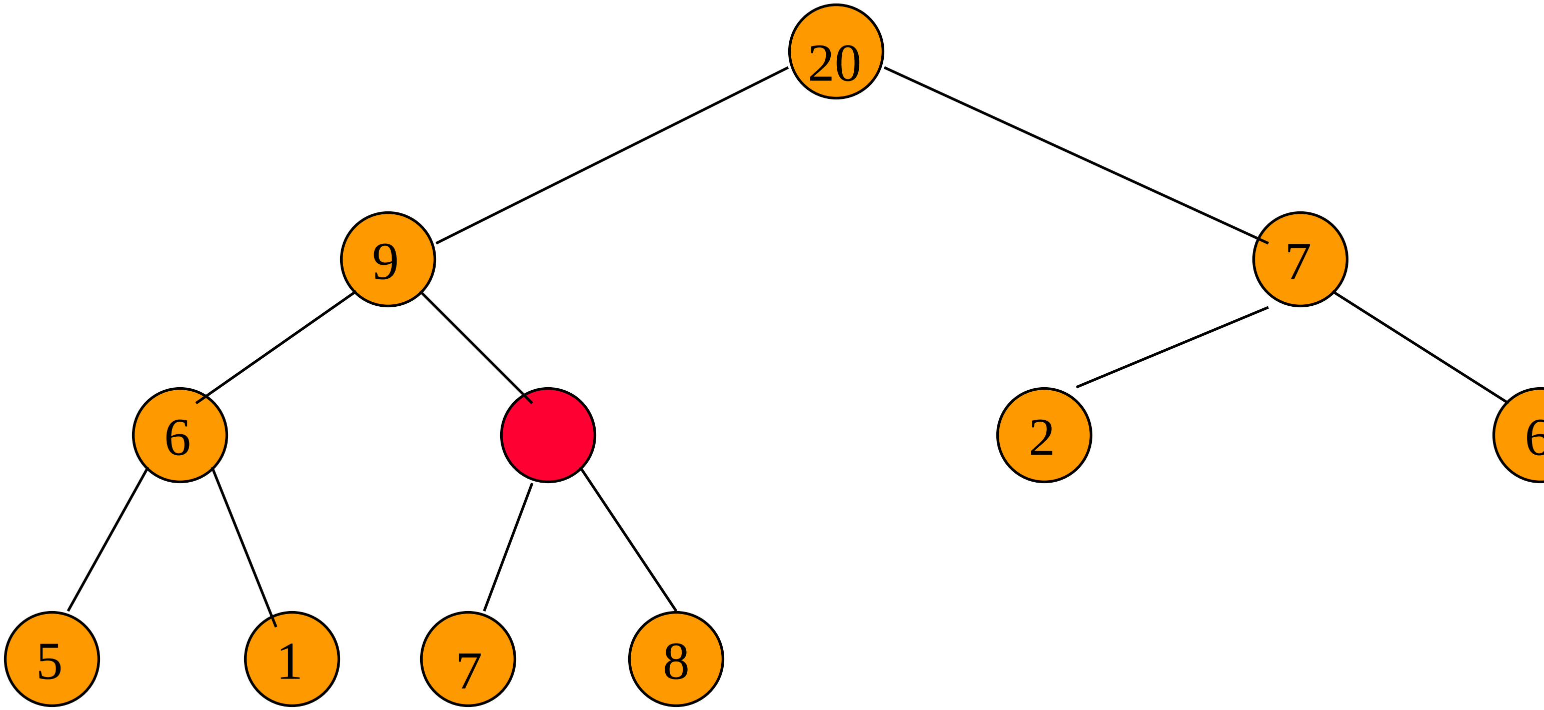
New element is 20.

Inserting an Element into a Max Heap



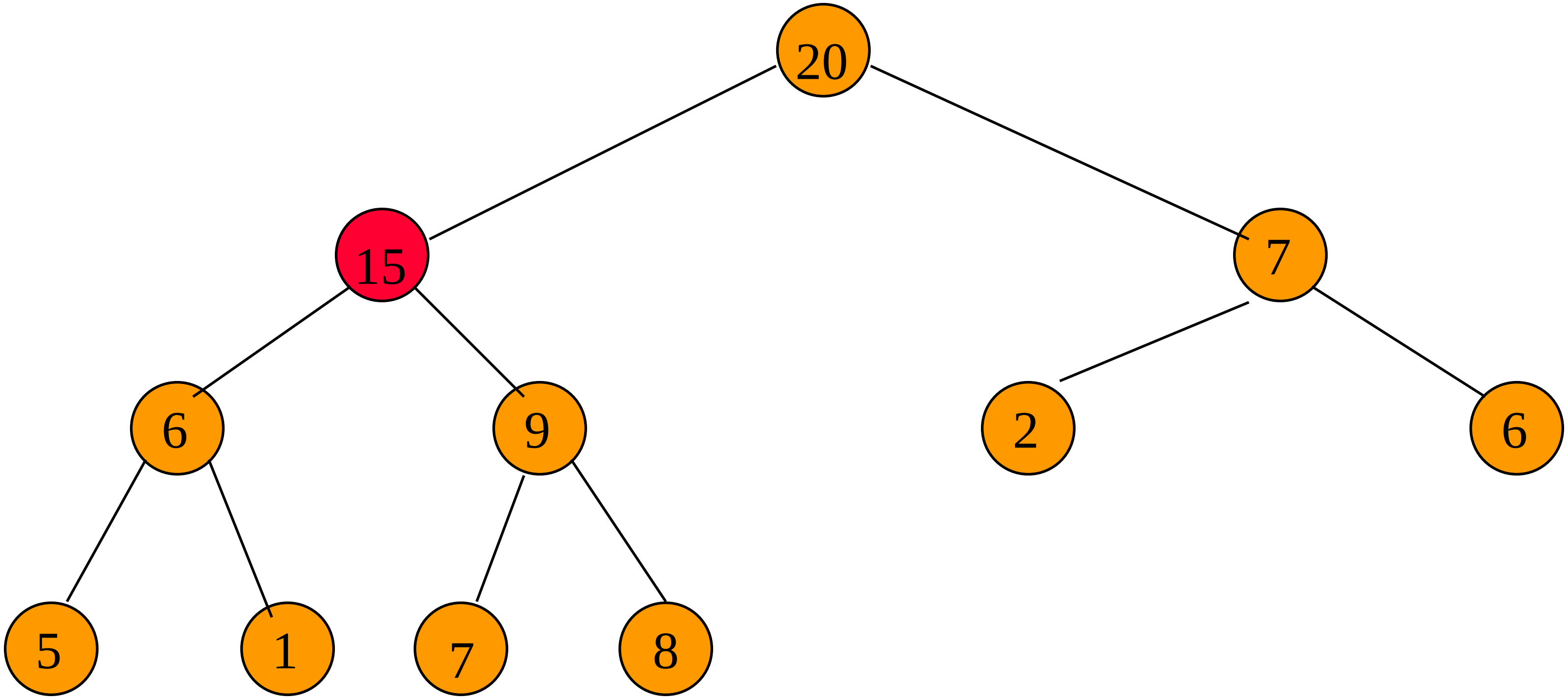
New element is 15.

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

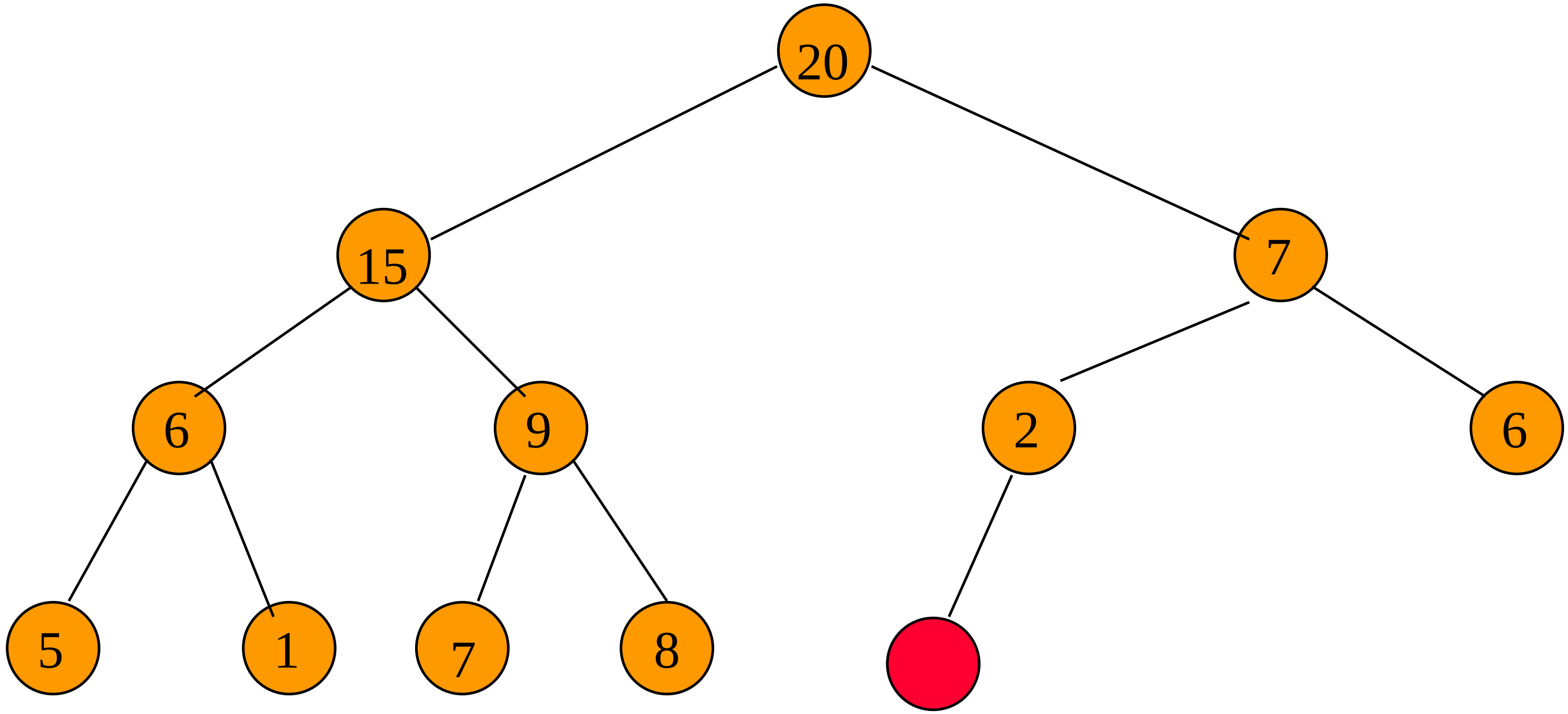
```
template <class T>
void MaxHeap<T>::Push(const T& e)
{// Insert e into the max heap.
  if (heapSize == capacity) {// double the capacity
    ChangeSize1D(heap, capacity, 2 * capacity);
    capacity *=2;
  }
  int currentNode = ++heapSize;
  while (currentNode != 1 && heap[currentNode / 2] < e)
 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

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

