Department of Computer Sciences and IT,
The University of Lahore

Heap Data Structure

Rao Muhammad Umer

Lecturer,

CS and IT Department,

The University of Lahore.

Web: raoumer.com

Outlines

- Heap
- Max/Min Heap
- Operations on Heap
- Build Heap
- Complexity Analysis of Heap
- Binomial Heap
- Fibonacci Heap
- Applications of Heap
  - Heap Sort
  - Priority Queue
  - Event-Driven Simulation

Heap Data Structure

- **Heap:** A special form of complete binary tree that key value of each node is no smaller (larger) than the key value of its children (if any).
- Heaps are based on the notion of a **complete tree**
- A binary tree is **completely full** if it is of height, *h*, and has 2^{h+1}-1 nodes.

Complete Binary Tree

- A binary tree of height, *h*, is **complete** *iff* :
  - ➢ it is empty ***OR***
  - ➢ its left subtree is complete of height *h*-1 and its right subtree is completely full of height *h*-2 **or**
  - ➢ its left subtree is completely full of height *h*-1 and its right subtree is complete of height *h*-1.
- A complete tree is filled from the left

A complete binary tree in nature

Binary tree in Computing

Max/Min Tree

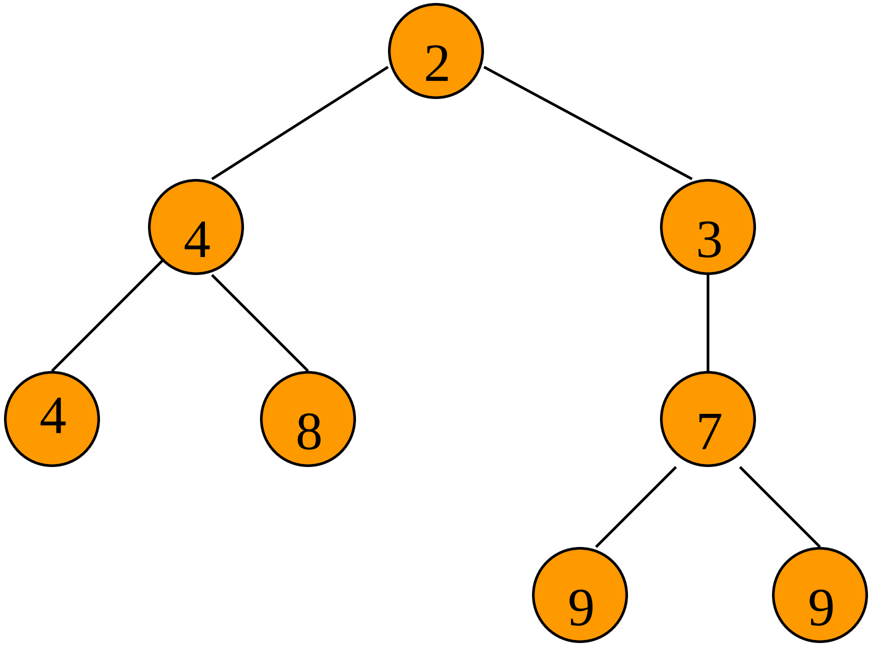
Max-Tree:

A *max tree* is a tree in which the key value in each node is no smaller than the key values in its children.

Min-Tree:

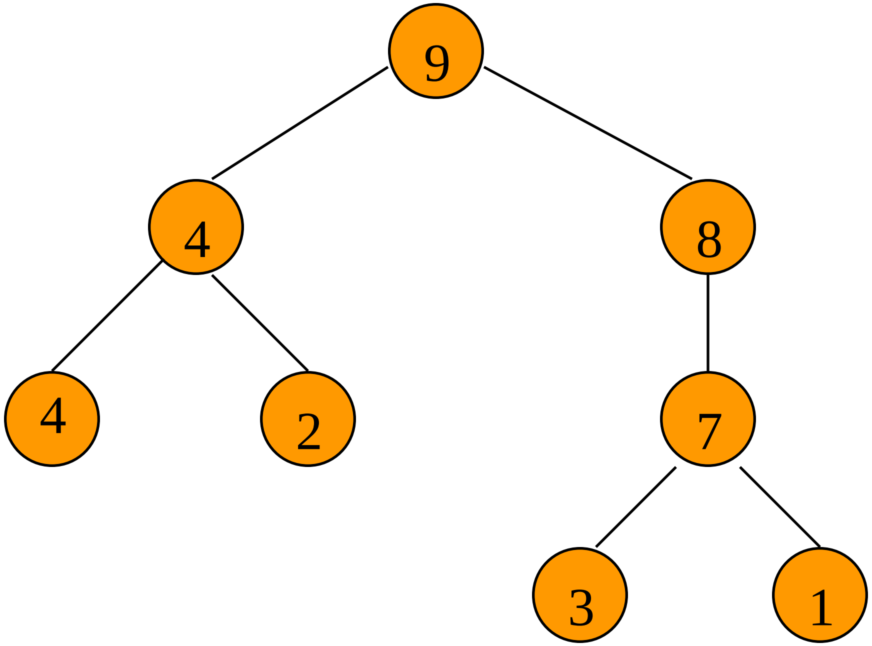
A *min tree* is a tree in which the key value in each node is no larger than the key values in its children.

Min Tree Example



Root has minimum element.

Max Tree Example



Root has maximum element.

Max/Min Heap

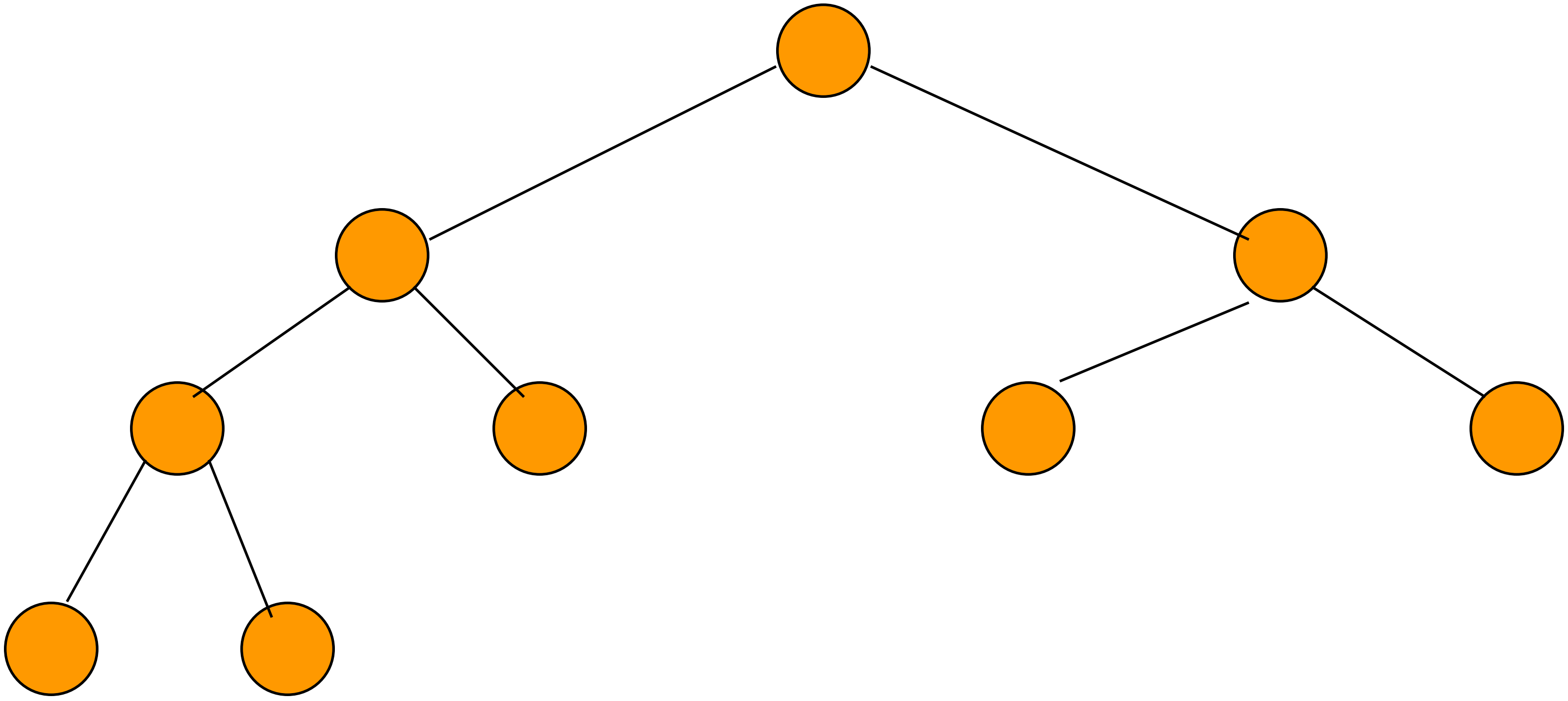
Max-Heap: root node has the **largest** key.

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

Min-Heap: root node has the **smallest** key.

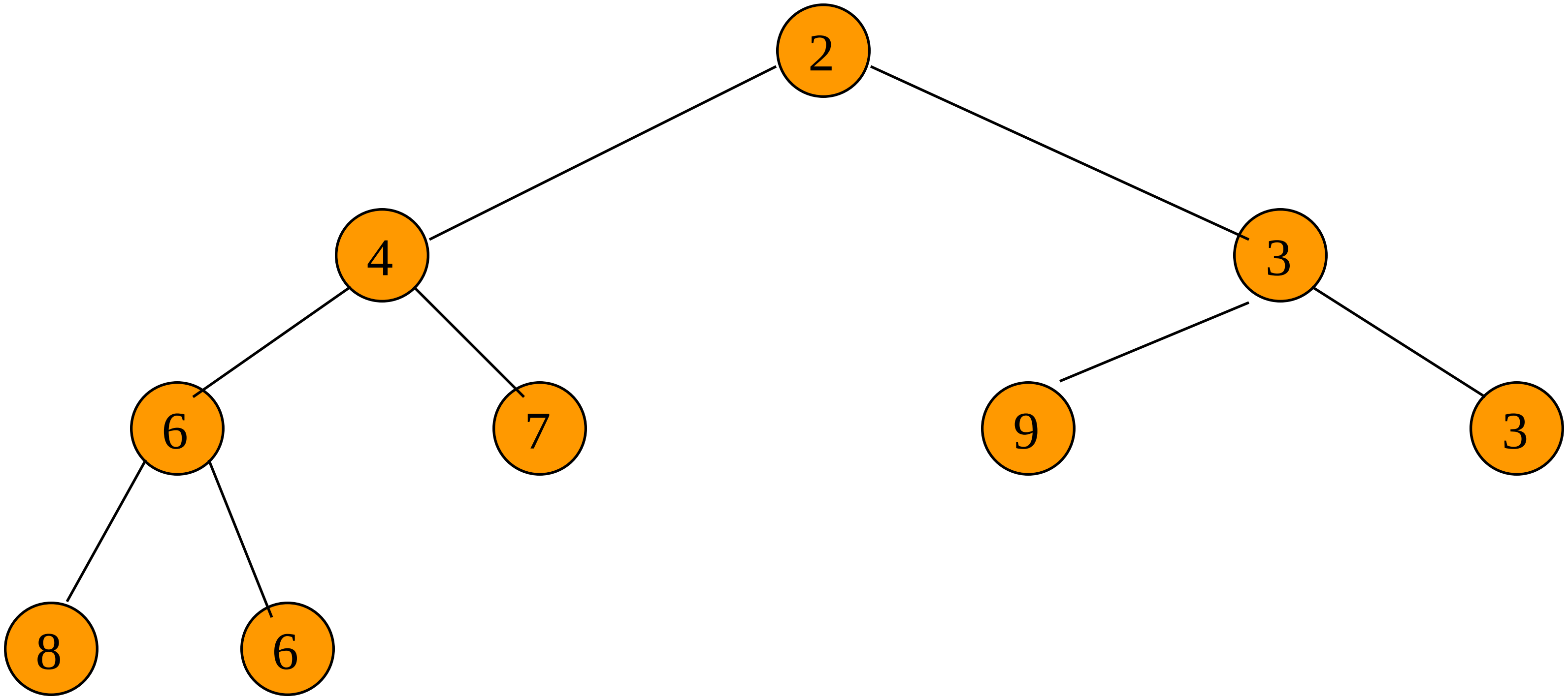
A *min heap* is a complete binary tree that is also a min tree.

Min Heap With 9 Nodes



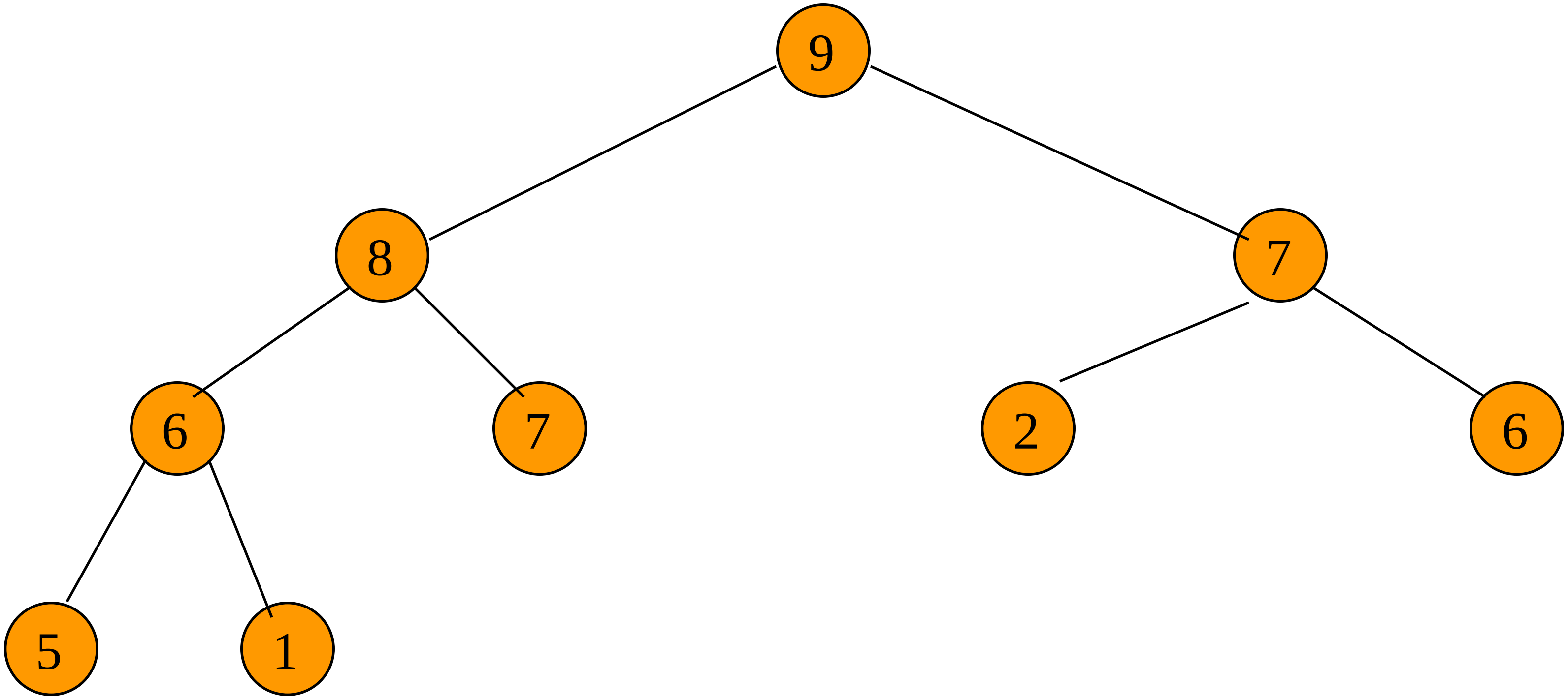
Complete binary tree with 9 nodes.