```
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

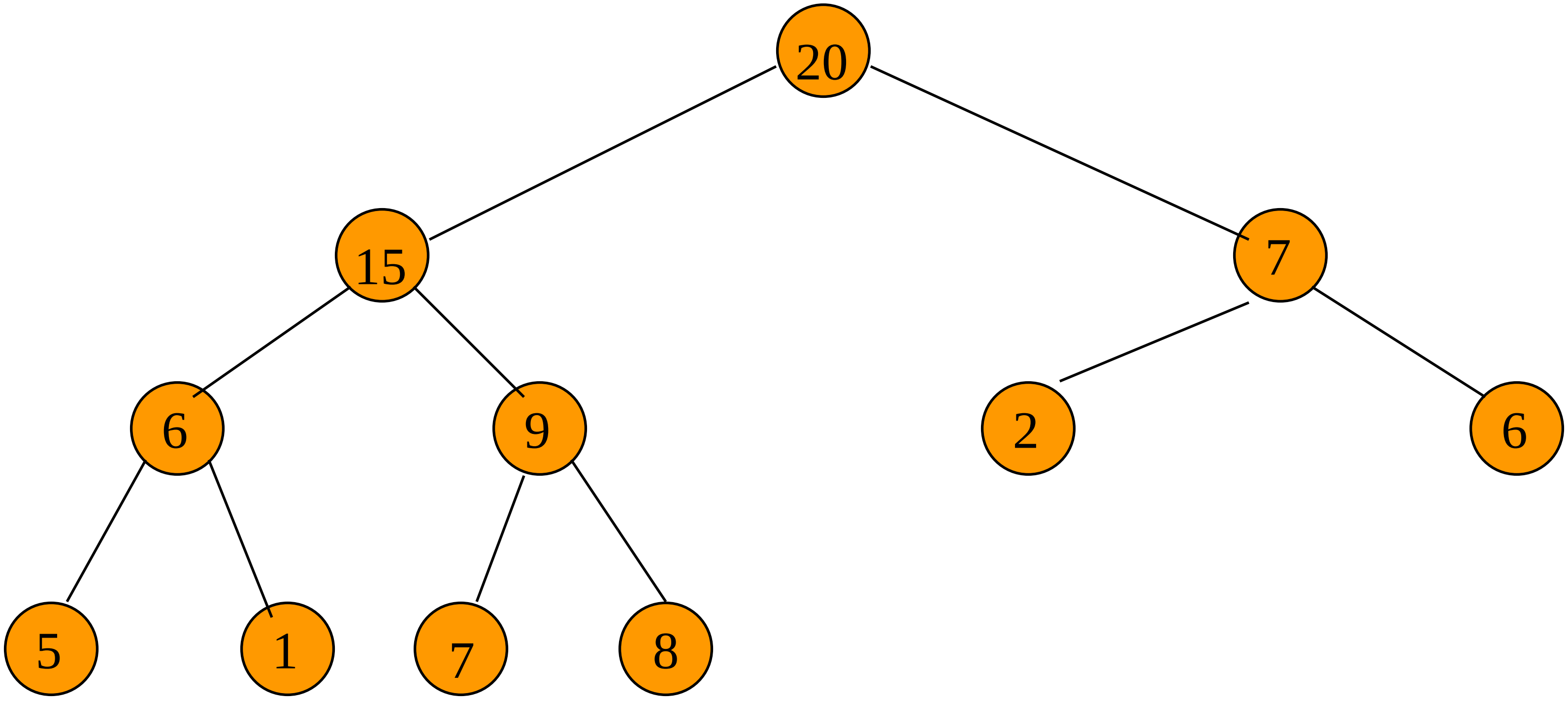


Complexity is O(log n), where n is heap size.

Deletion of the Root from a Max Heap

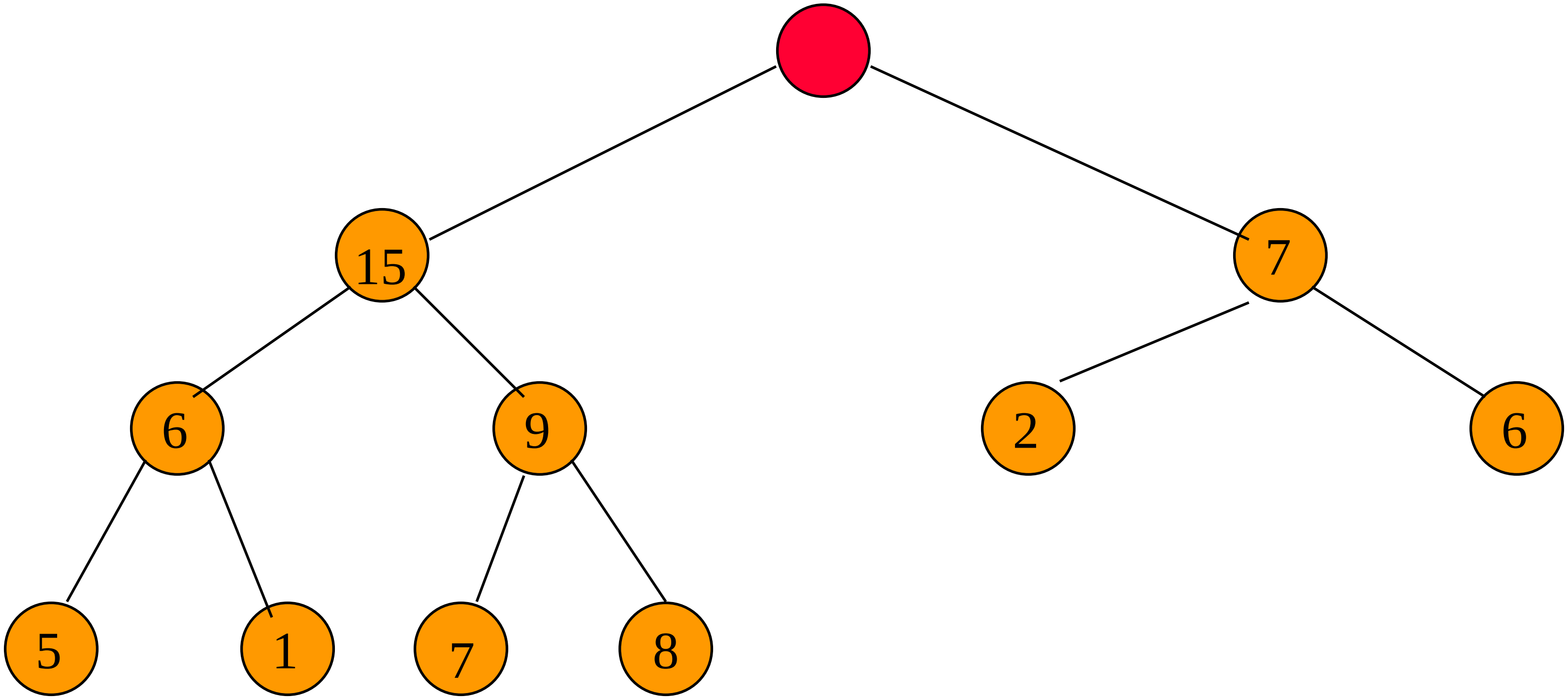
1. Replace the root of the heap with the last element on the last level
2. Compare the new root with its children; if they are in the correct order, stop
3. If not, swap the element with one of its children and return to the previous step
   – Swap with its smaller child in a min-heap and its larger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

Removing the Max Element



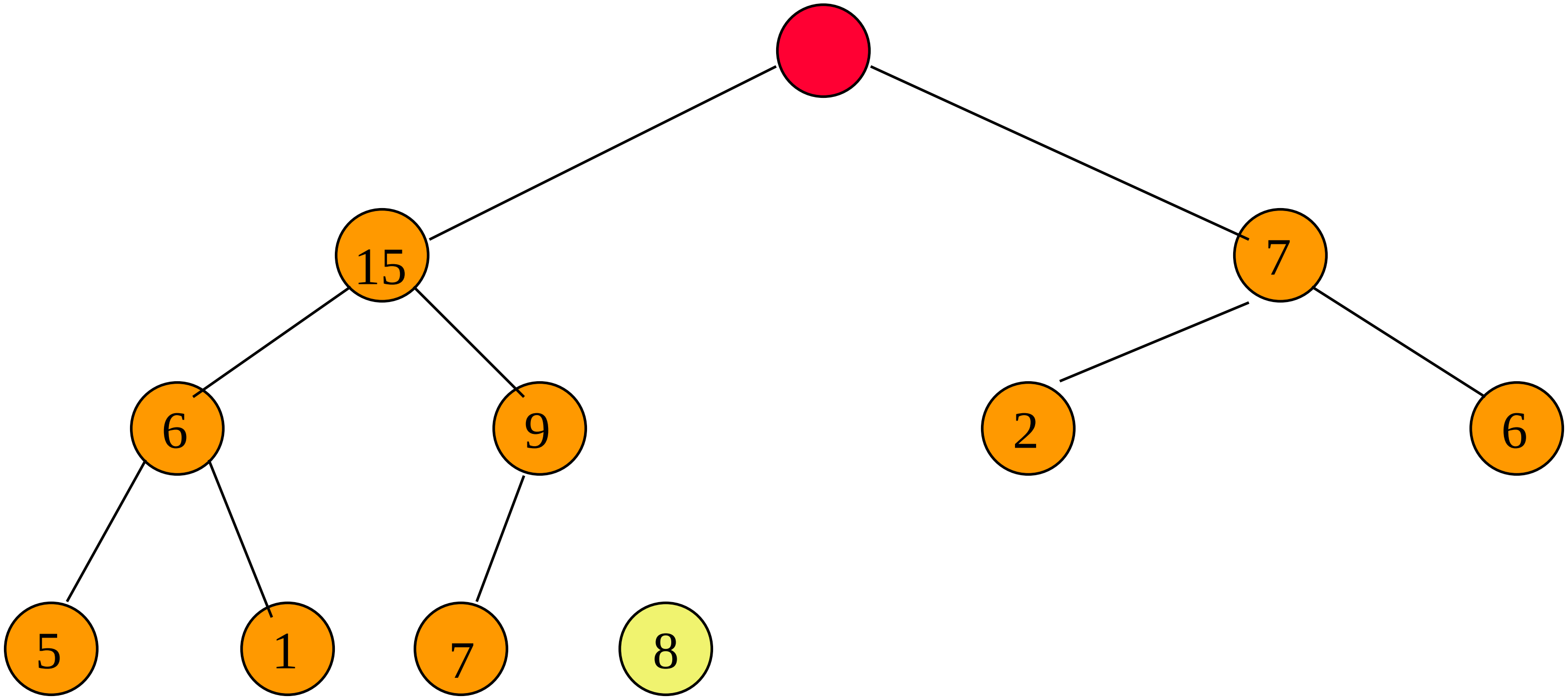
Max element is in the root.

Removing the Max Element



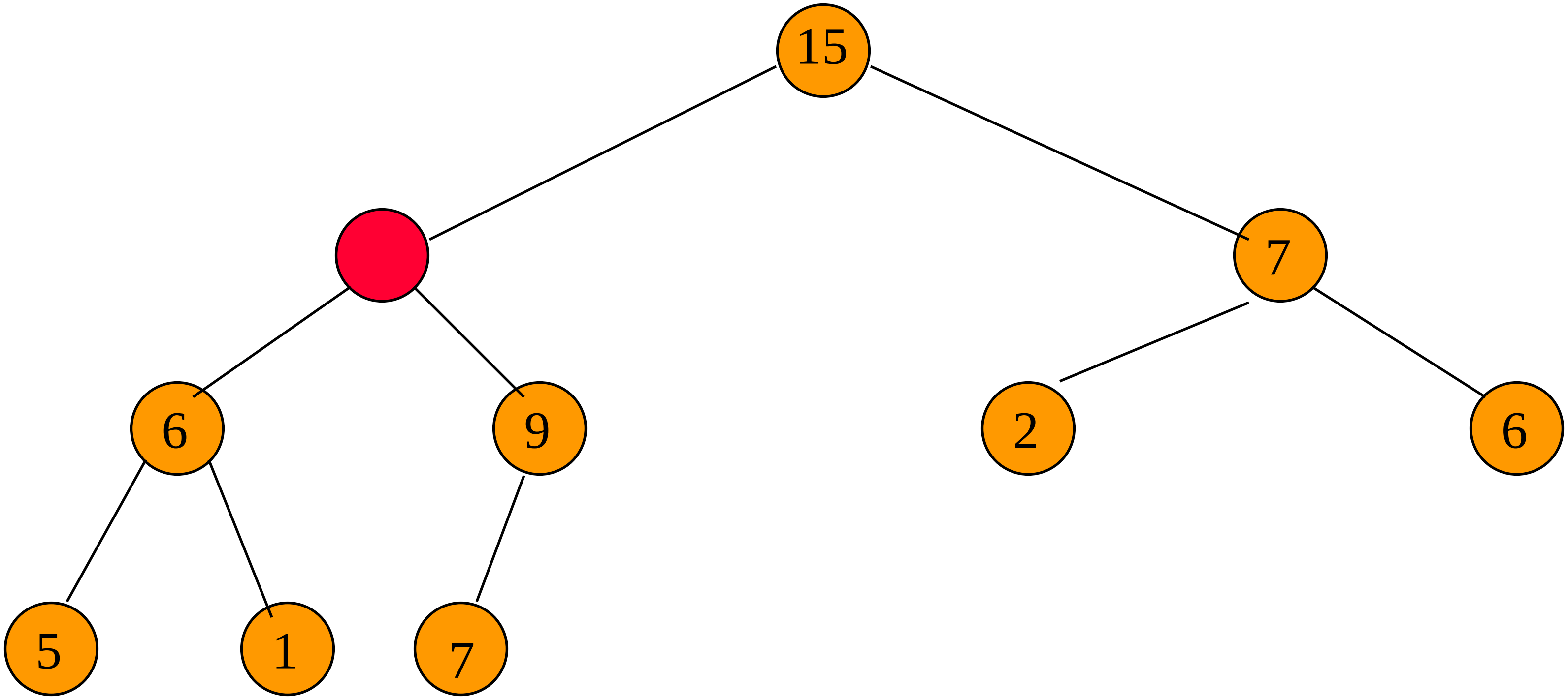
After max element is removed.