Min Heap With 9 Nodes



Complete binary tree with 9 nodes that is also a min tree.

Max Heap With 9 Nodes

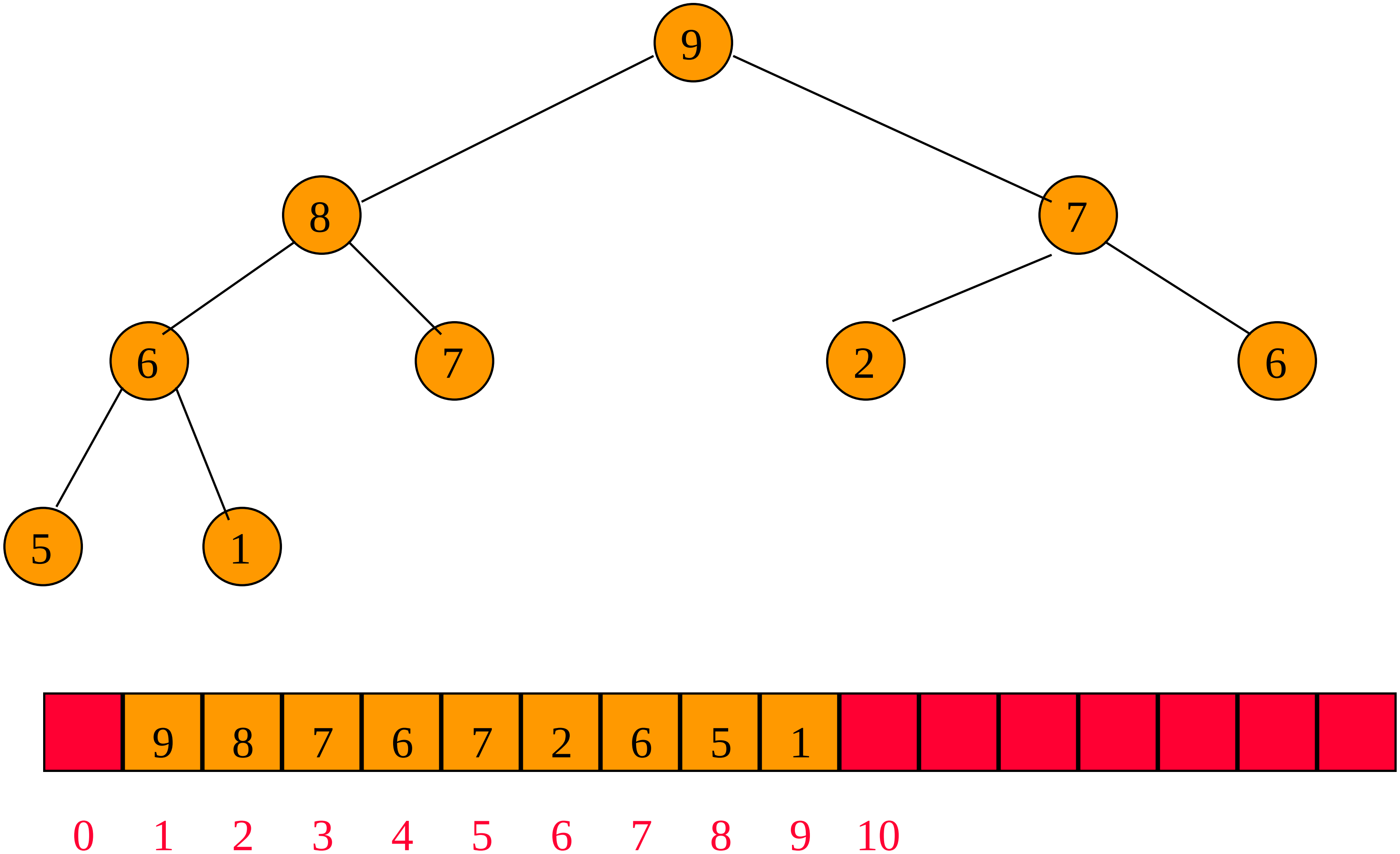


Complete binary tree with 9 nodes that is also a max tree.

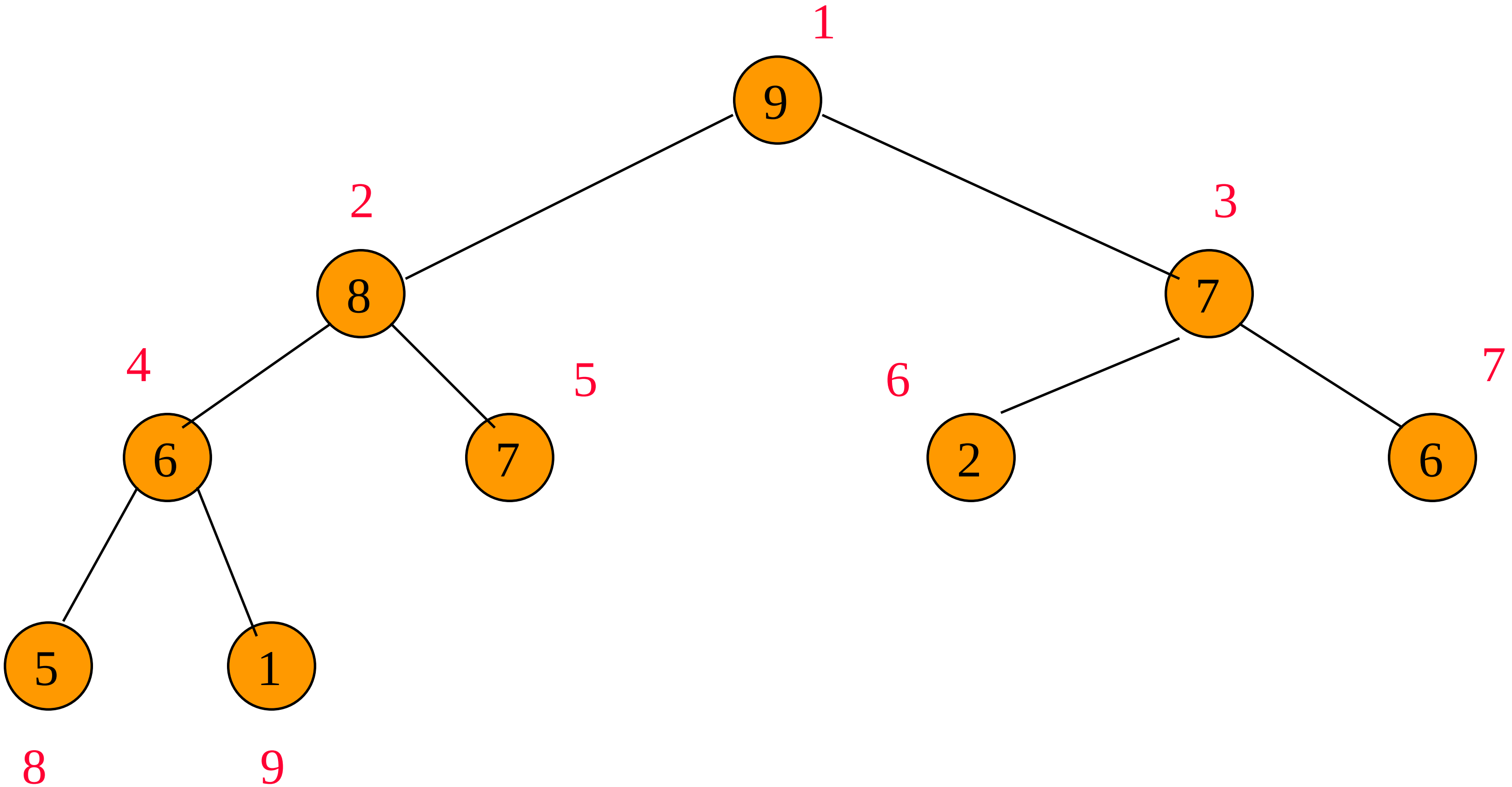
Heap Height

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

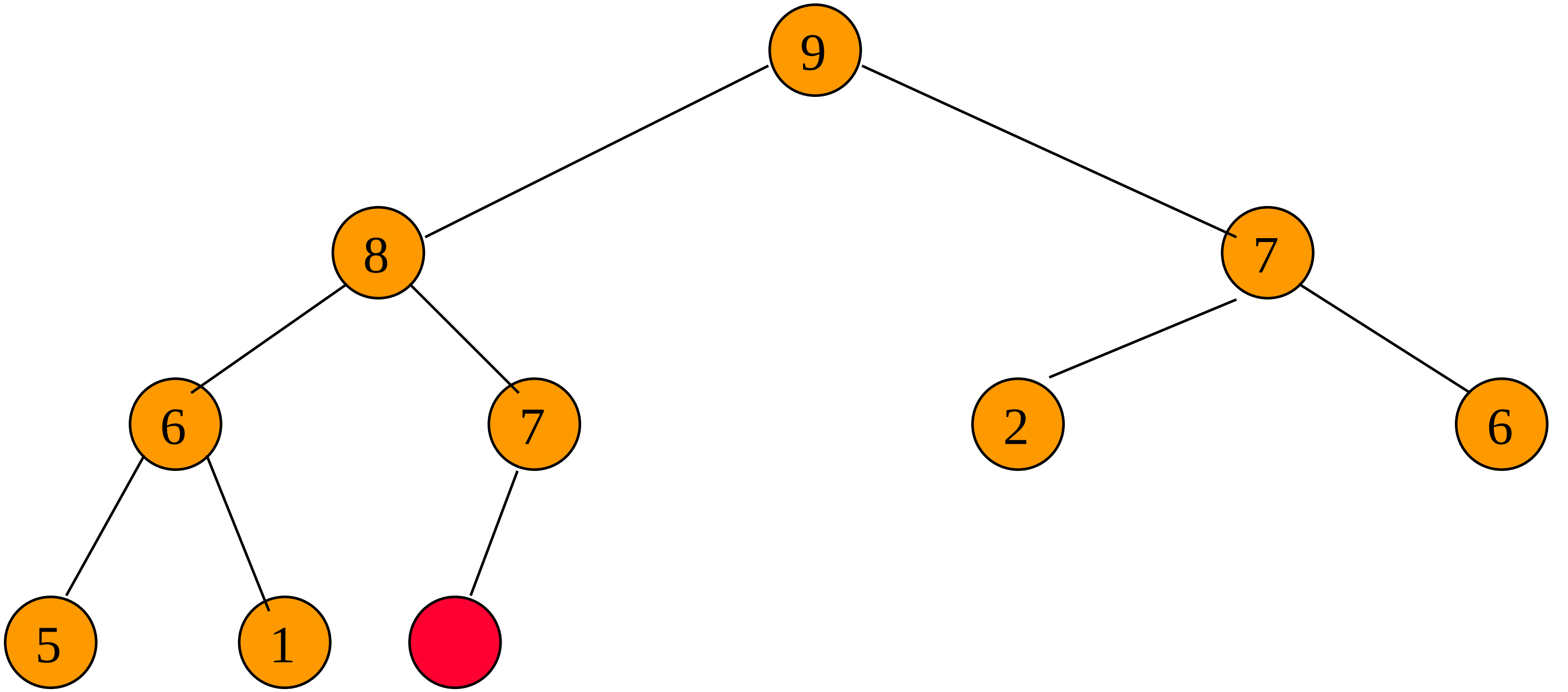
A Heap Is Efficiently Represented As An Array



Moving Up And Down A Heap

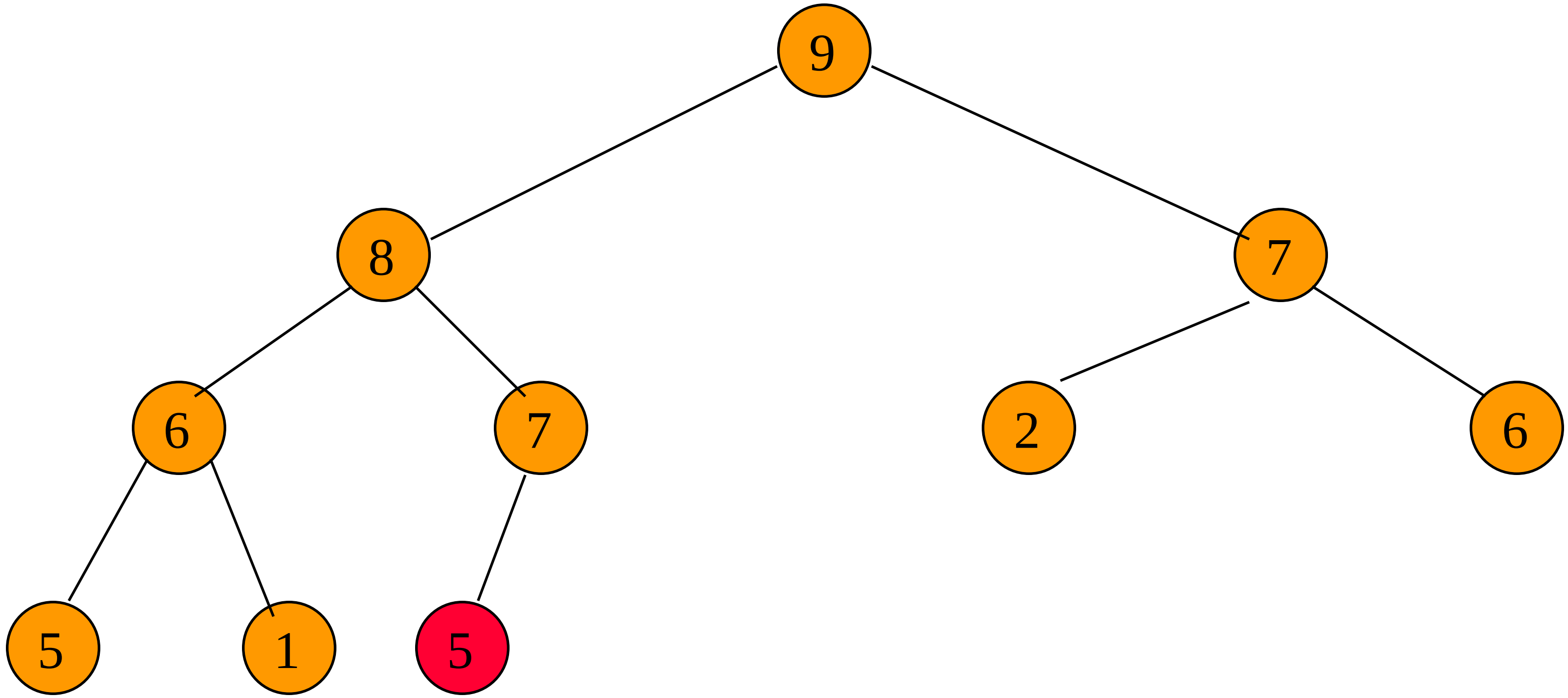


Inserting An Element Into A Max Heap



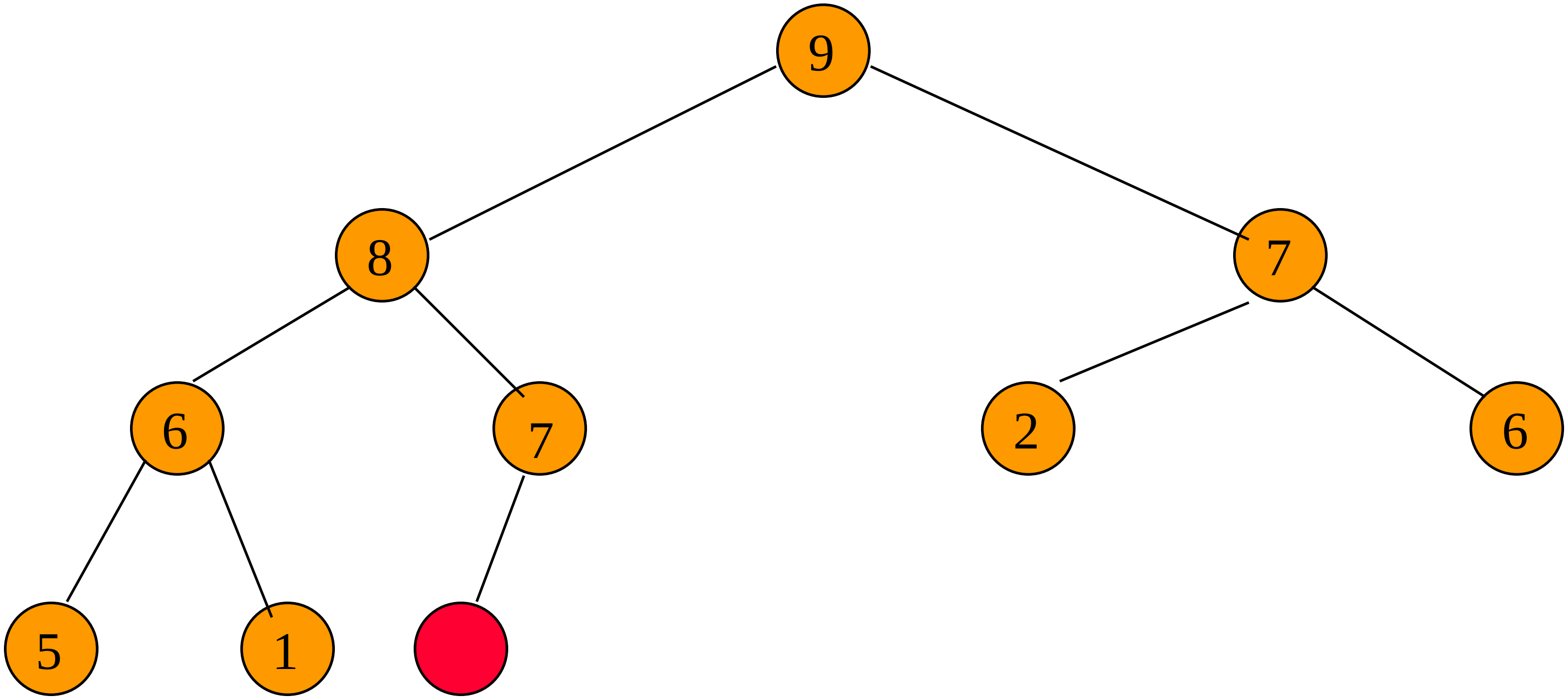
Complete binary tree with 10 nodes.

Inserting An Element Into A Max Heap



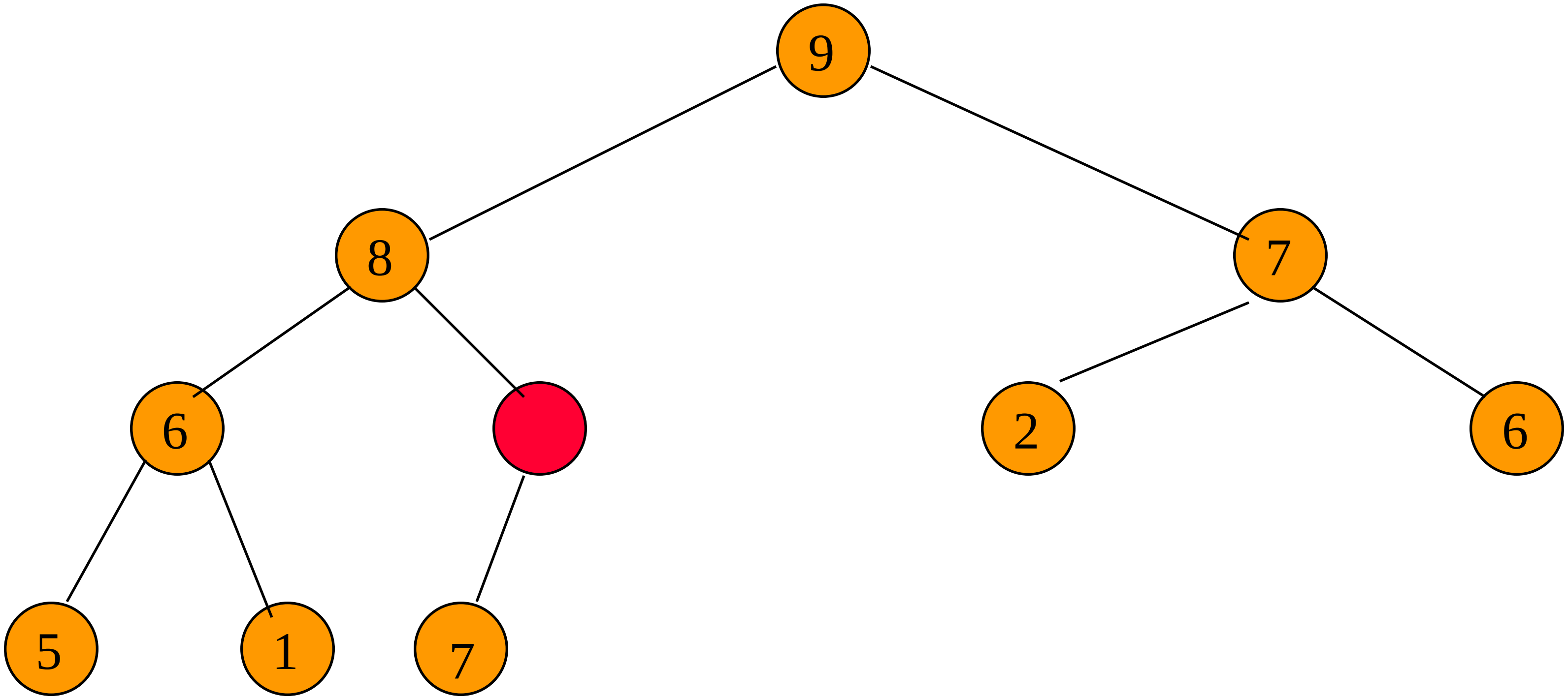
New element is 5.

Inserting An Element Into A Max Heap



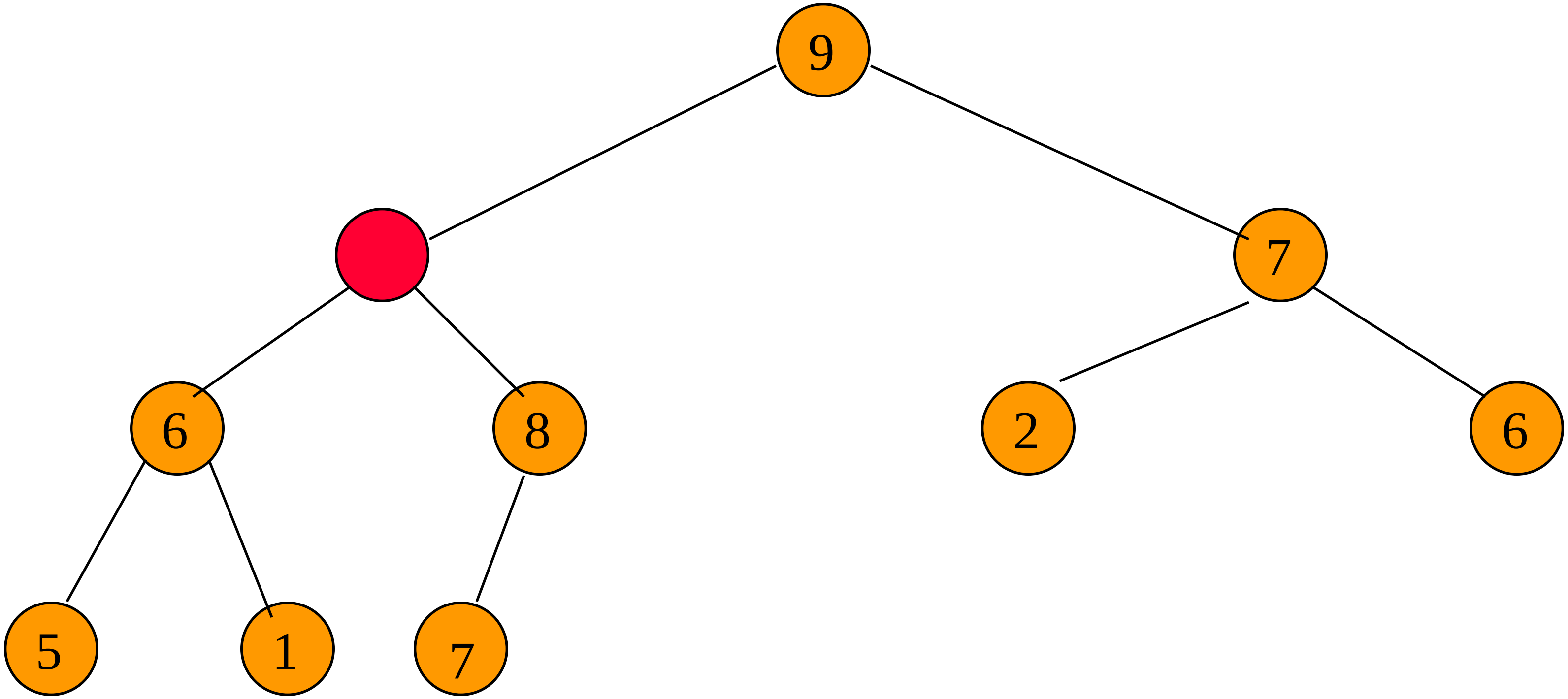
New element is 20.

Inserting An Element Into A Max Heap



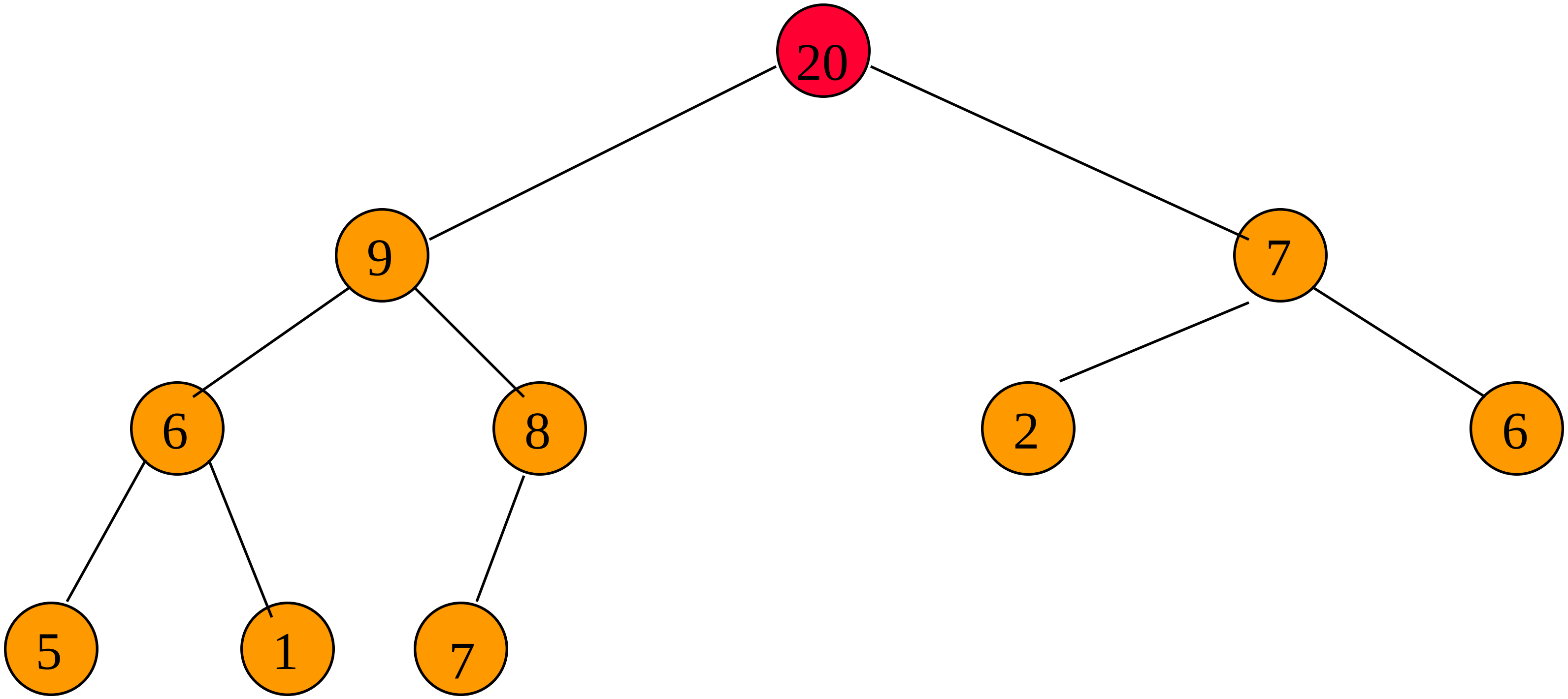
New element is 20.

Inserting An Element Into A Max Heap



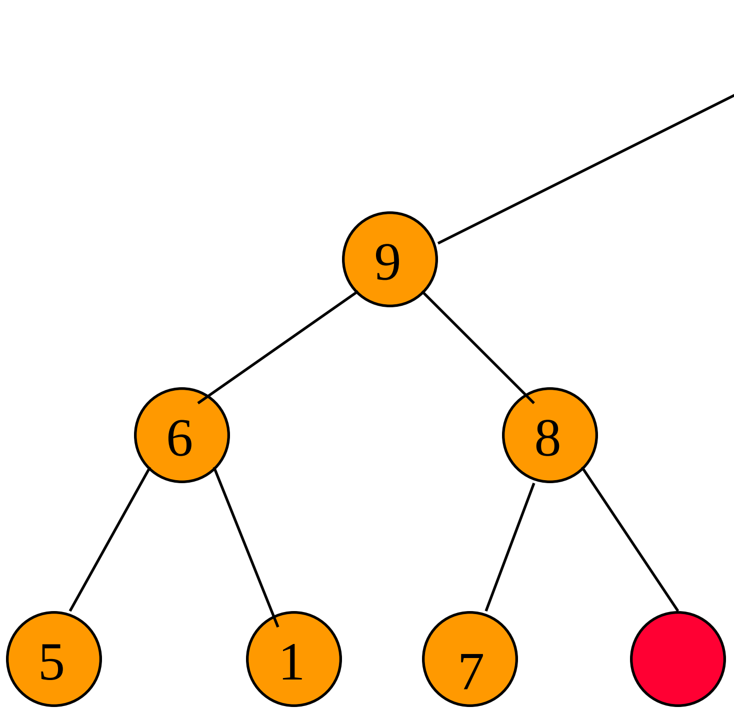
New element is 20.