Removing the Max Element



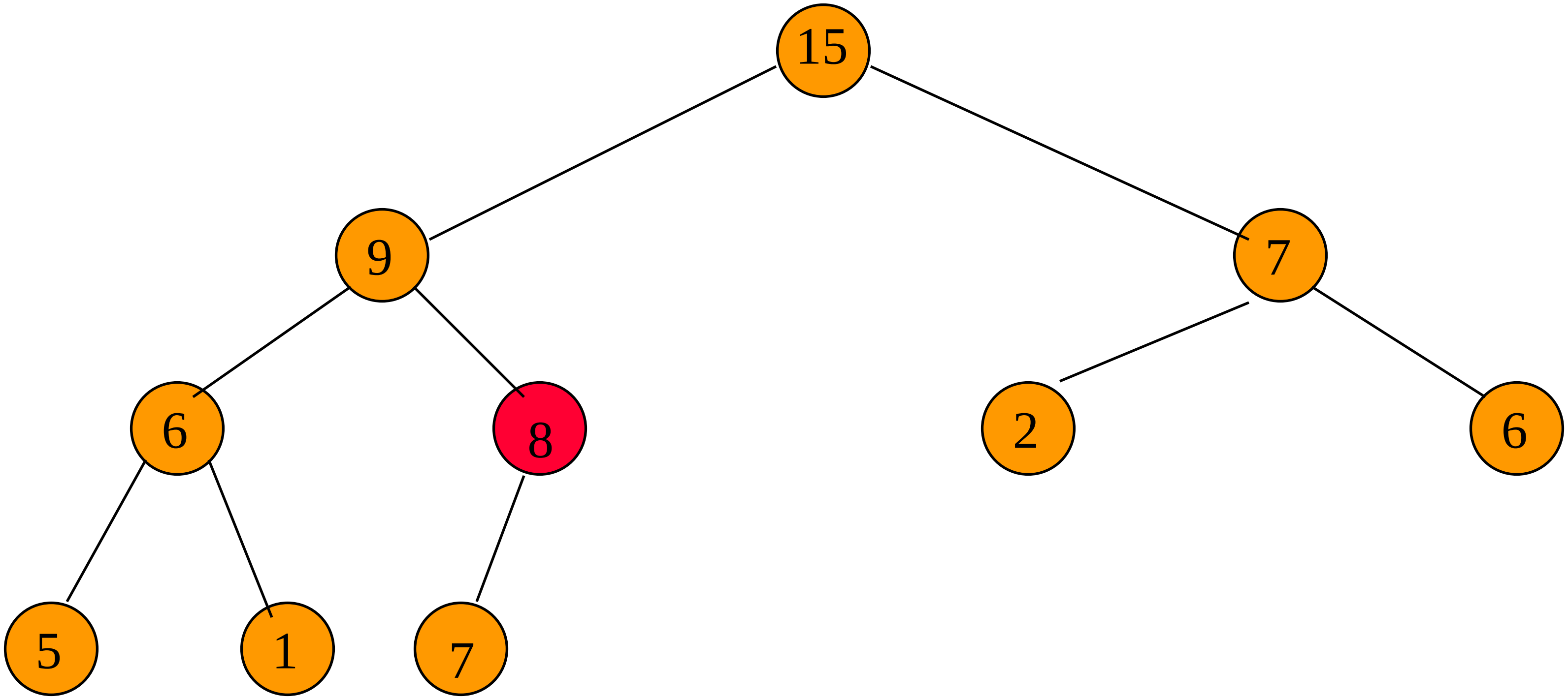
Reinsert 8 into the heap.

Removing the Max Element



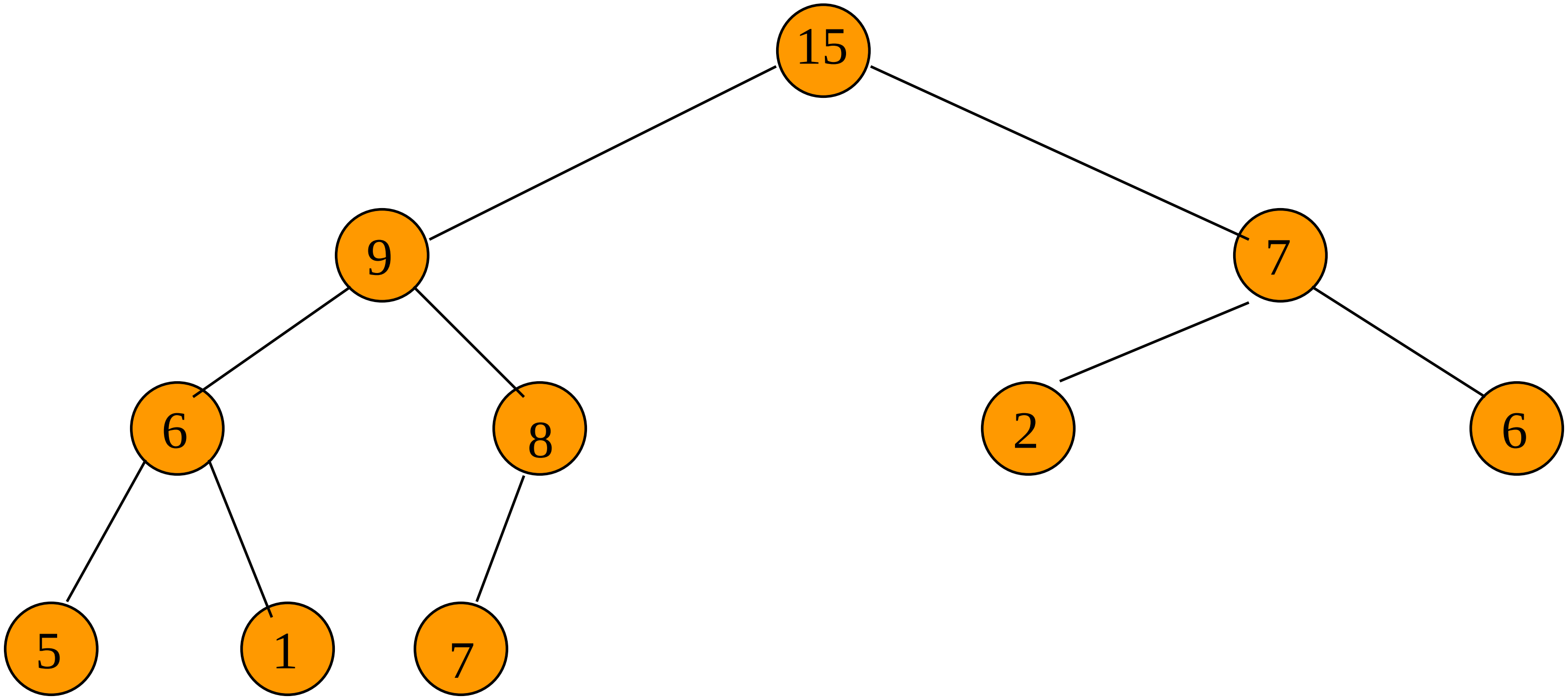
Reinsert 8 into the heap.

Removing the Max Element



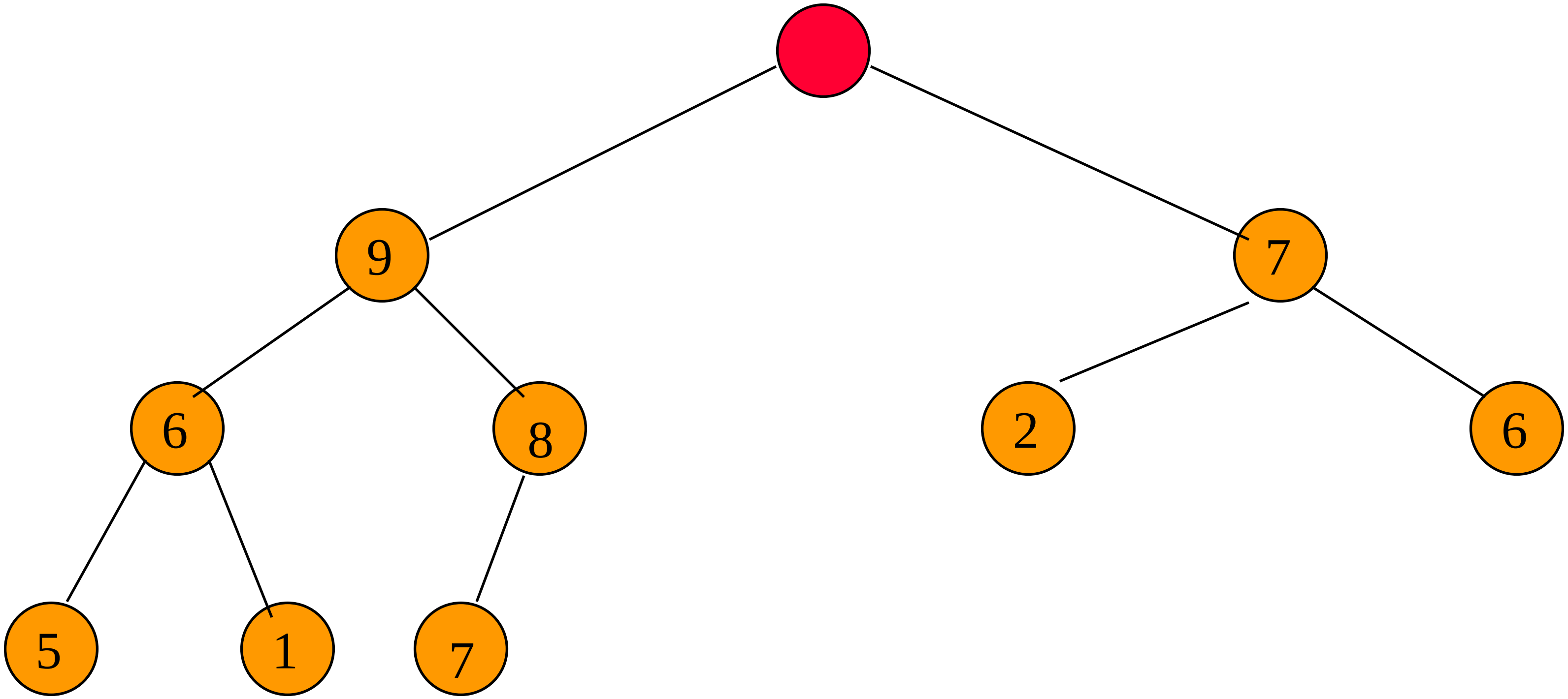
Reinsert 8 into the heap.

Removing the Max Element



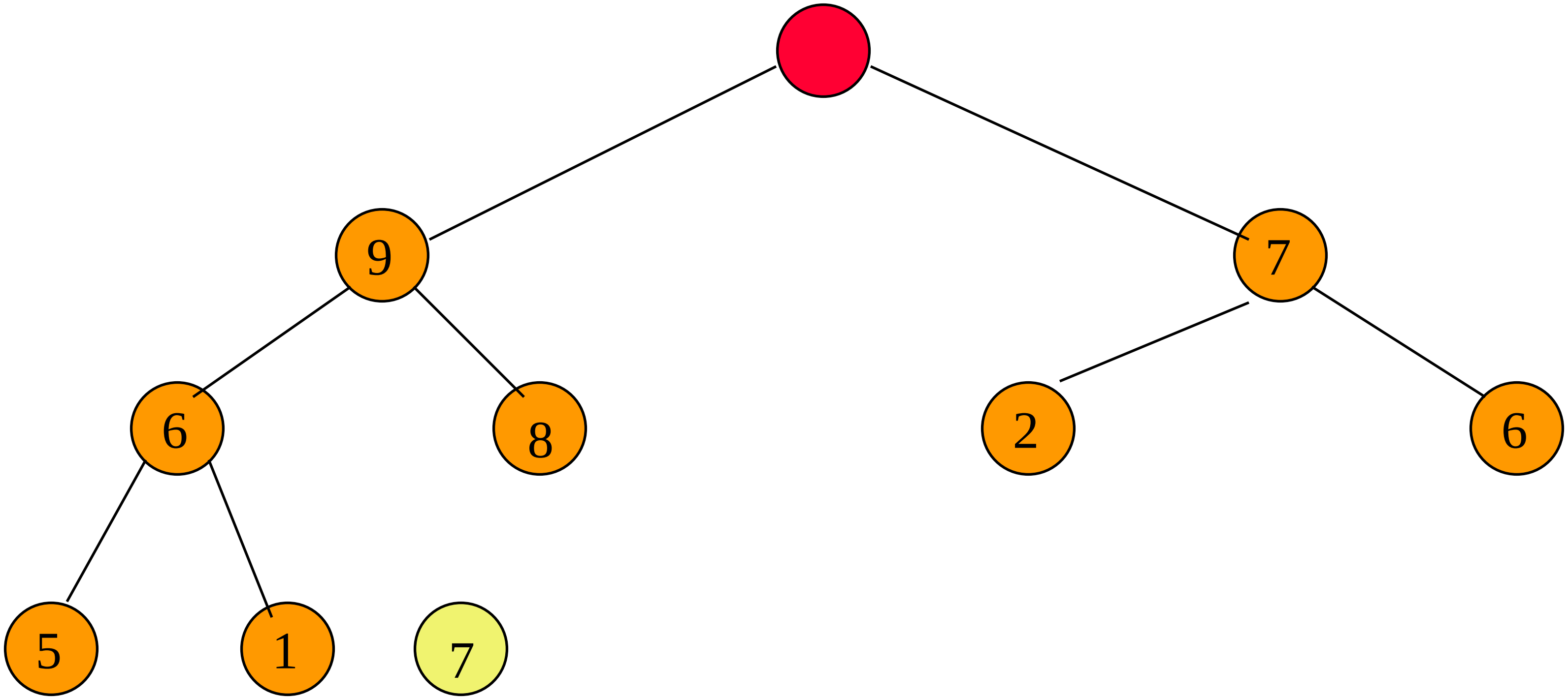
Max element is 15.