Inserting An Element Into A Max Heap



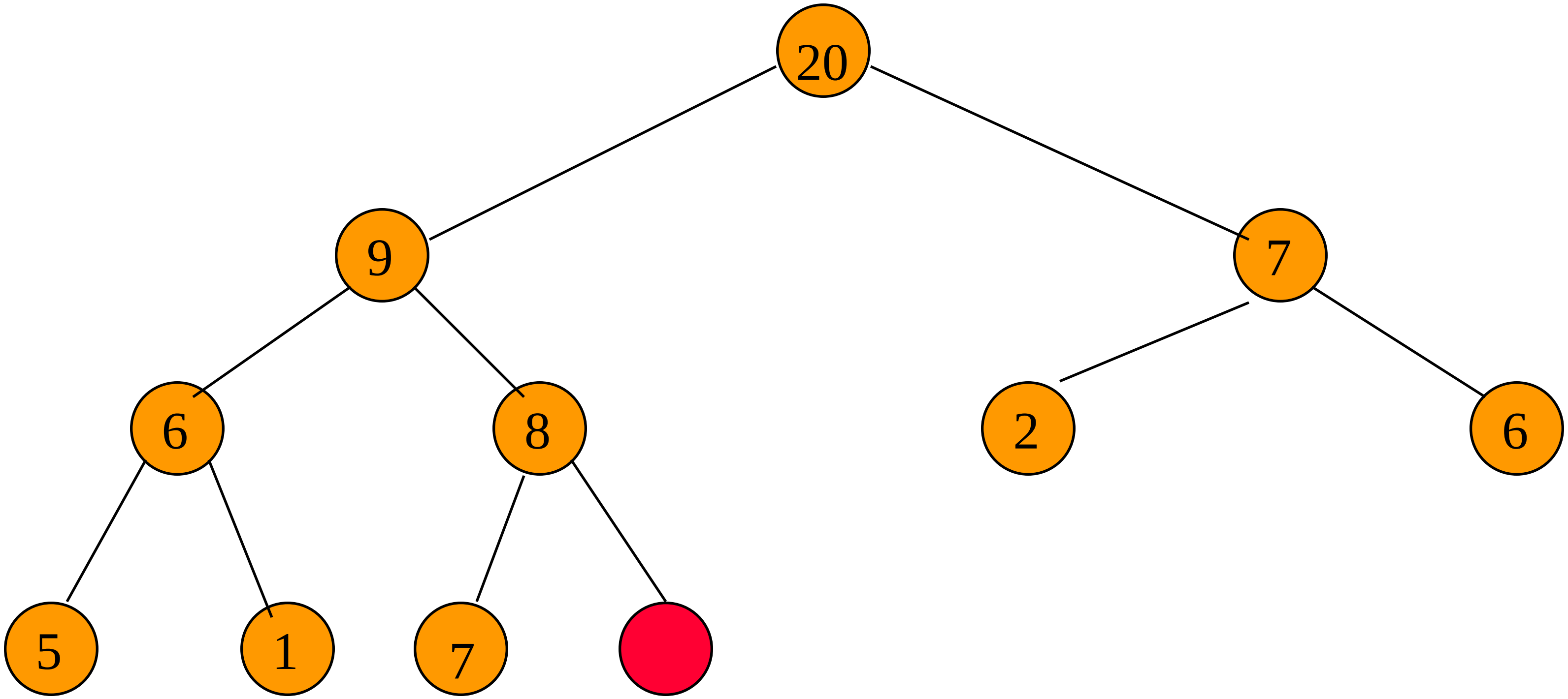
New element is 20.

Inserting An Element Into A Max Heap



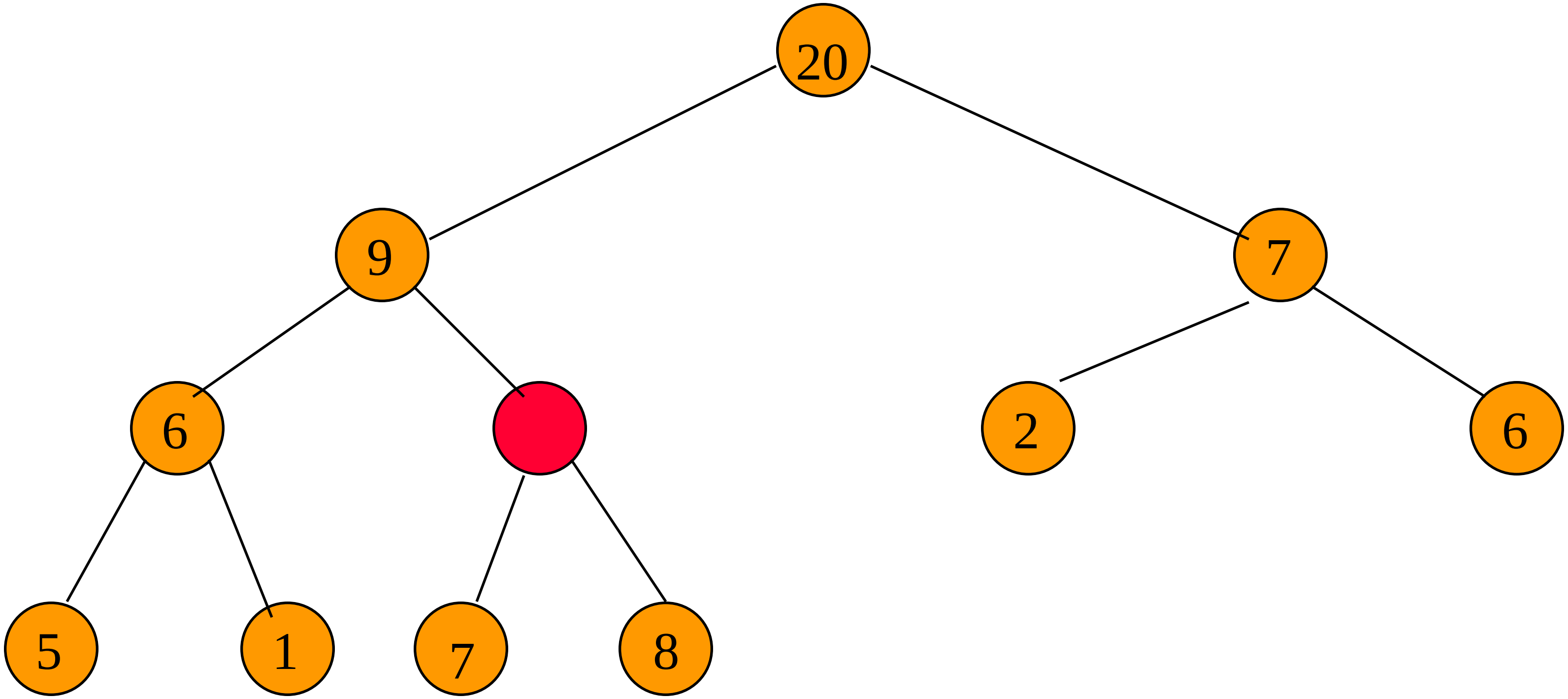
Complete binary tree with 11 nodes.

Inserting An Element Into A Max Heap



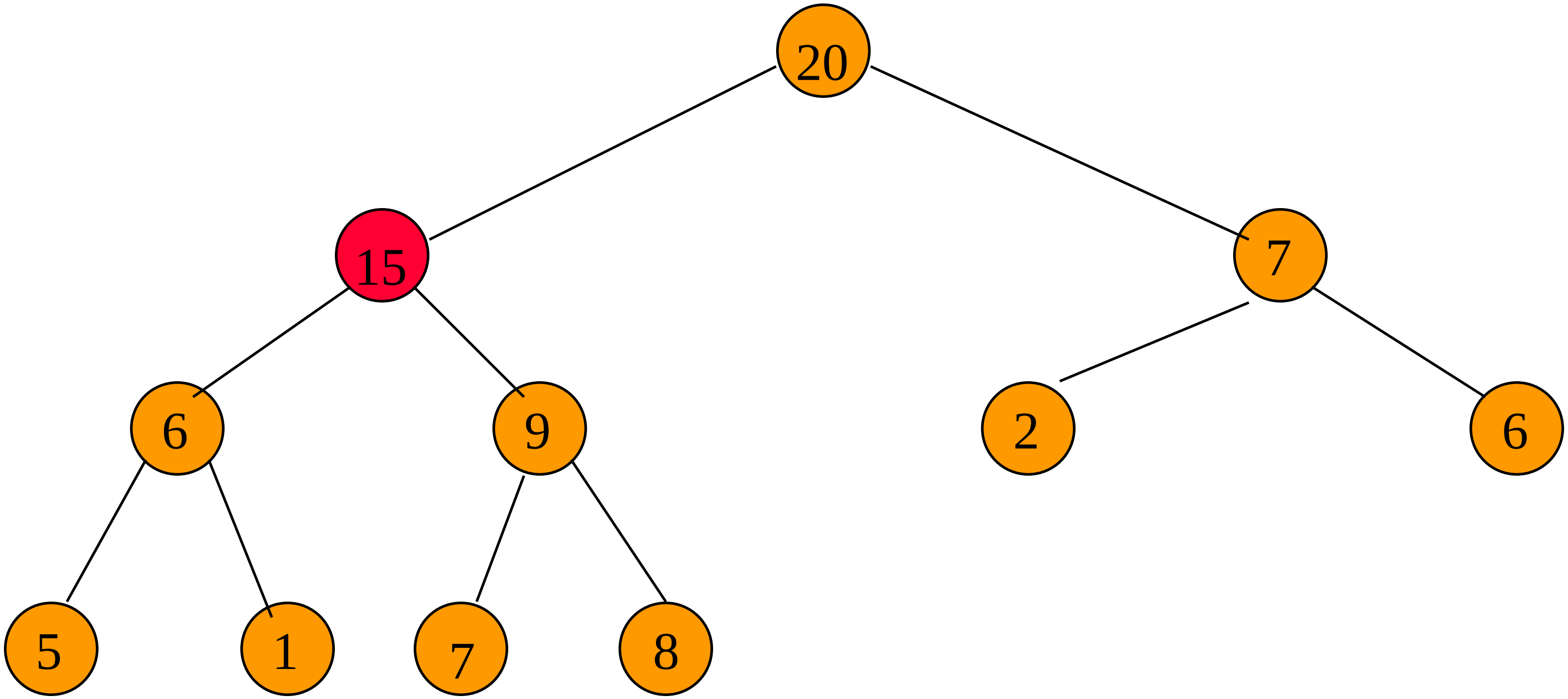
New element is 15.

Inserting An Element Into A Max Heap



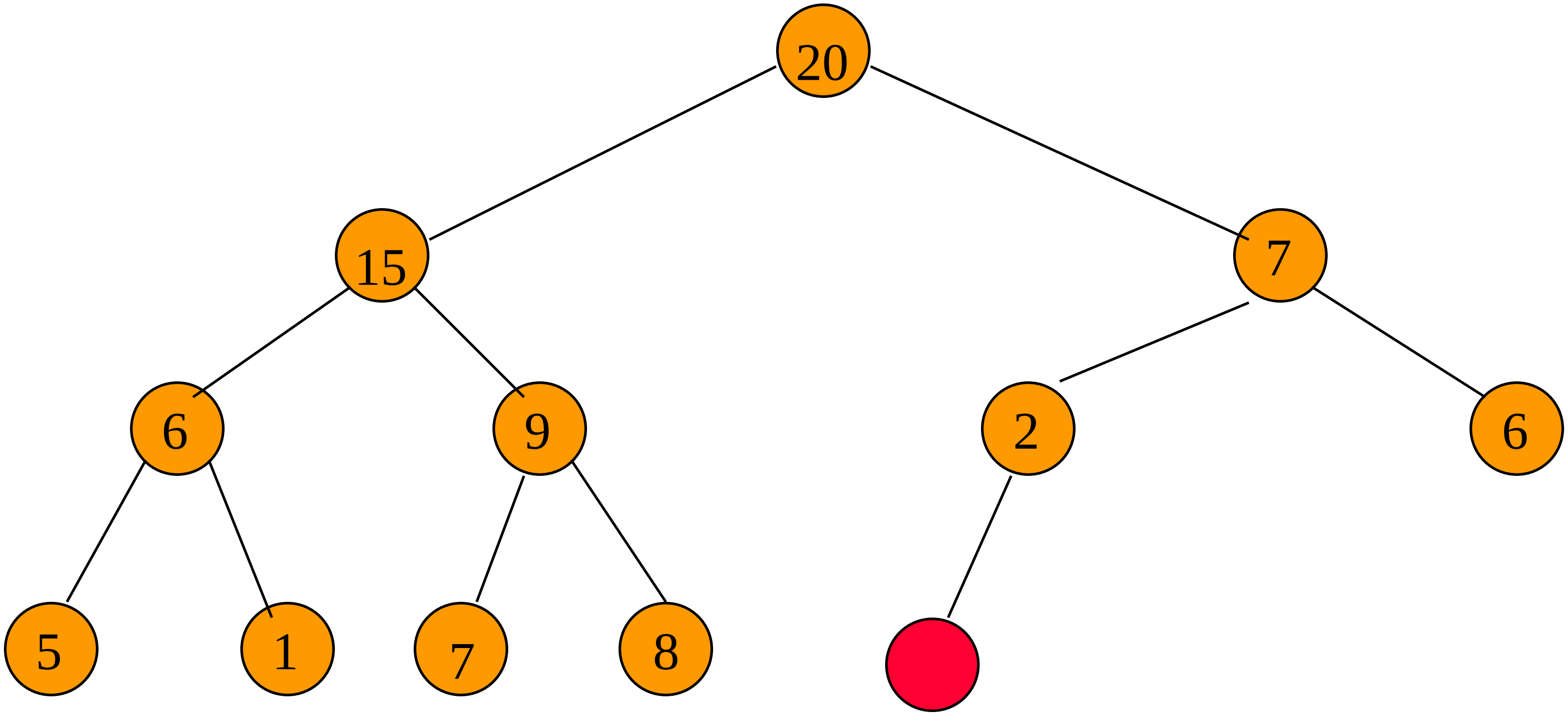
New element is 15.

Inserting An Element Into A Max Heap



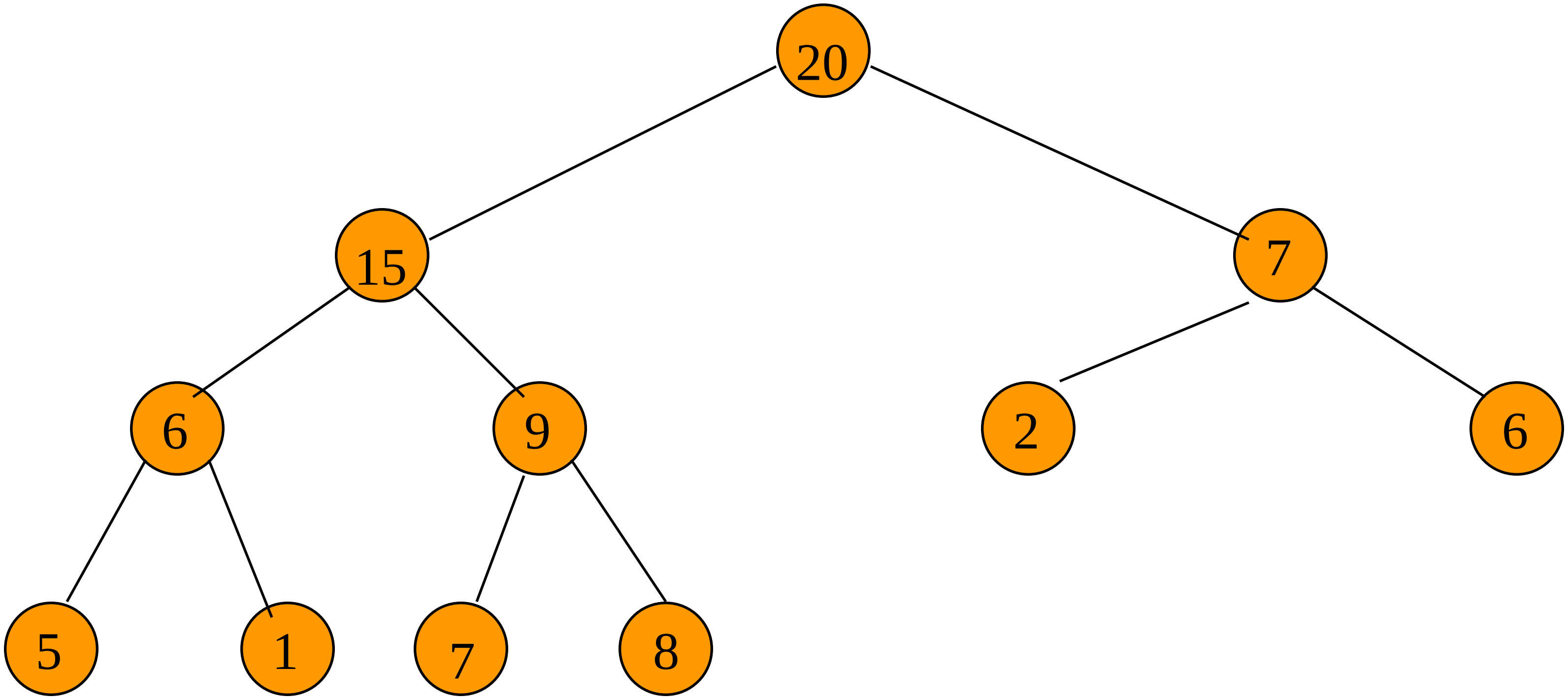
New element is 15.

Complexity Of Insert



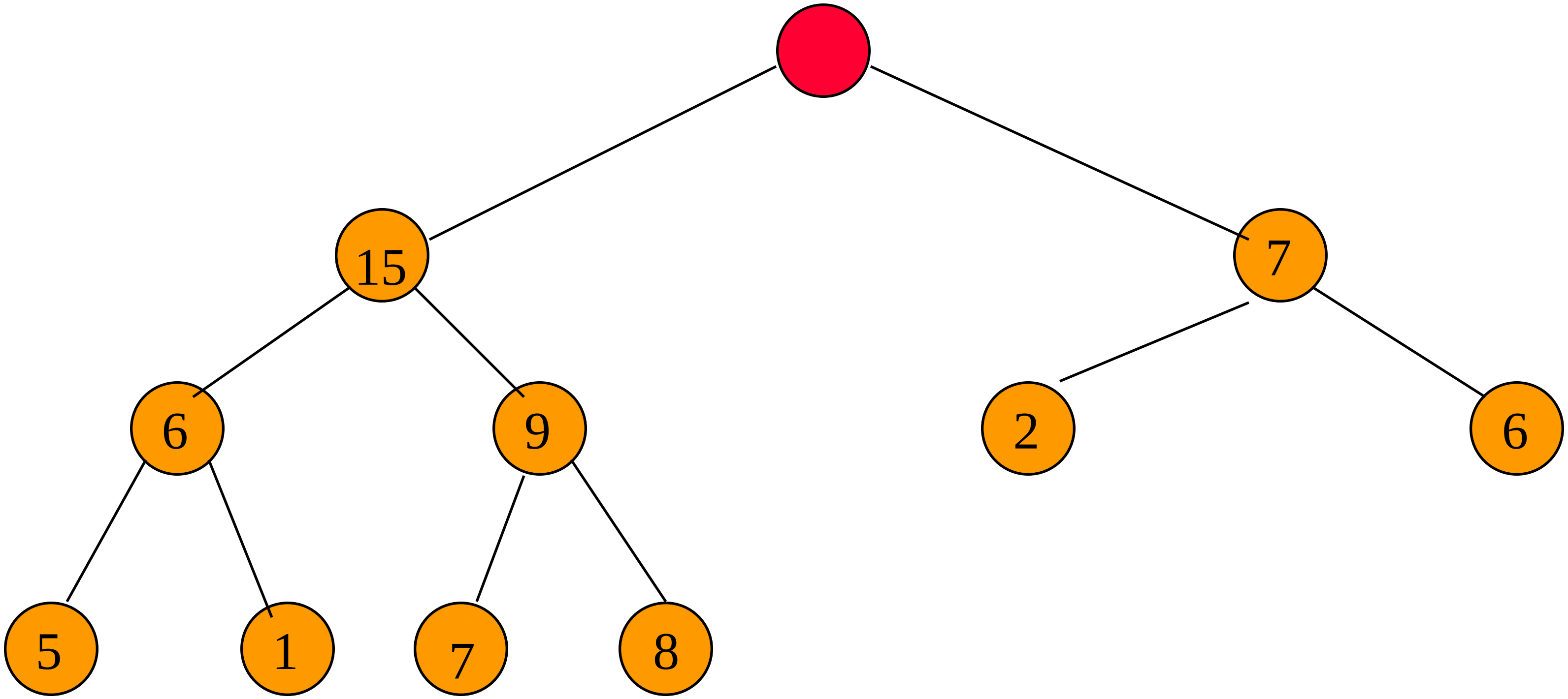
Complexity is O(log n), where n is heap size.

Removing The Max Element



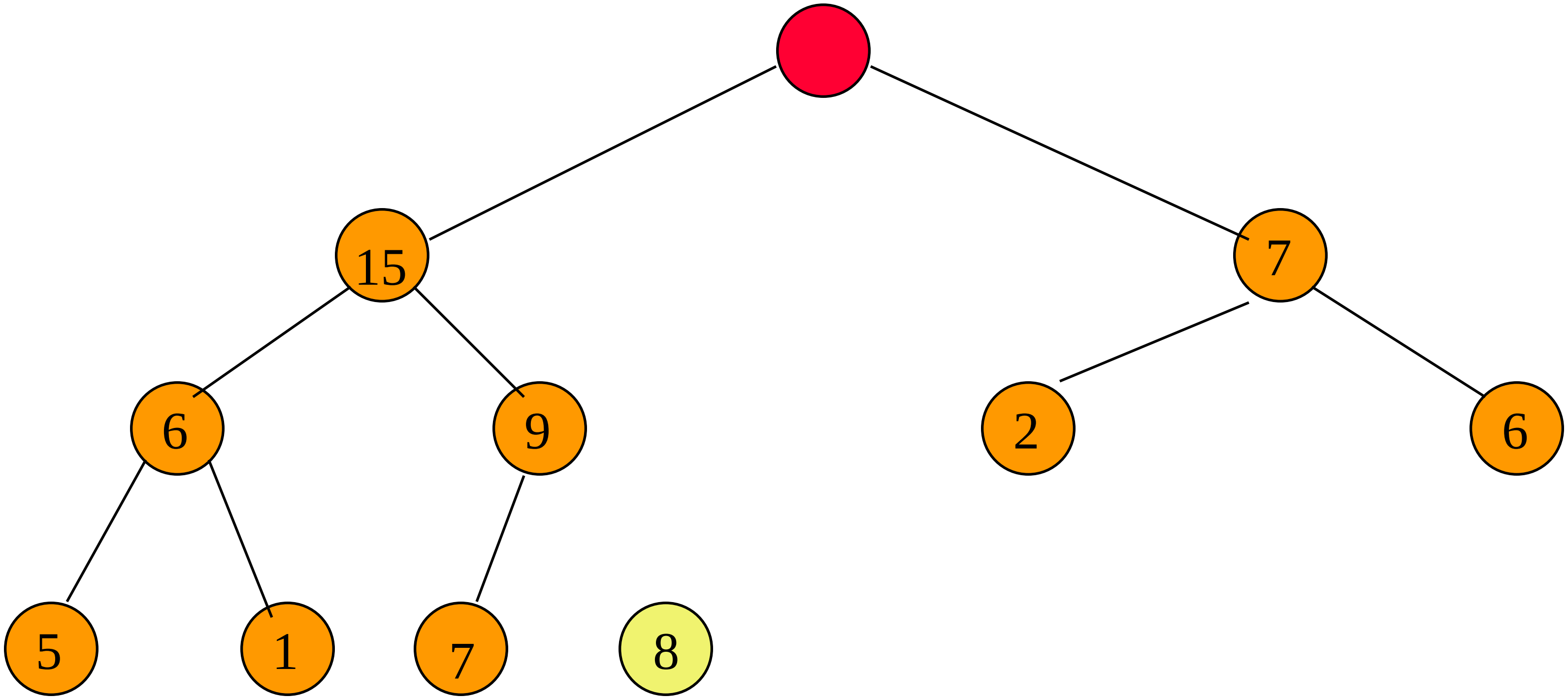
Max element is in the root.

Removing The Max Element



After max element is removed.

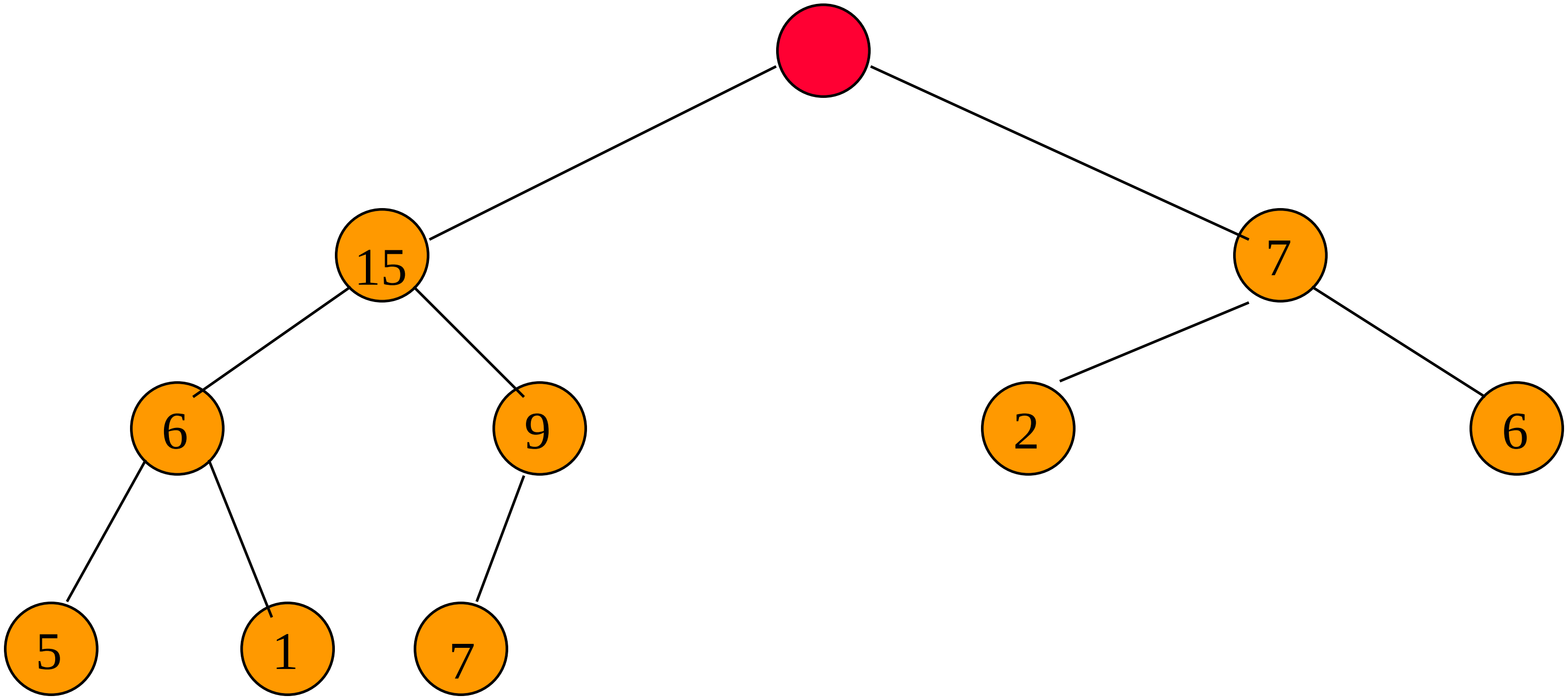
Removing The Max Element



Heap with 10 nodes.

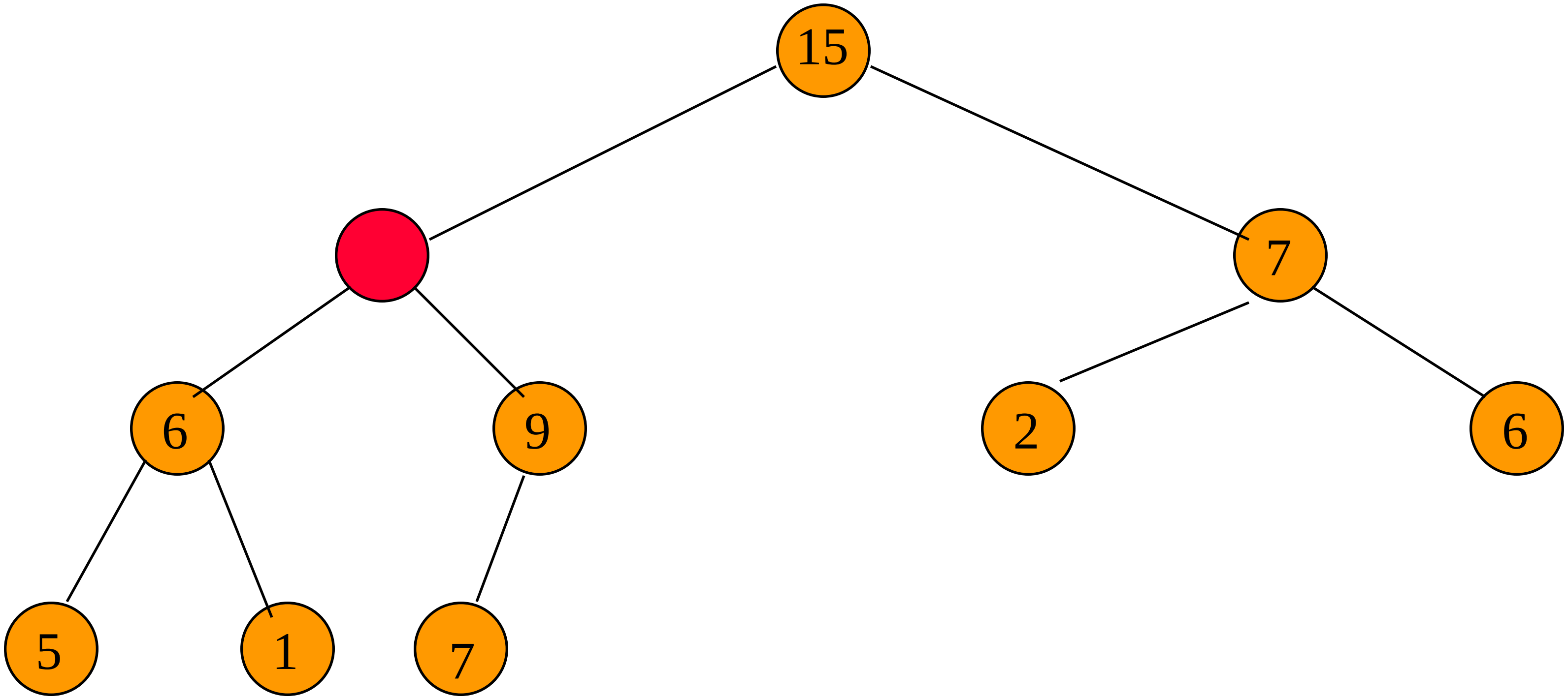
Reinsert 8 into the heap.