Removing the Max Element



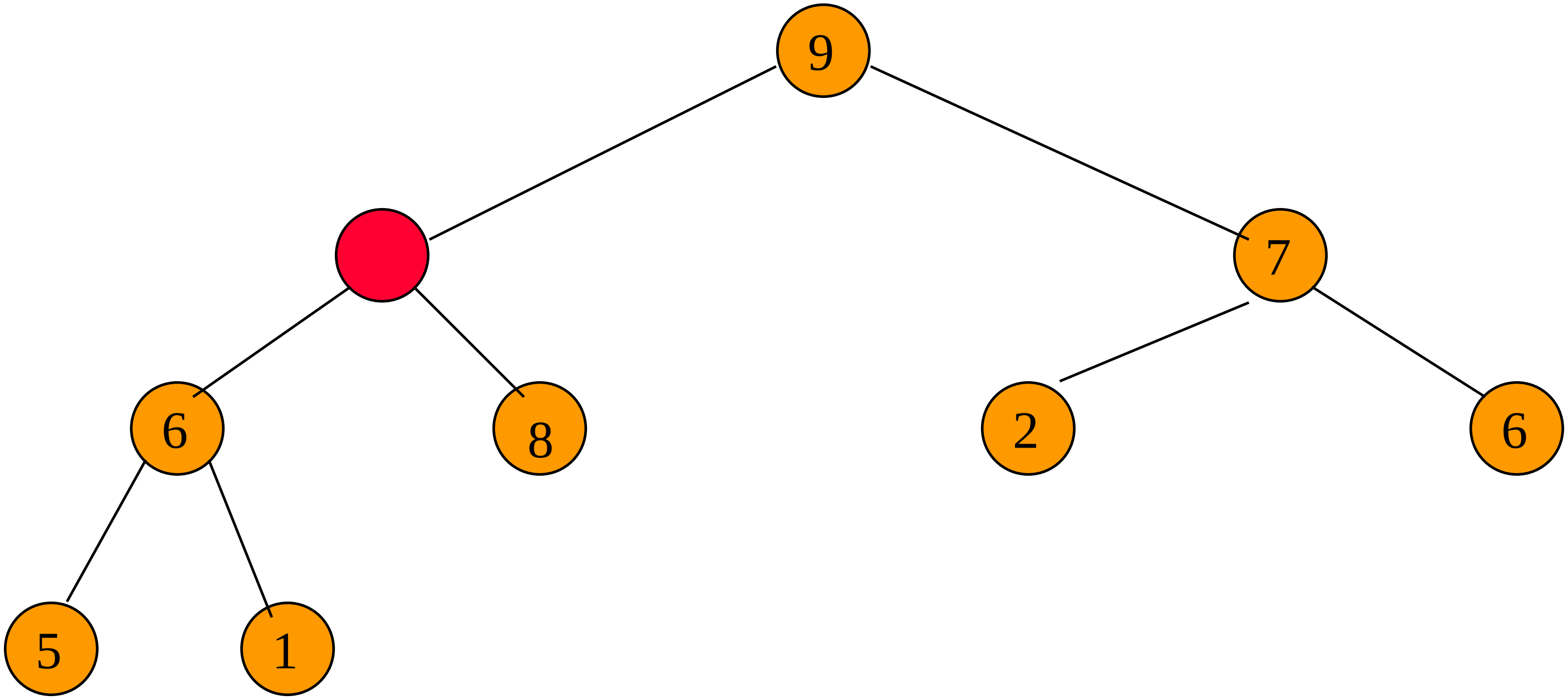
After max element is removed.

Removing the Max Element



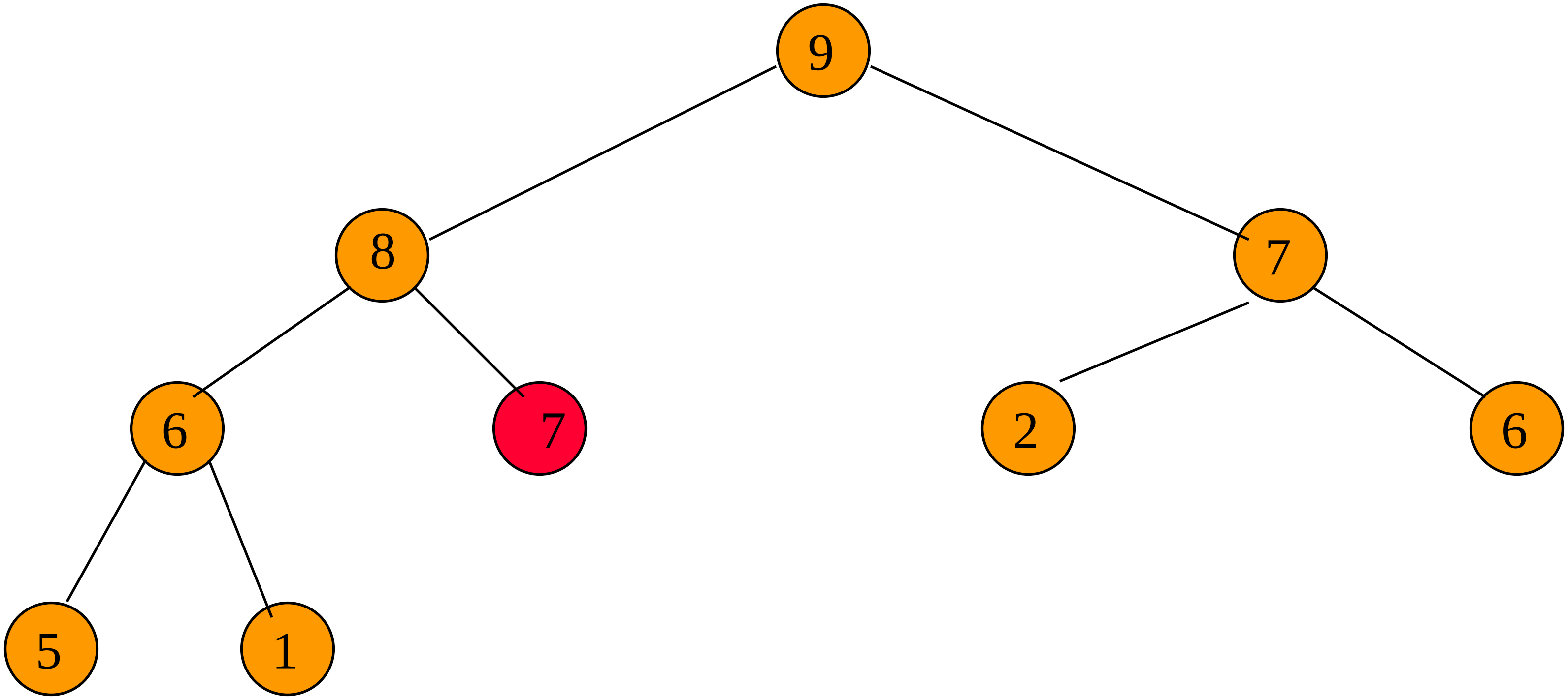
Reinsert 7.

Removing the Max Element



Reinsert 7.

Removing the Max Element

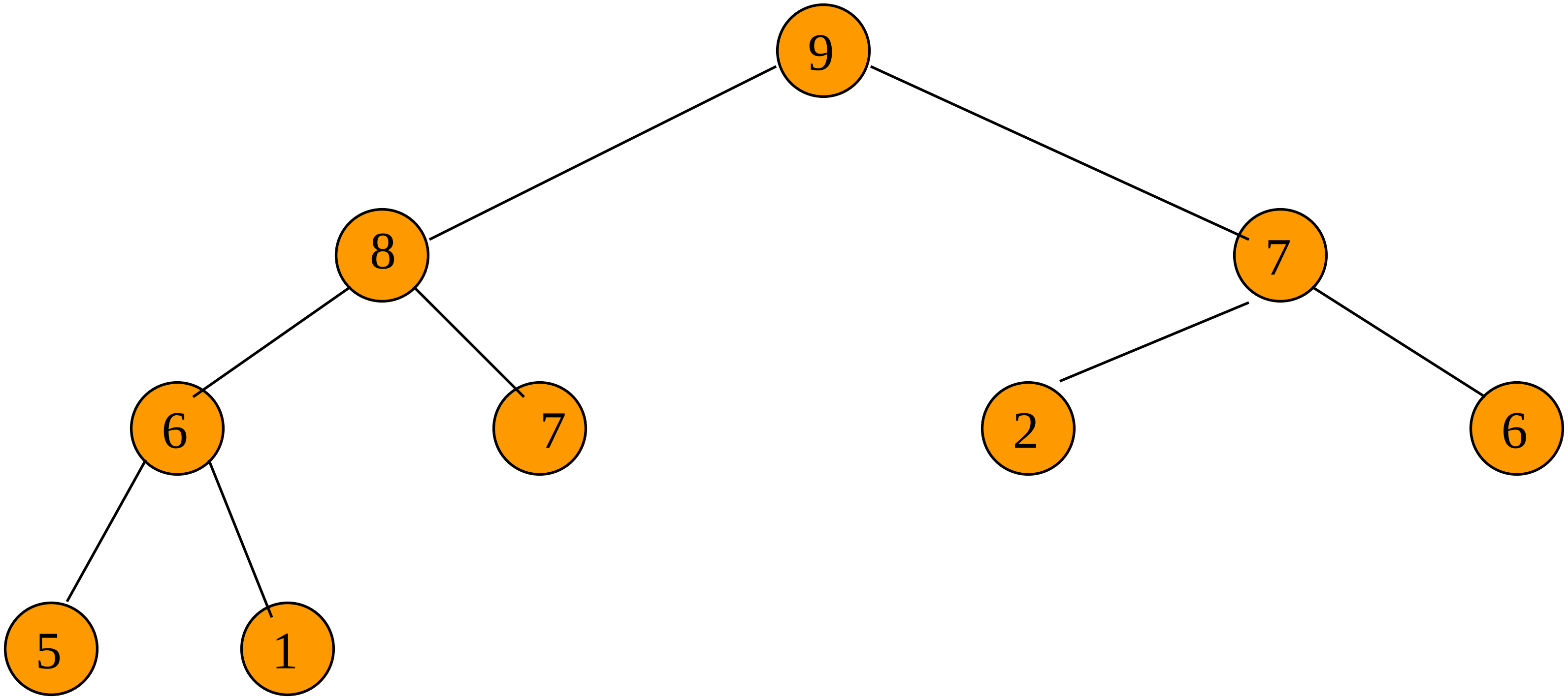


Reinsert 7.