Removing The Max Element



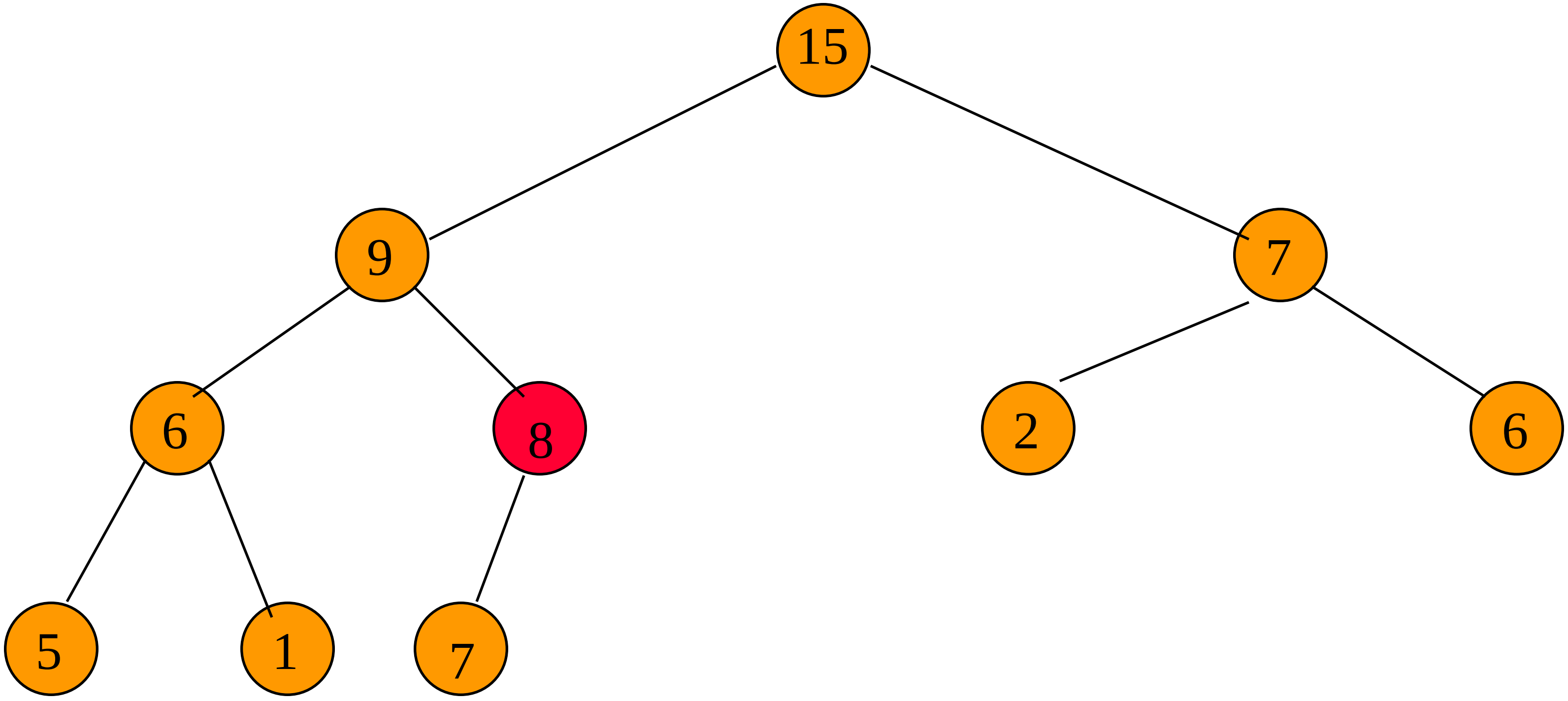
Reinsert 8 into the heap.

Removing The Max Element



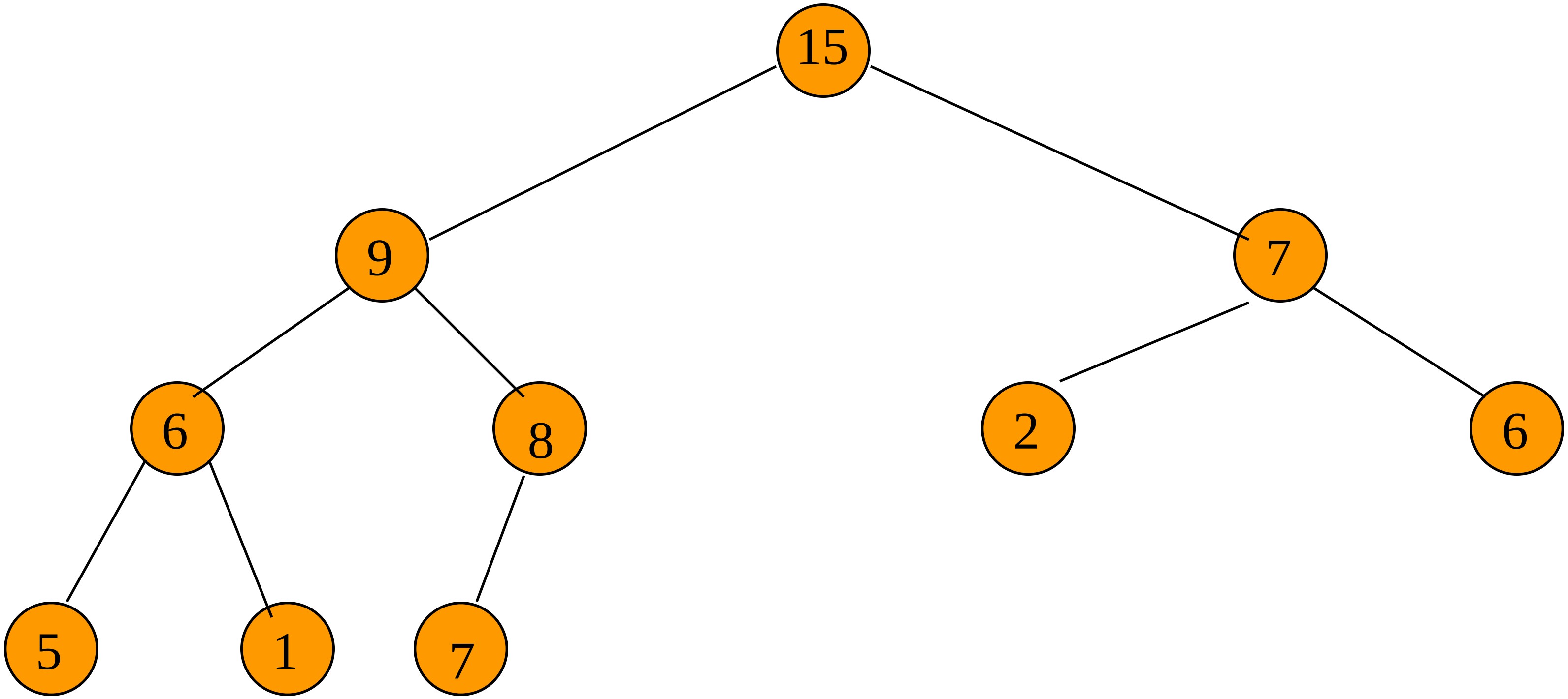
Reinsert 8 into the heap.

Removing The Max Element



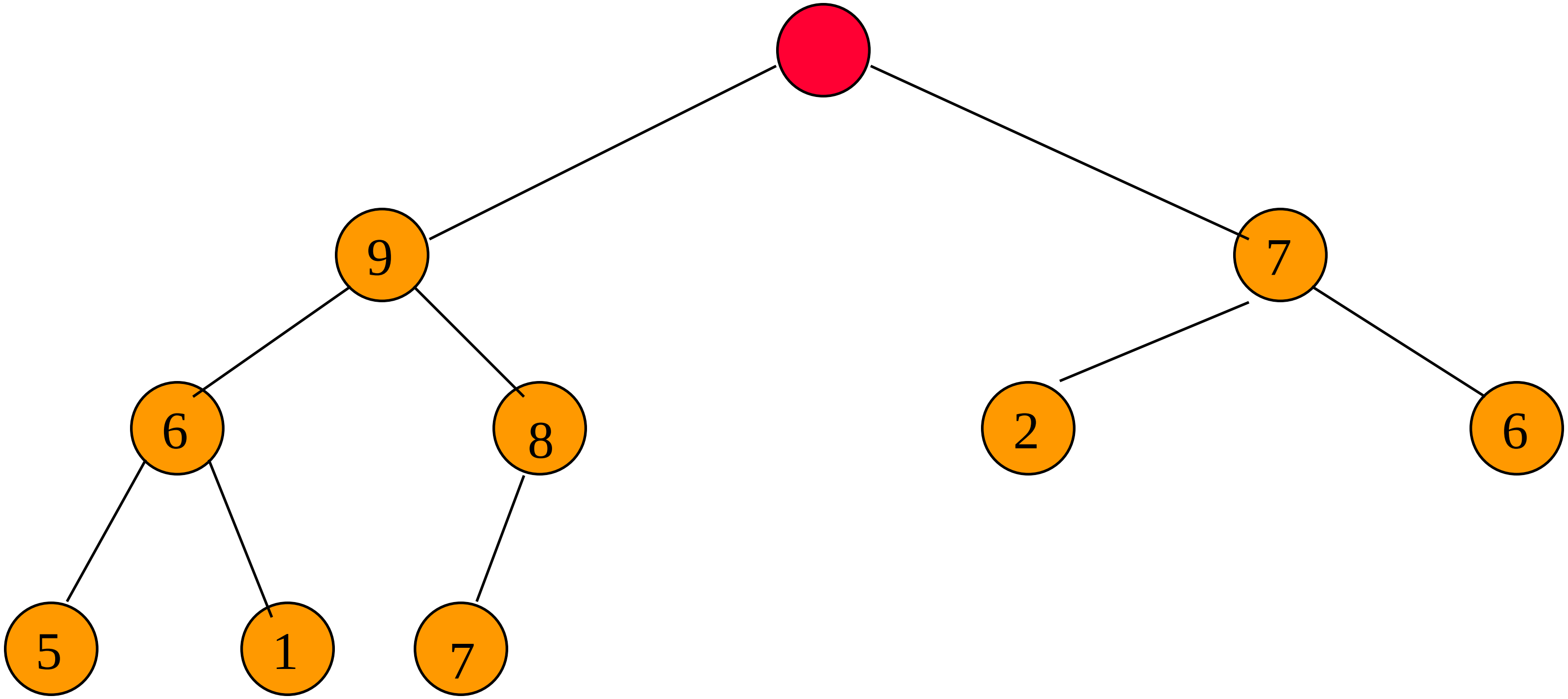
Reinsert 8 into the heap.

Removing The Max Element



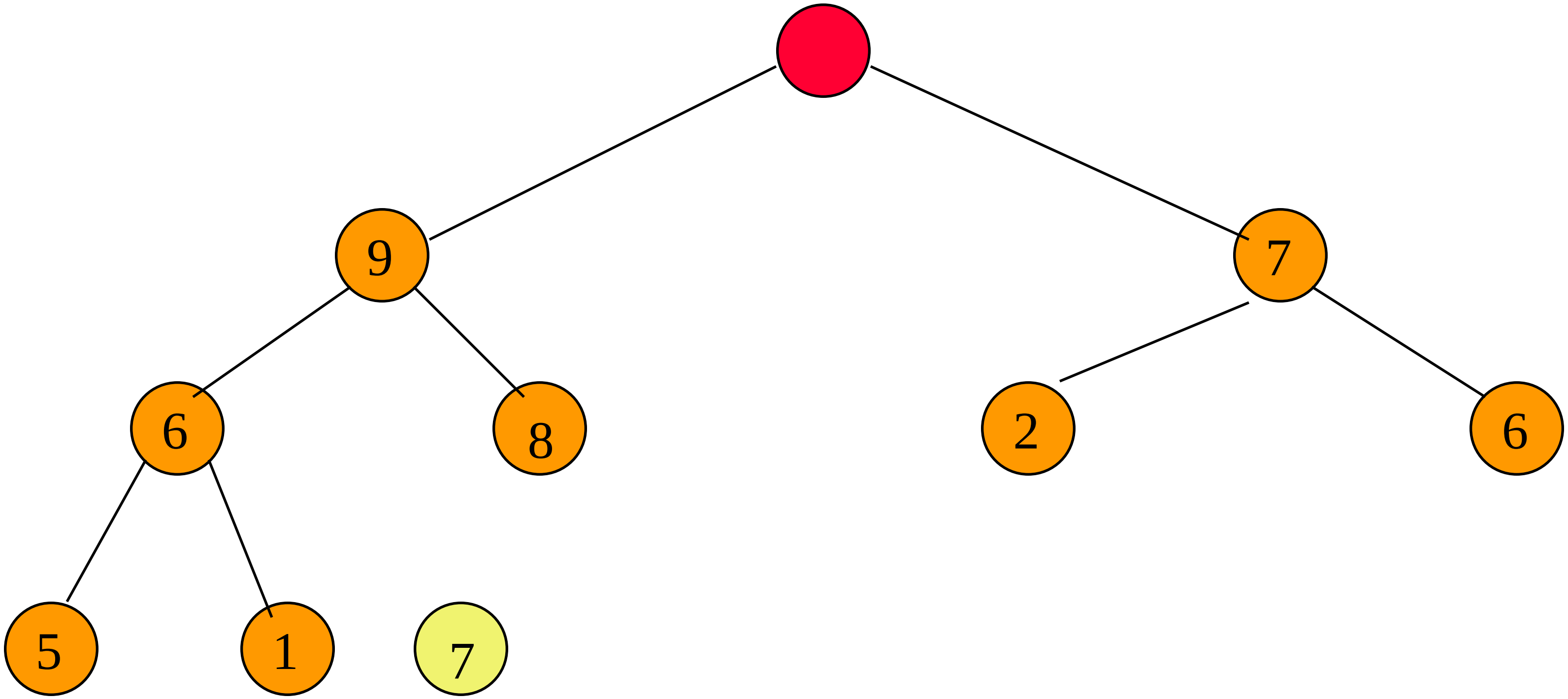
Max element is 15.