Deletion from a max heap

```
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;
}
```

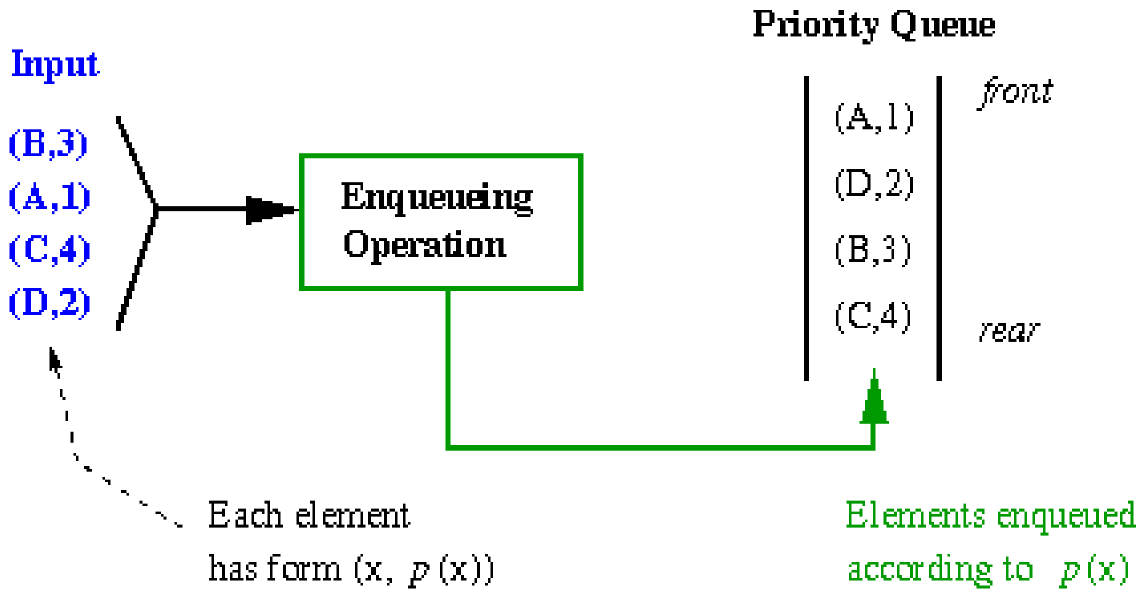
Complexity of Deletion



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
  - push elements
  - top/pop elements
    - 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

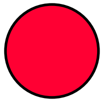
- Complexity of sorting n elements.
  - n insert operations => O(n log n) time.
  - n remove max operations => O(n log n) time.
  - total time is O(n log n).
  - compare with $O(n^2)$ for insertion or bubble sort

Binomial Heaps

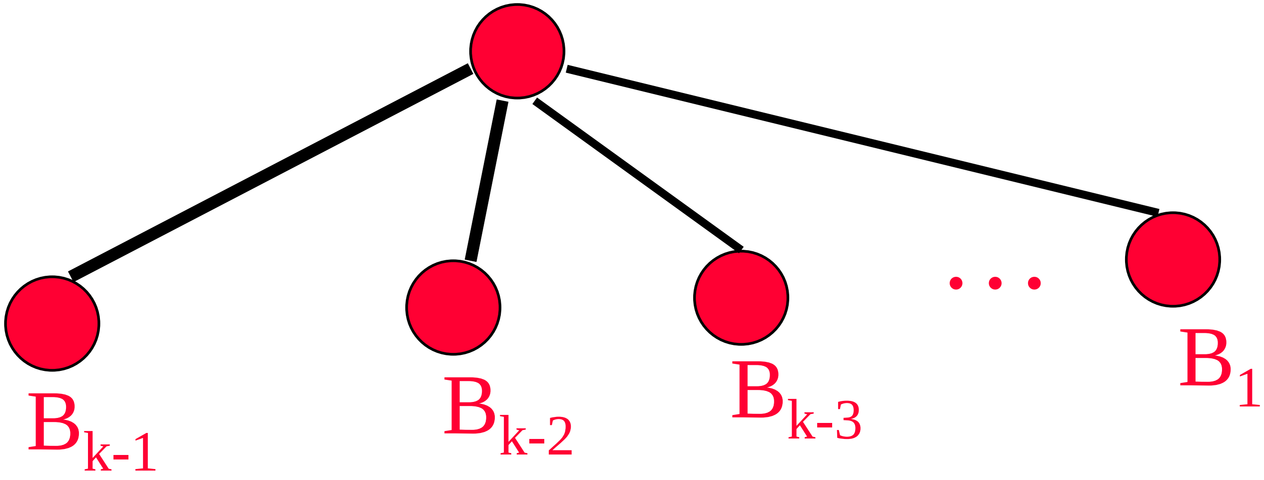
- Binomial heaps are similar to binary heaps, but binomial heaps allow for efficient merging of heaps.
  - Binary heap: O(n) for merging, O(log n) for insertion and deletion
  - Binomial heap: O(log n) for merging, insertion, and deletion
- A binomial heap is implemented as a set of binomial trees.

Binomial Trees

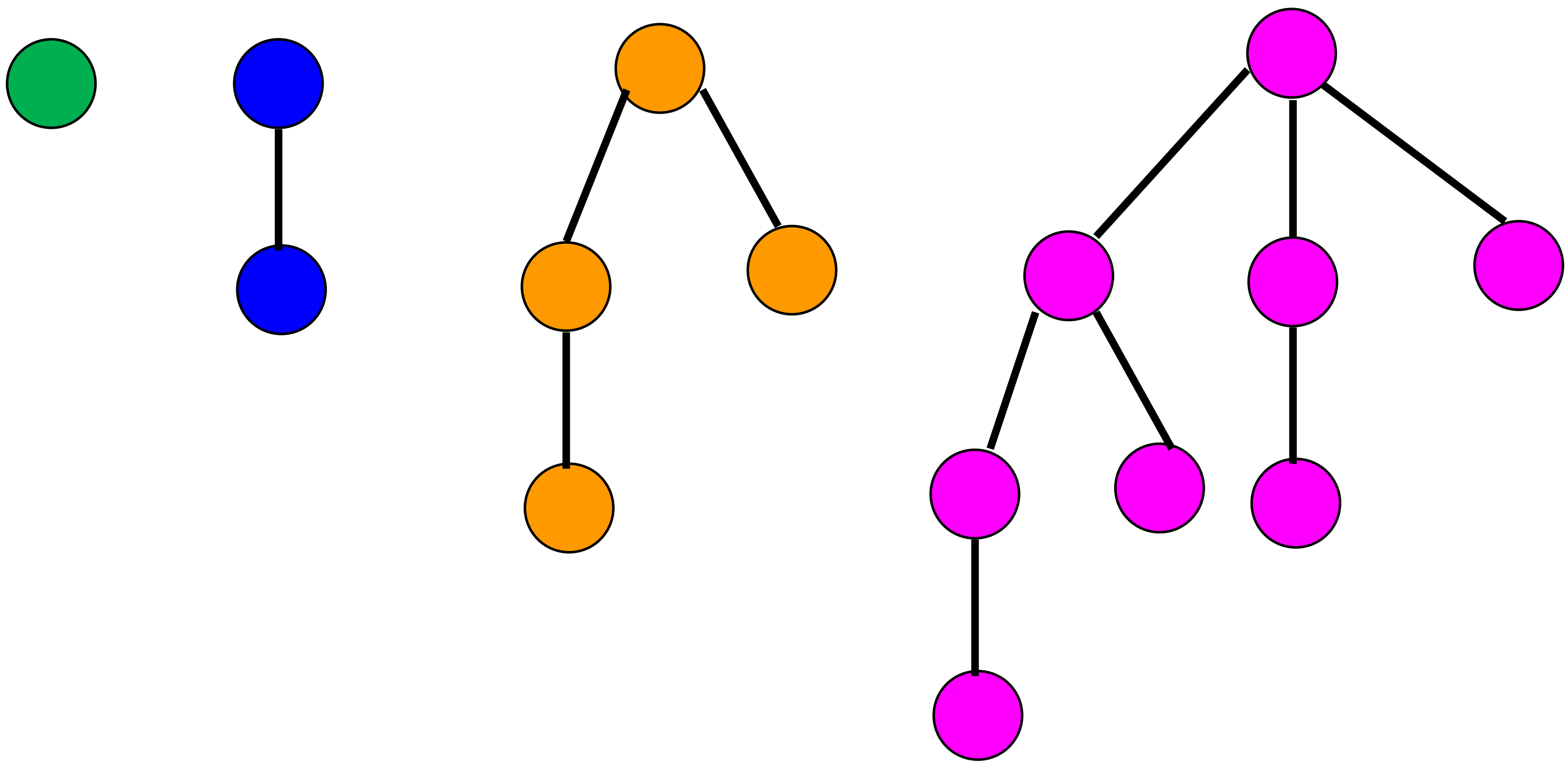
- B_k is degree k binomial tree.

B_0 

- B_k , $k > 0$, is:



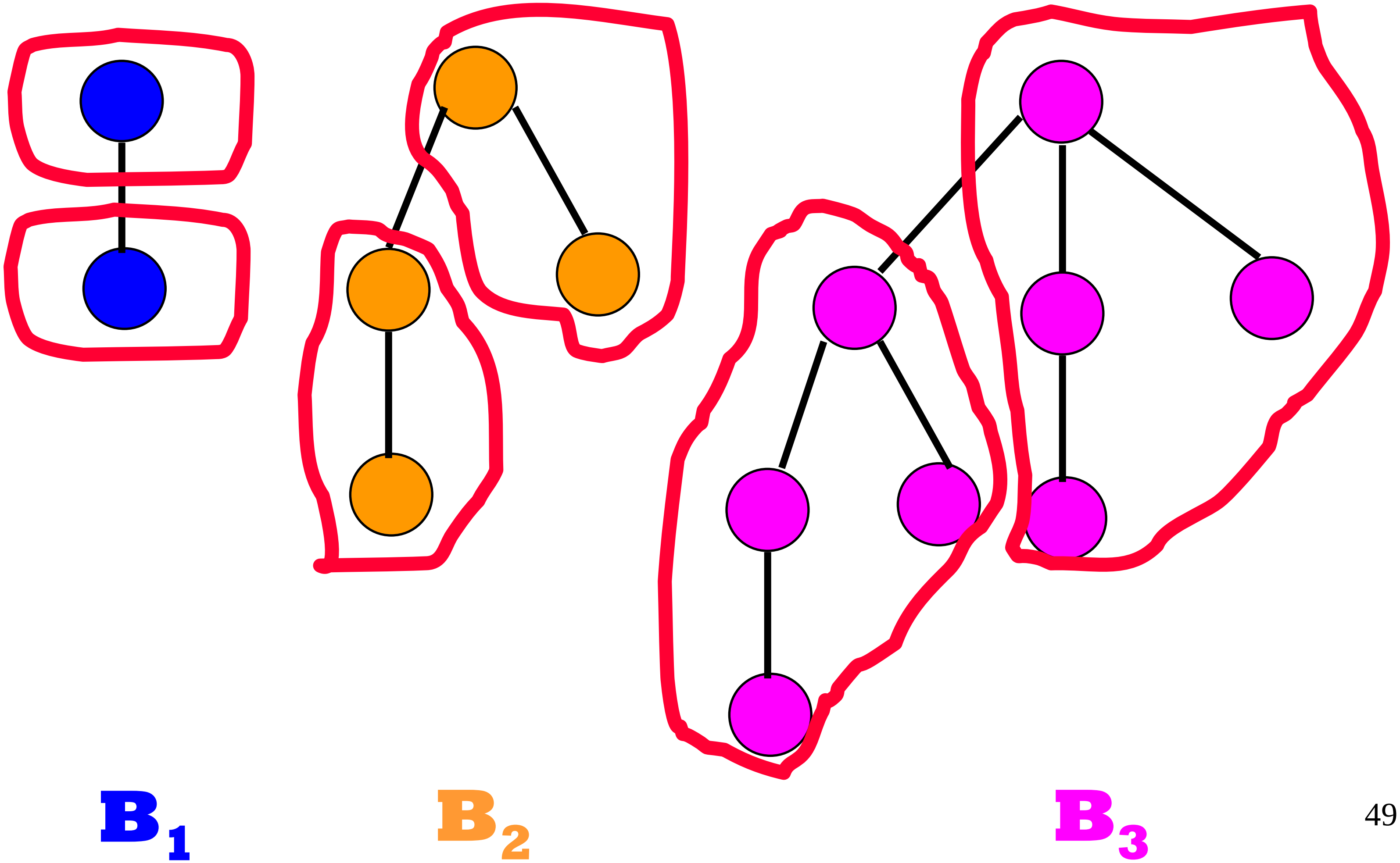
Examples



B_0 B_1 B_2 B_3

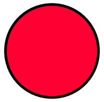
Number of Nodes in B_k

- B_k , $k > 0$, is two B_{k-1}s.
- One of these is a subtree of the other.

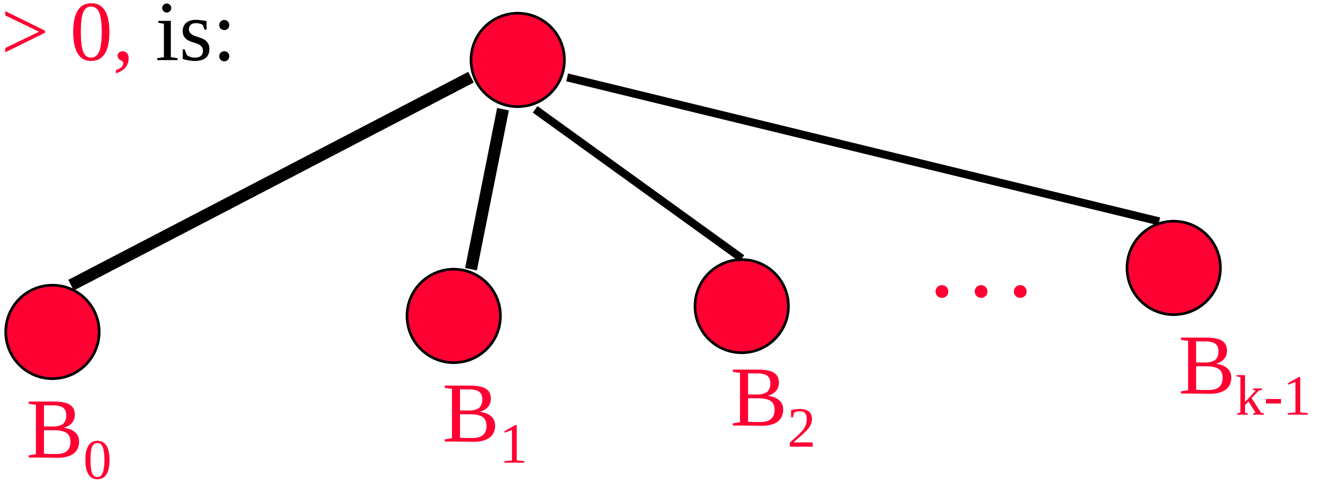


Number of Nodes in B_k

- $N_k =$ number of nodes in B_k

B_0  $N_0 = 1$

- B_k , $k > 0$, is:



- $N_k = 2^0 + 2^1 + 2^2 + \ldots + 2^{k-1} + 1 = 2^k$
- $k = \log_2 N_k$

Binomial Min Heap

- A binomial min heap H is a set of binomial trees that satisfies the following properties:

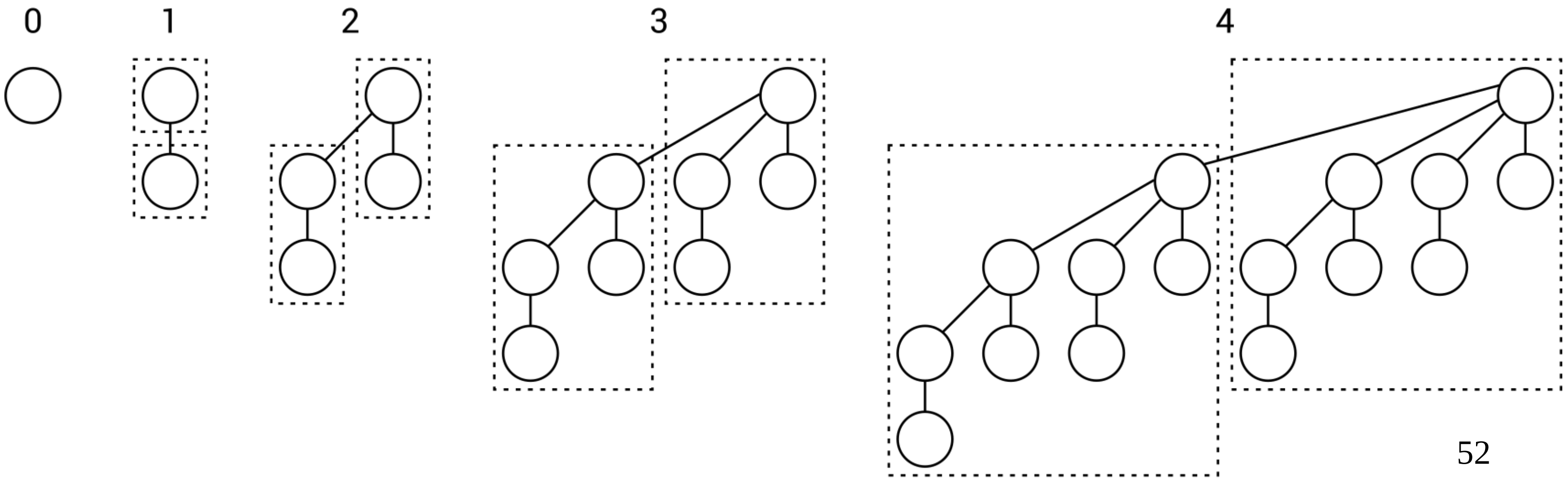
1. Each binomial tree in H obeys the min-heap property: the key of a node is greater than or equal to the key of its parent.
2. For any nonnegative integer k, there is at most one binomial tree in H whose root has degree k.

Binomial Heap

- For any nonnegative integer k, there is at most one binomial tree in H whose root has degree k.
- An n-node binomial heap H consists of $O(\log_2 n)$ binomial trees.

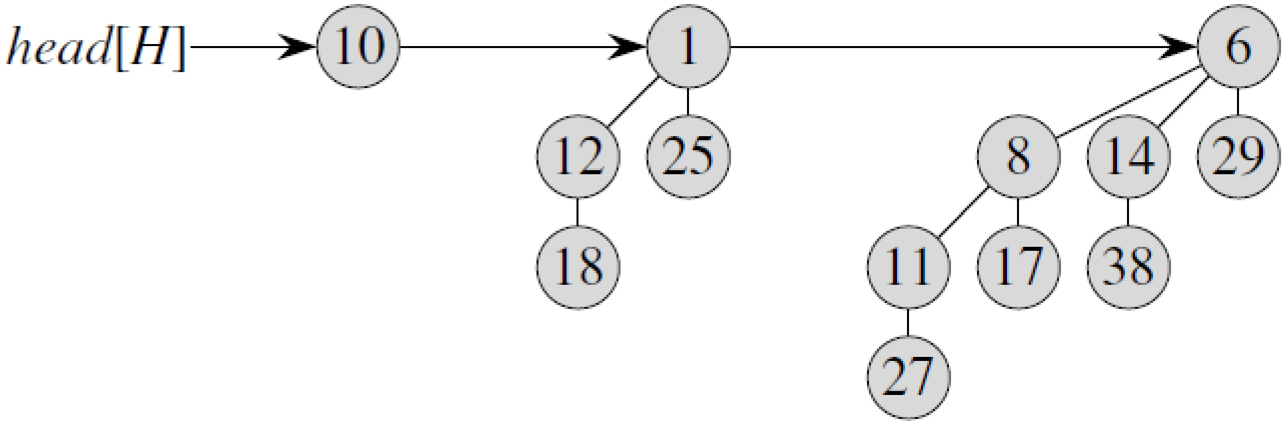
$$n = 2^0 + 2^1 + 2^2 + \ldots + 2^k = 2^{k+1} - 1$$

$$k + 1 = \log_2(n+1)$$



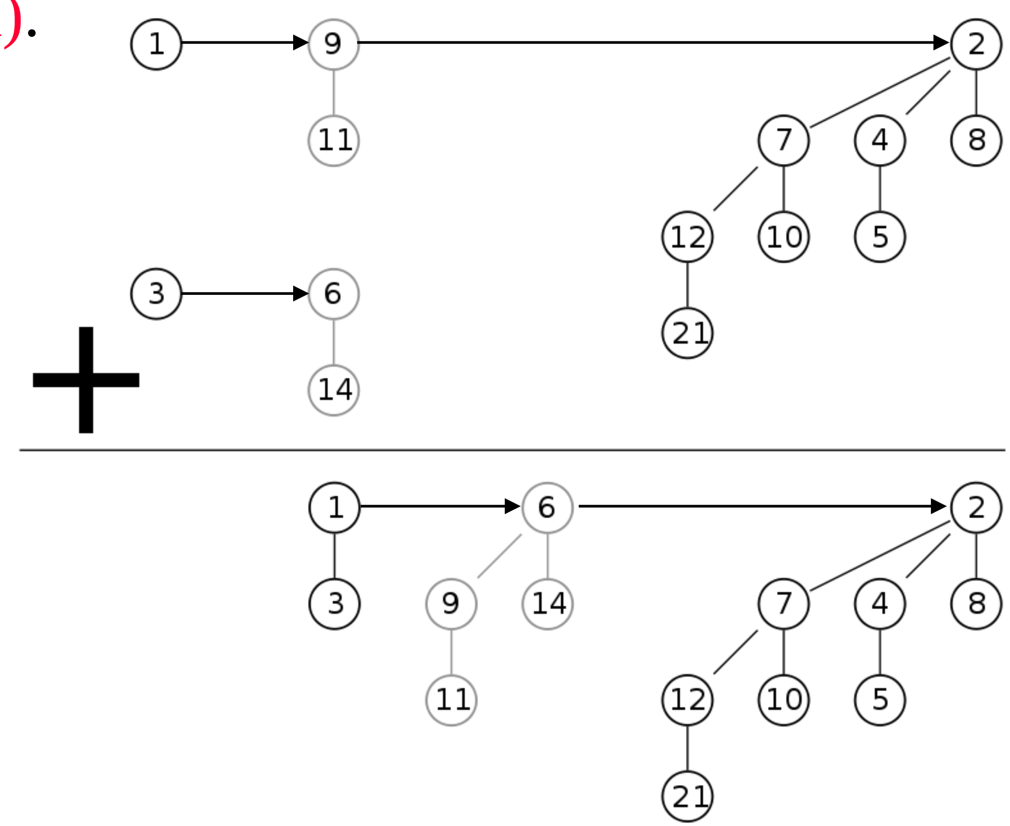
Binomial Heap Representation

- The roots of the binomial trees within a binomial heap are organized in a linked list called the *root list*.
- The degrees of the roots strictly increase as we traverse the root list.