Removing The Max Element



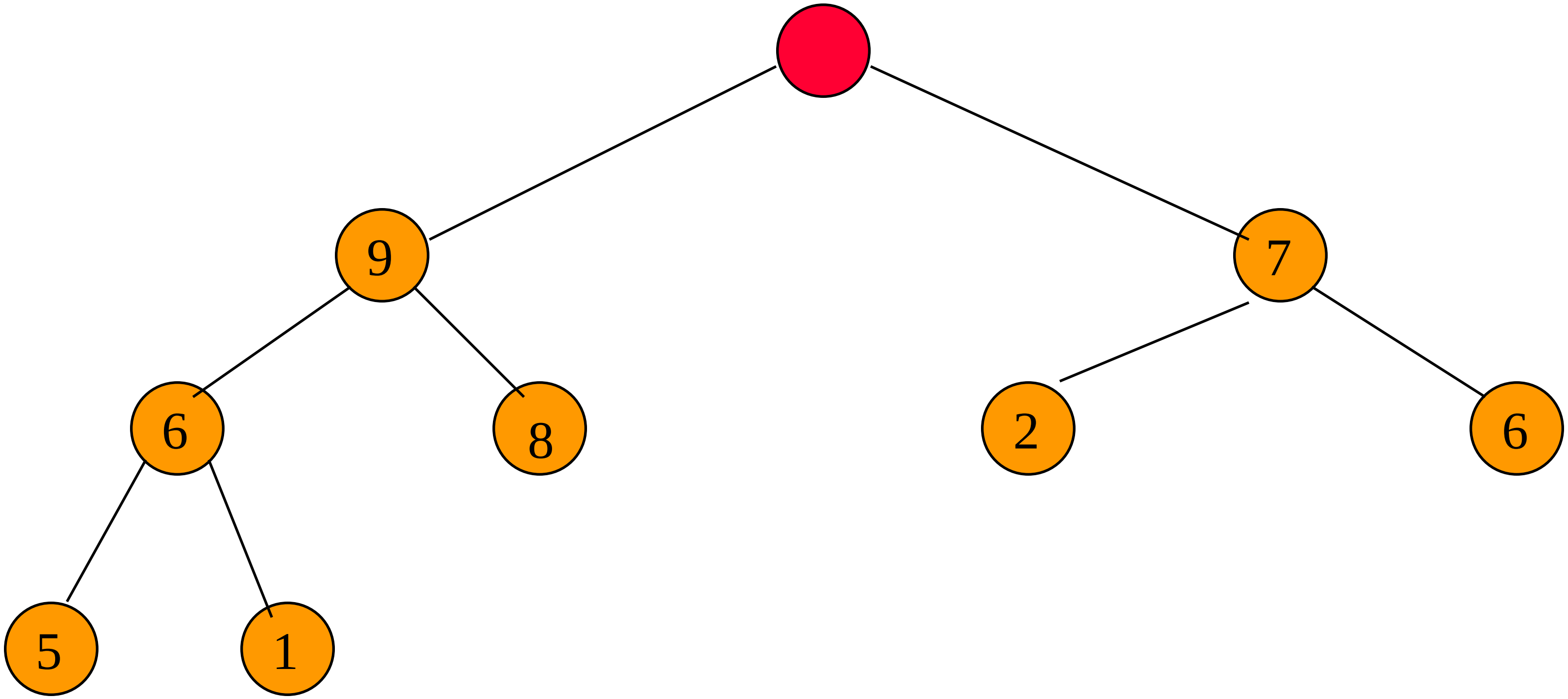
After max element is removed.

Removing The Max Element



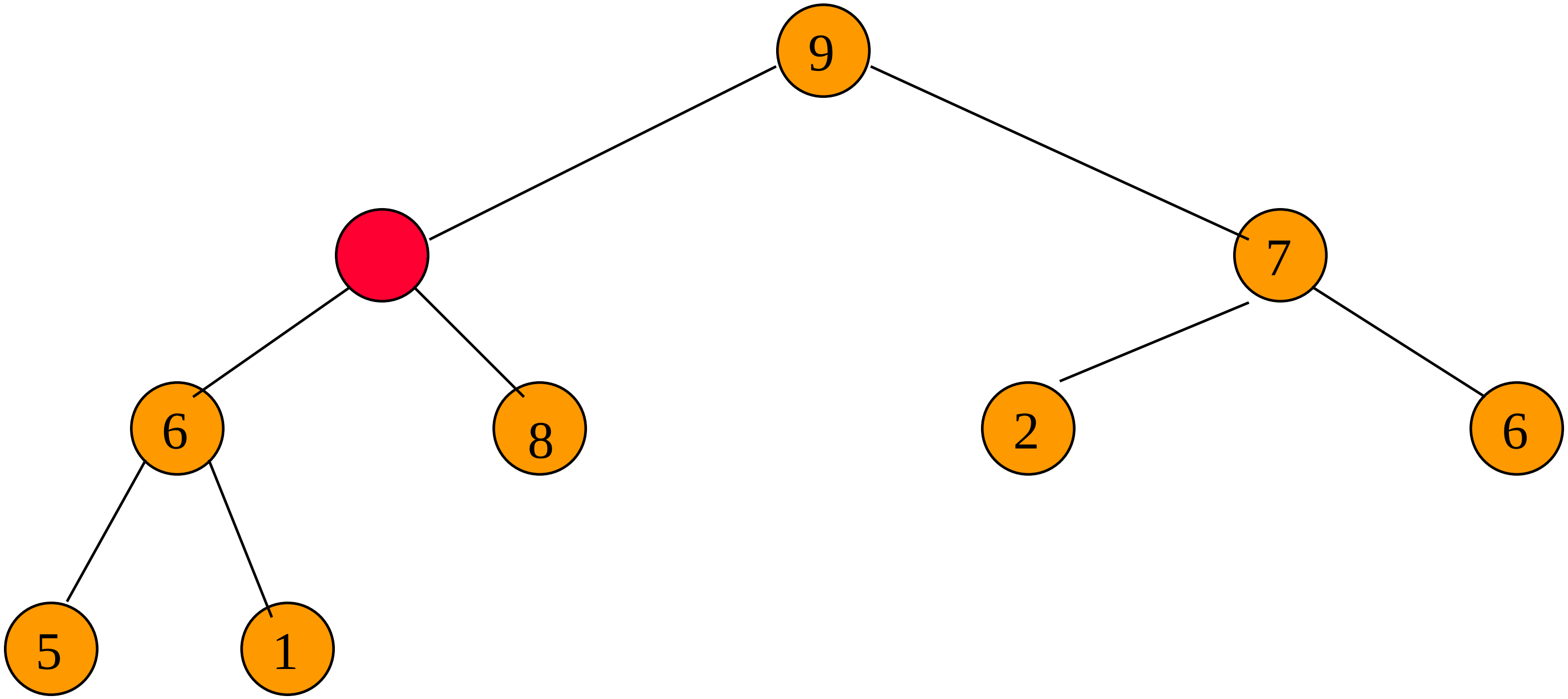
Heap with 9 nodes.

Removing The Max Element



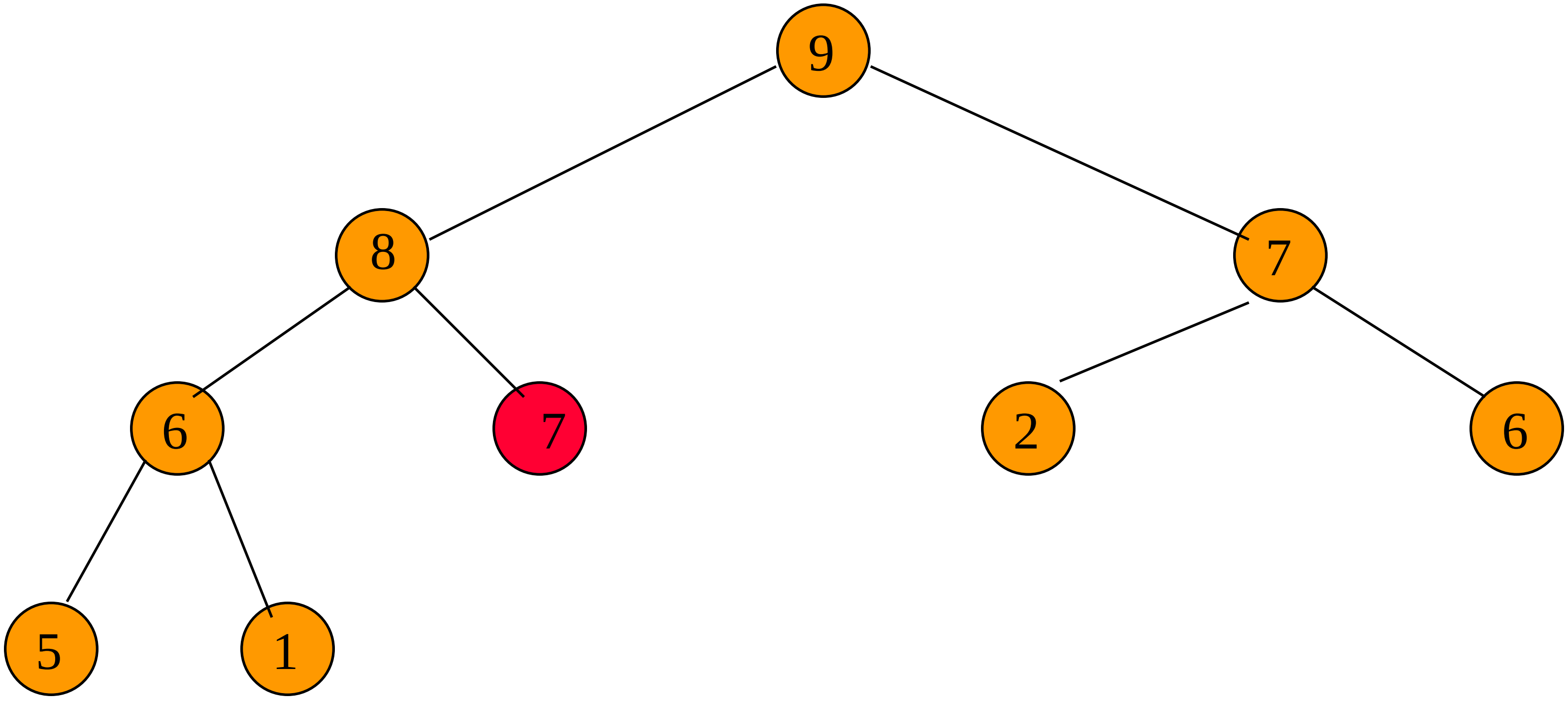
Reinsert 7.

Removing The Max Element



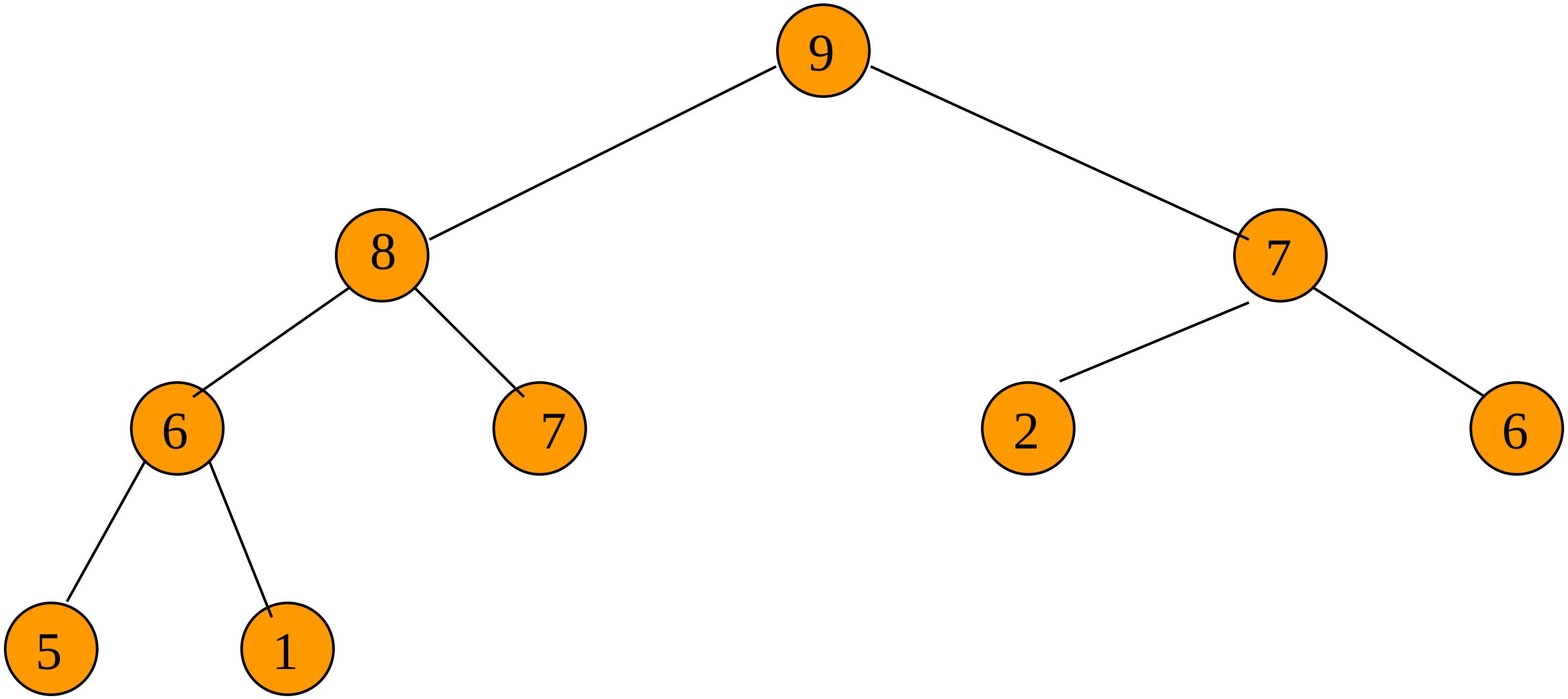
Reinsert 7.

Removing The Max Element



Reinsert 7.

Complexity Of Remove Max Element



Complexity is O(log n).

Construction, Insertion and Deletion of heap

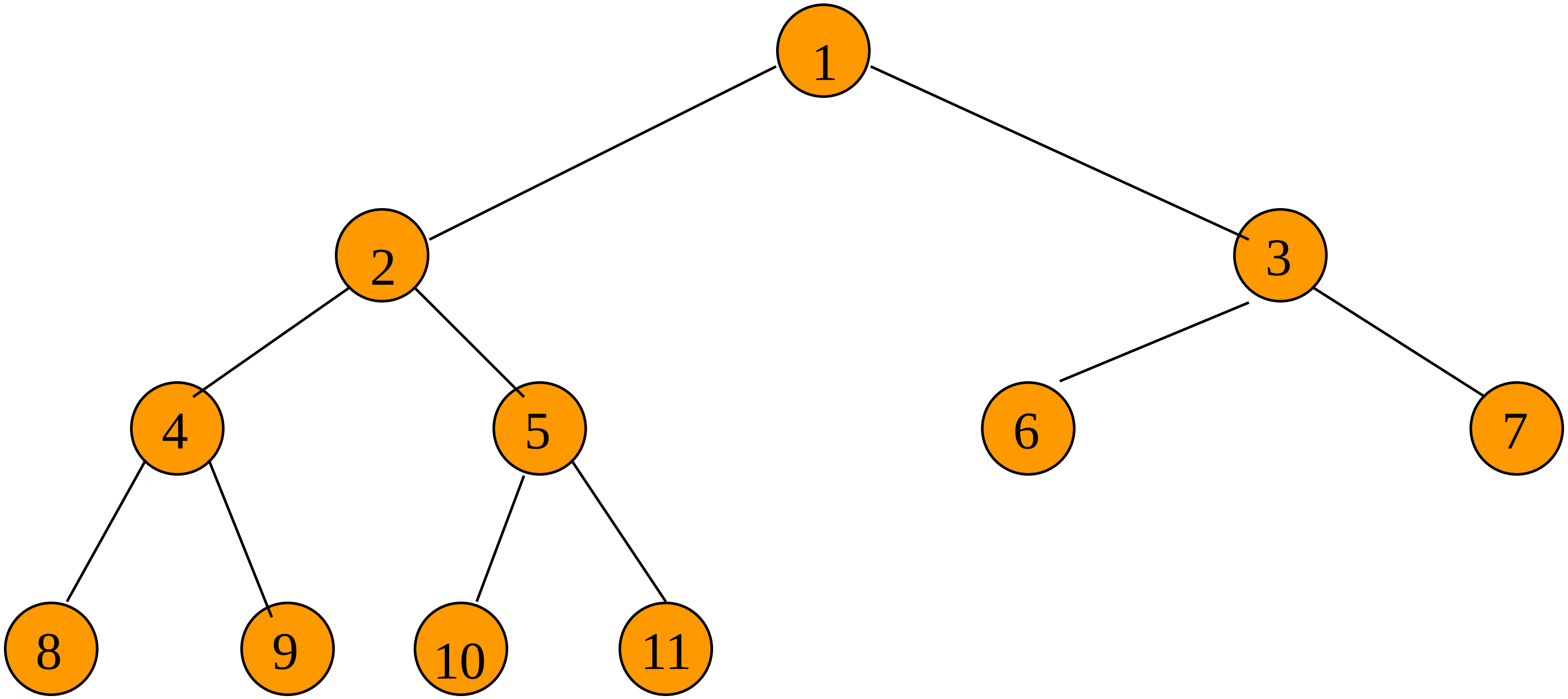
- See animation of construction of heap
- See animation of insertion of heap
- See animation of deletion of heap