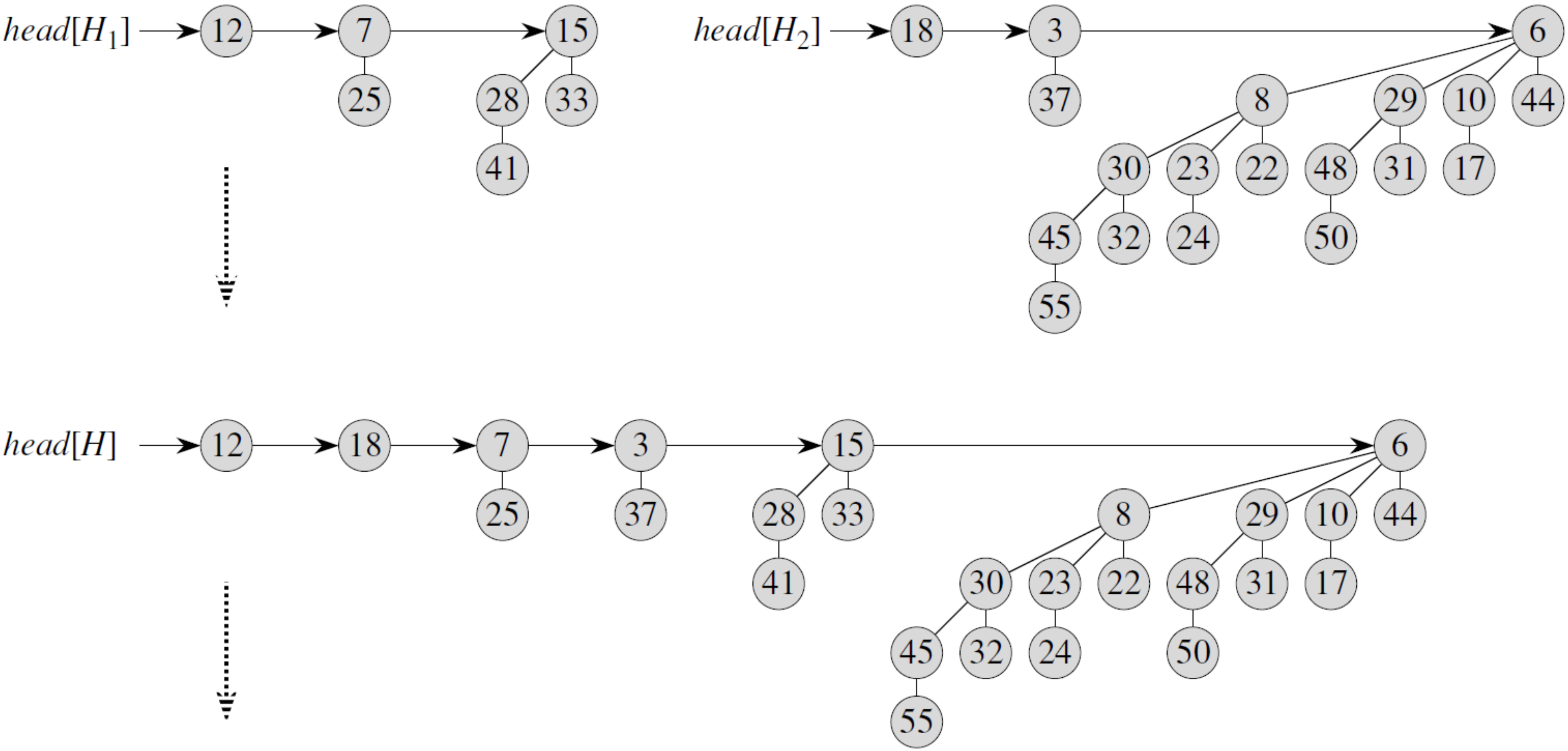


Merging Binomial Heaps

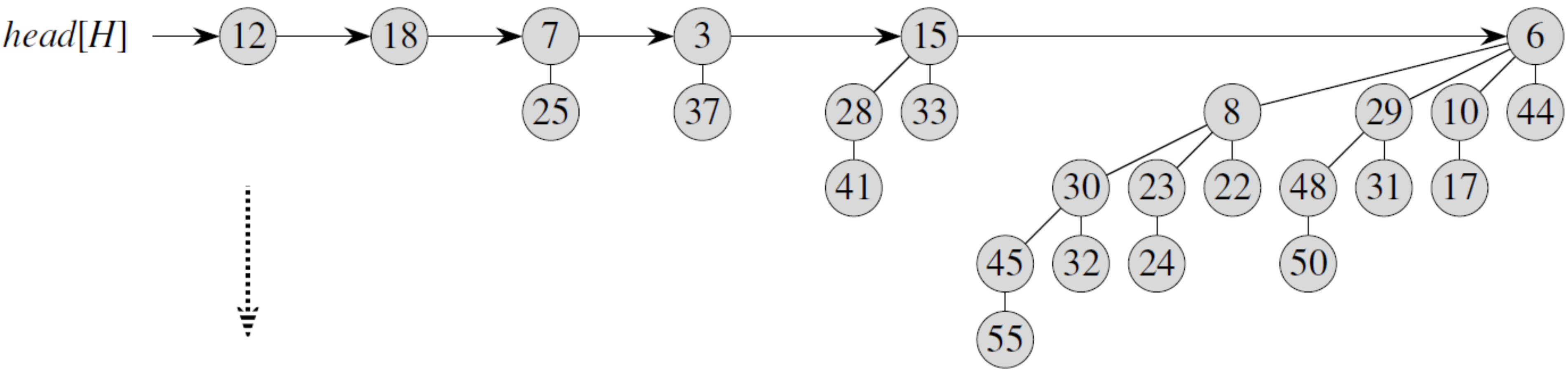
- Merging two binomial trees of the same degree one by one.
  - The root node with the larger key is made into a child of the root node with the smaller key.
  - Complexity is O(log n).

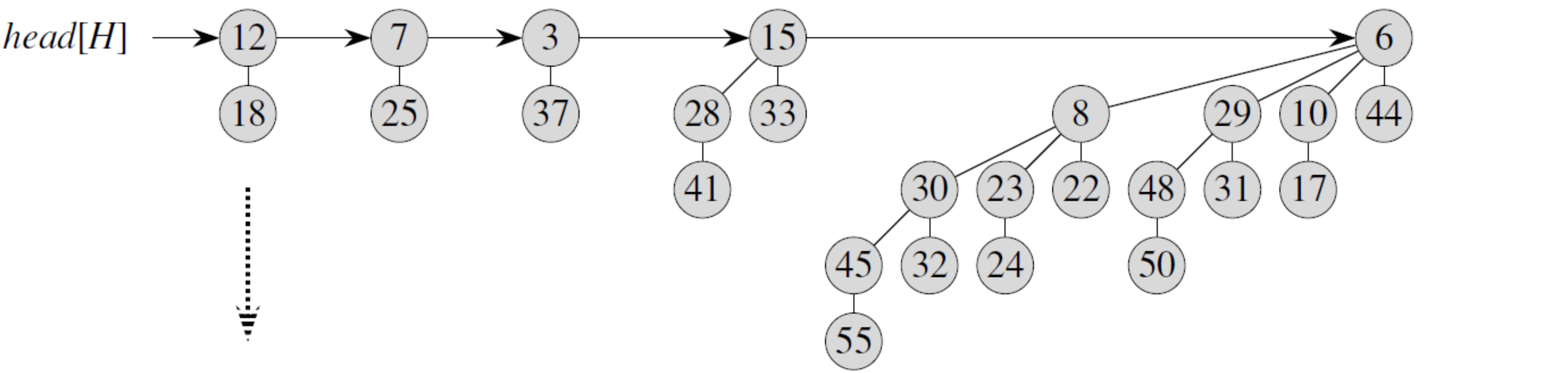


Merging Binomial Heaps



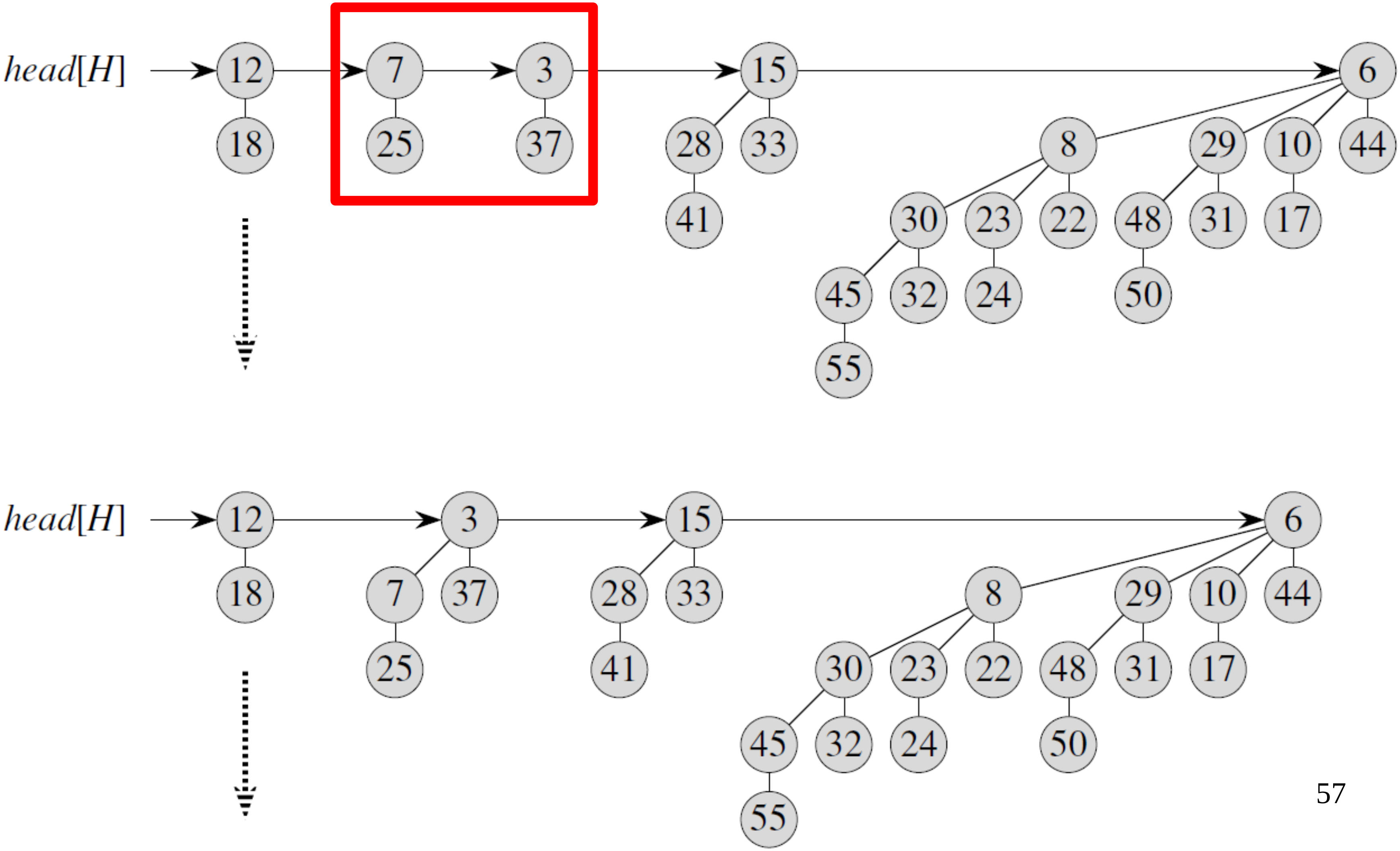
Merging Binomial Heaps



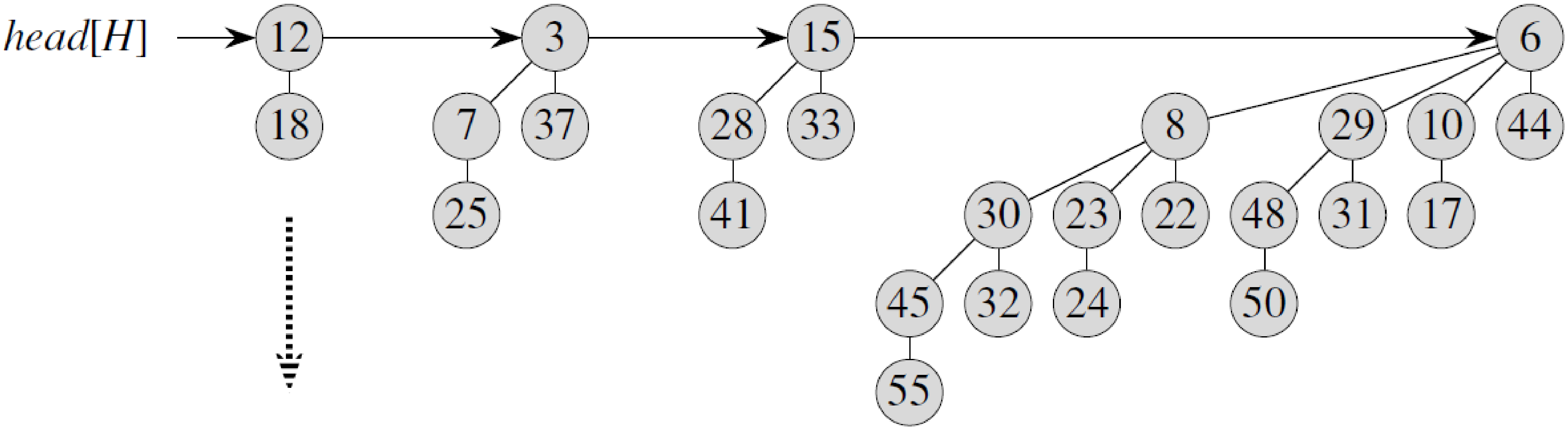


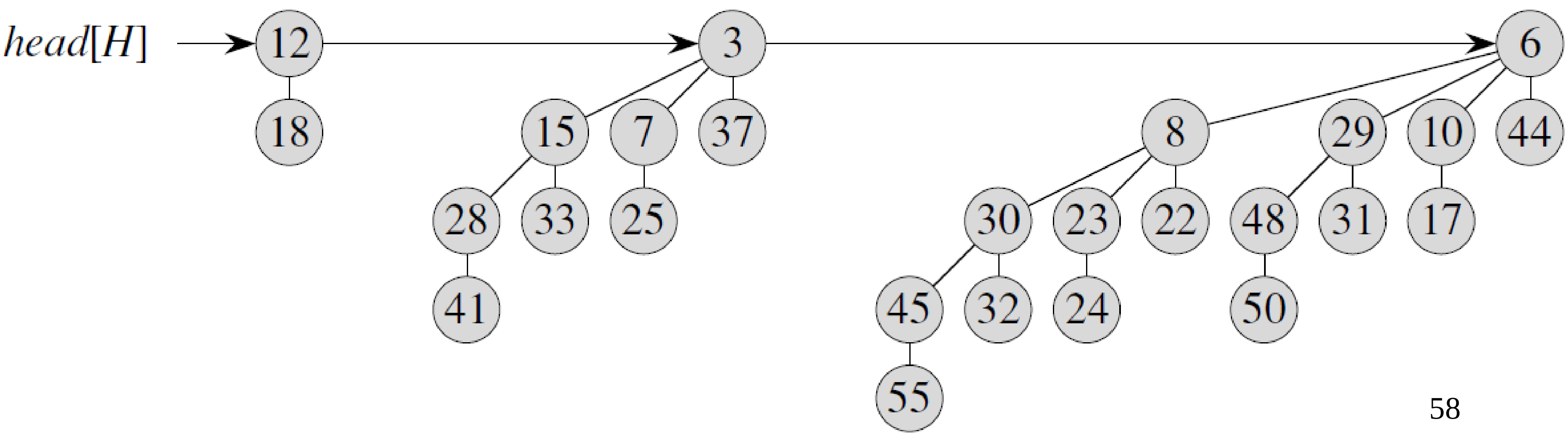
Merging Binomial Heaps

- There may be three roots of a given degree appearing on the root list at some time.



Merging Binomial Heaps





Inserting an Element

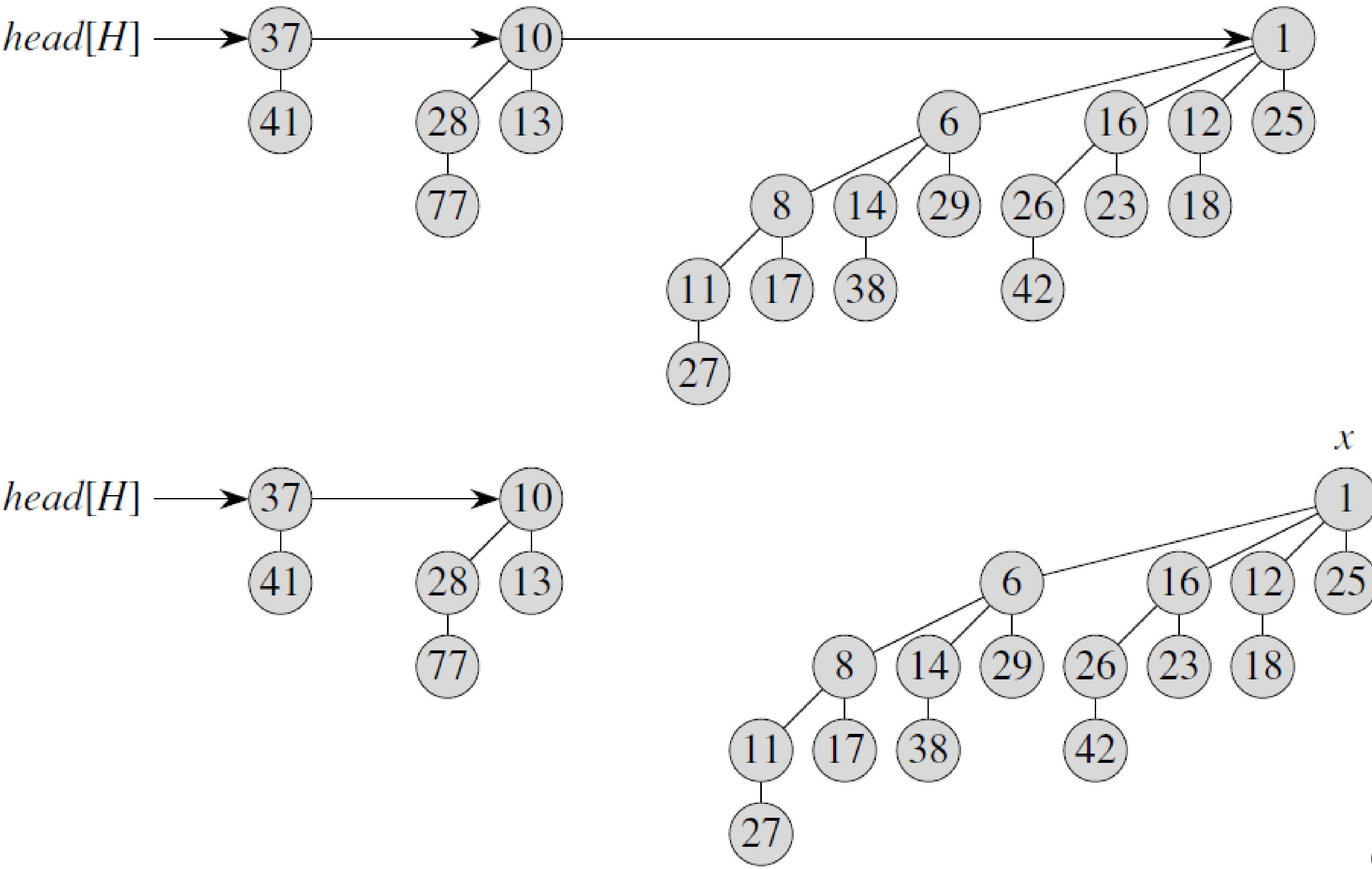
- Creating a one-node binomial heap and then merging it with the original heap.

- Complexity is $O(\log n)$.

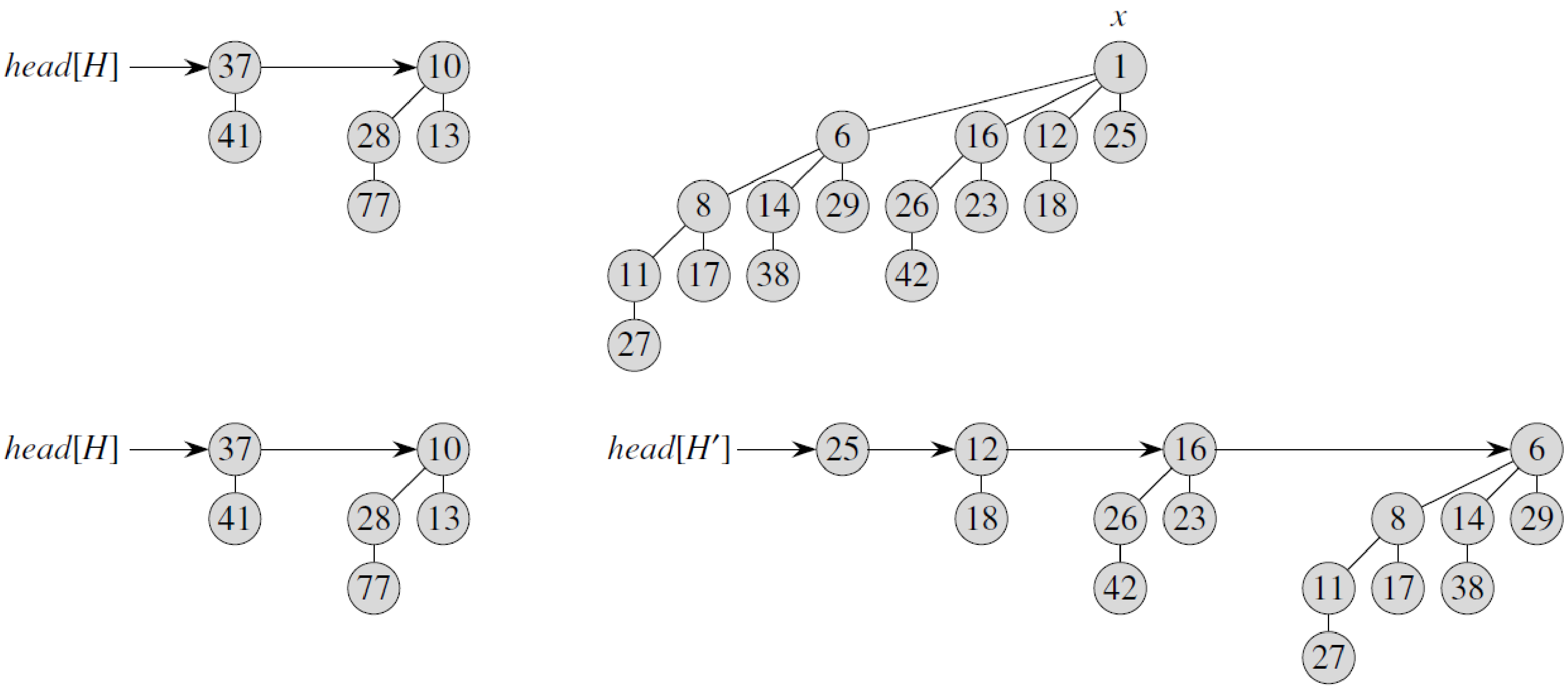
Deleting the Min Element

- Deleting the minimum element from the binomial heap *H*:
  1. Finding the root *x* with the minimum key in the root list of *H*.
  2. Removing *x* from its binomial tree, and obtain a list of its child subtrees.
  3. Transforming this list of subtrees into a separate binomial heap.
  4. Merging this heap with the original heap.
- Complexity is $O(\log n)$.

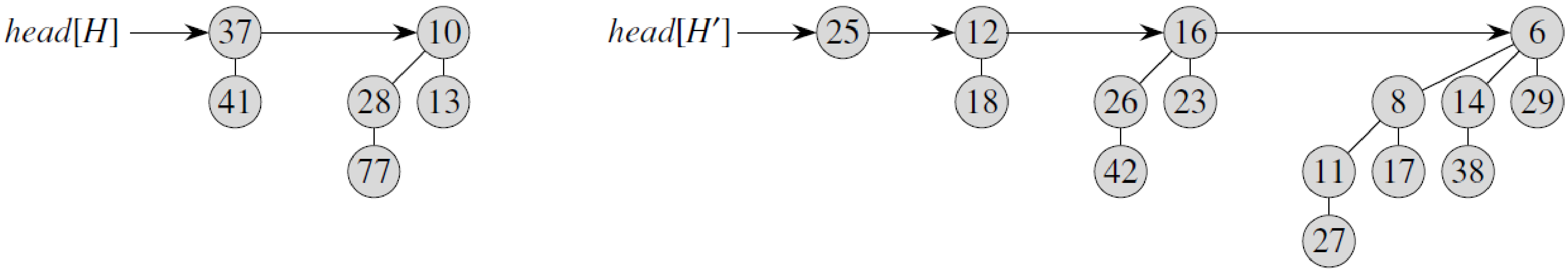
Deleting the Min Element

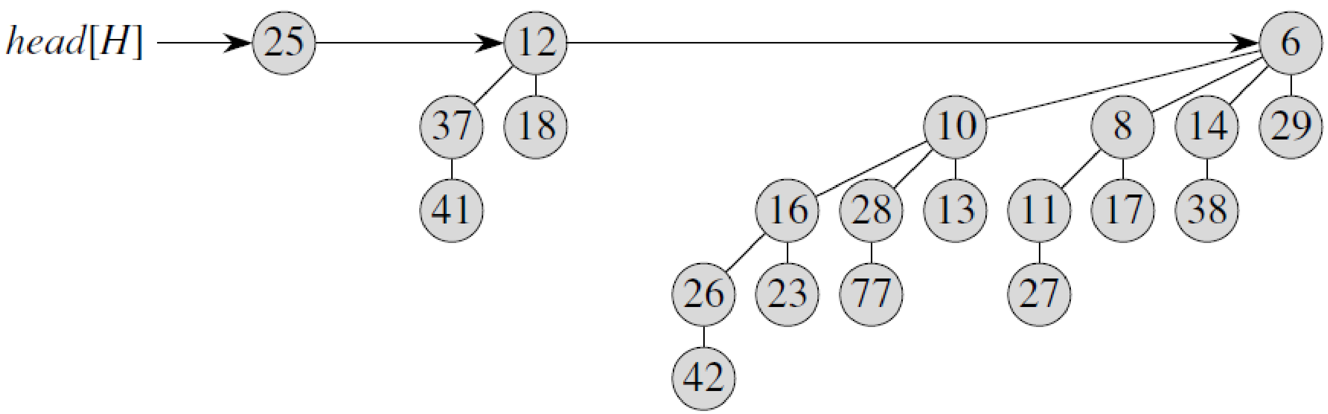


Deleting the Min Element



Deleting the Min Element





Homework

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

Homework을 제출할 필요는 없으나
중간/기말고사에 출제할 계획임