Initializing A Max Heap

```
BUILD-MAX-HEAP(A)
A.heap-size = A.length
for i = ⌊A.length/2⌋ downto 1
    MAX-HEAPIFY(A, i)
```

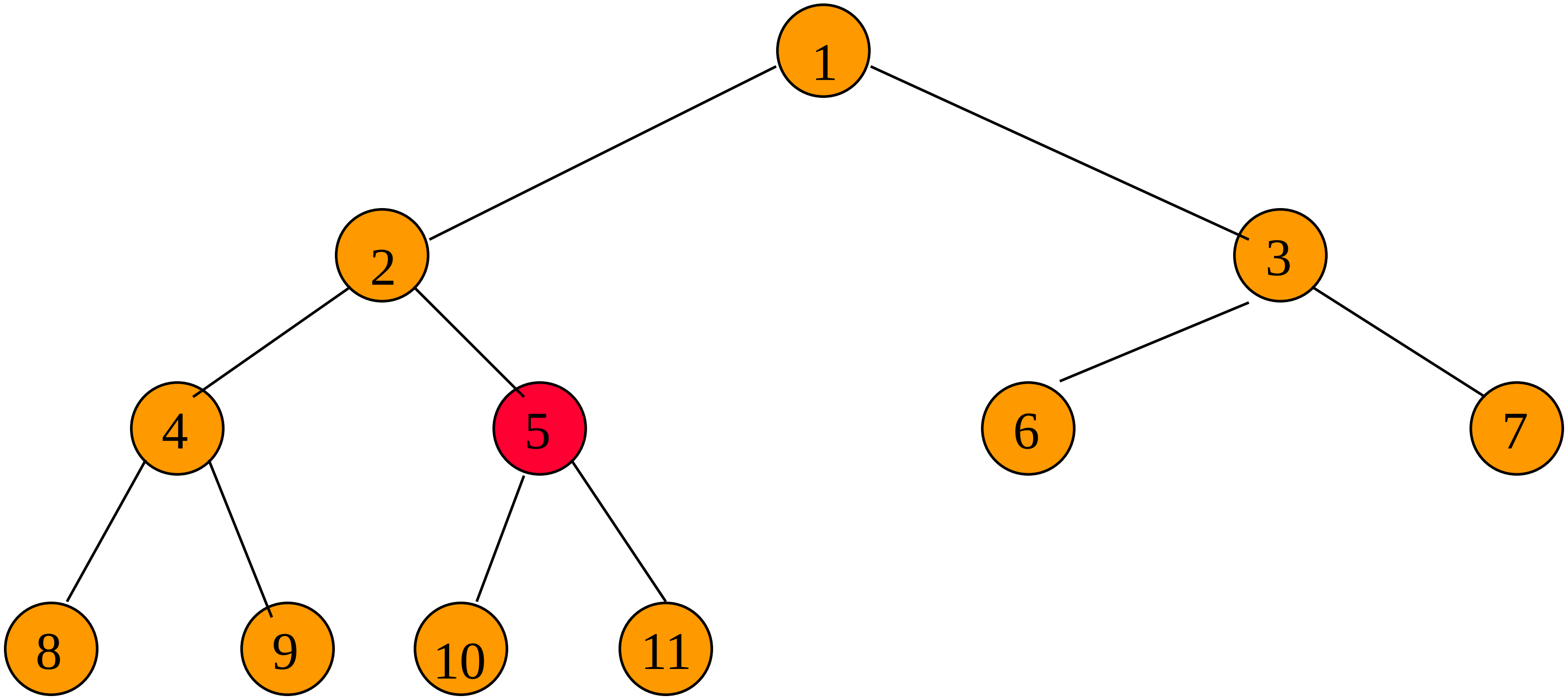
```
MAX-HEAPIFY(A, i)
l = LEFT(i)
r = RIGHT(i)
if l ≤ A.heap-size and A[l] > A[i]
    largest = l
else largest = i
if r ≤ A.heap-size and A[r] > A[largest]
    largest = r
if largest ≠ i
    exchange A[i] with A[largest]
    MAX-HEAPIFY(A, largest)
```

Initializing A Max Heap



input array = [-, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]

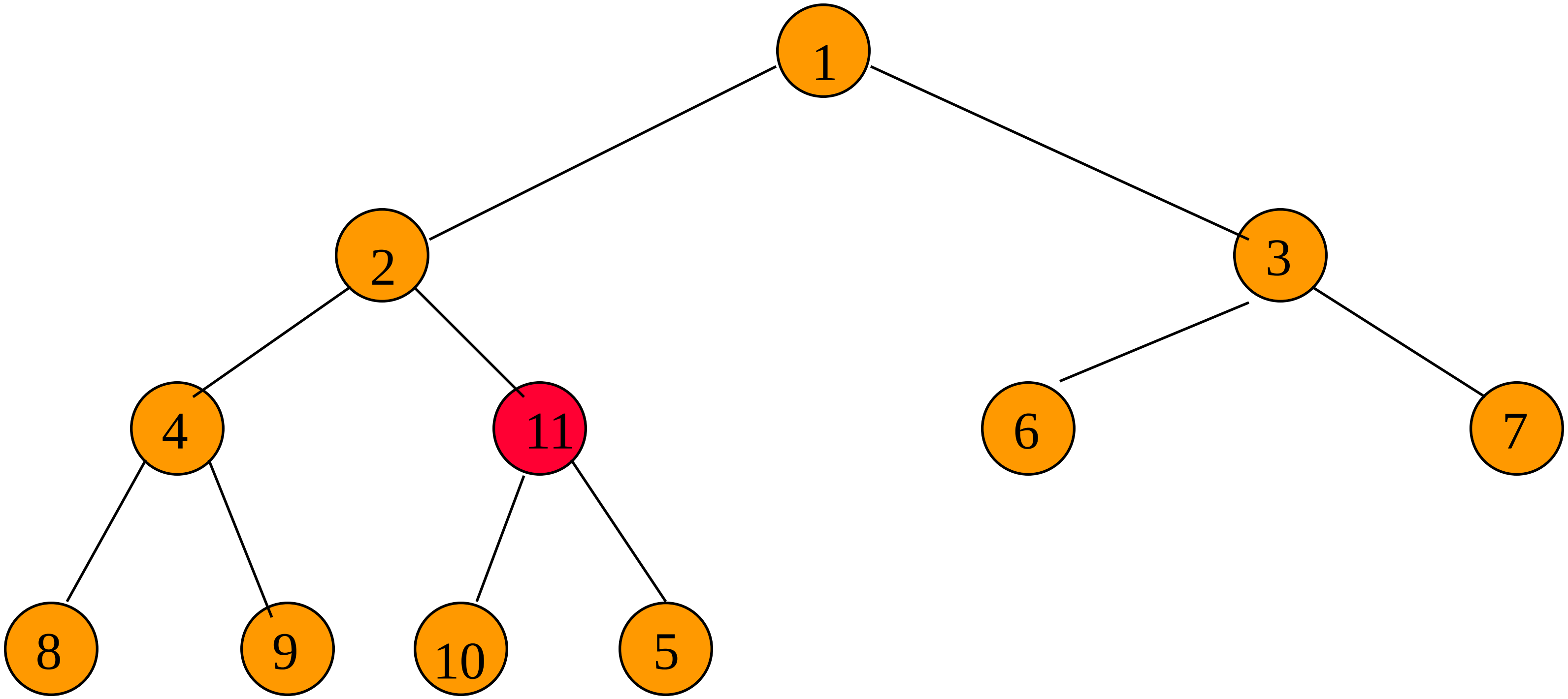
Initializing A Max Heap



Start at rightmost array position that has a child.

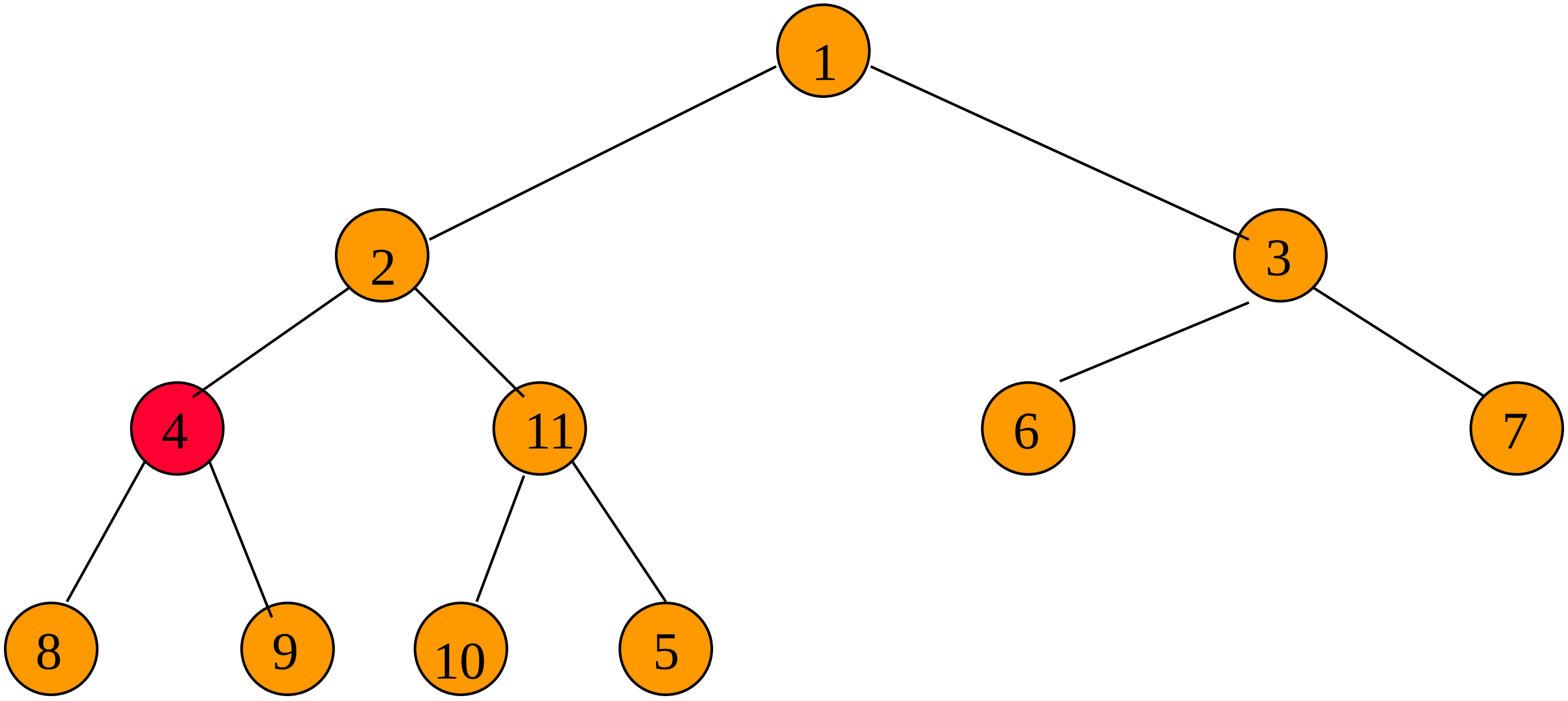
Index is n/2.

Initializing A Max Heap

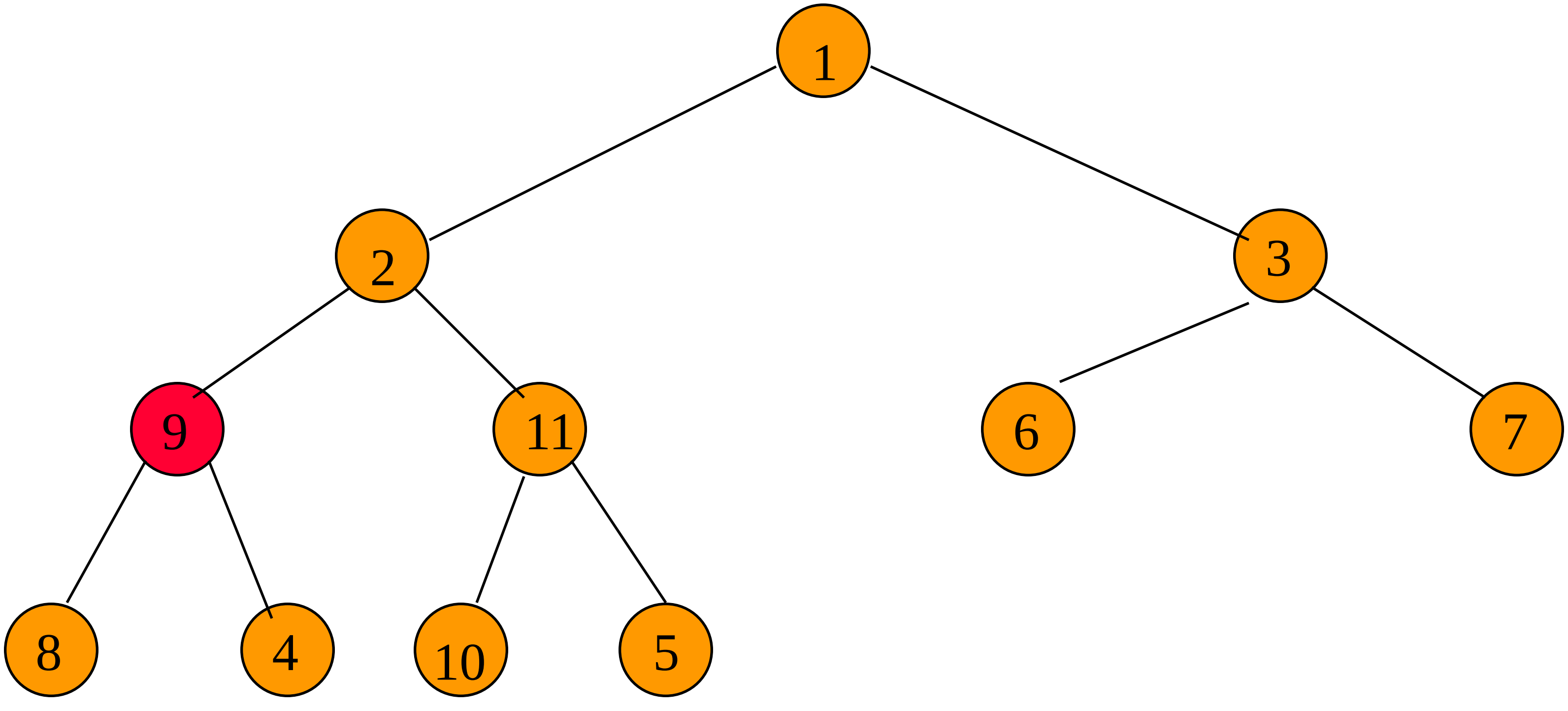


Move to next lower array position.

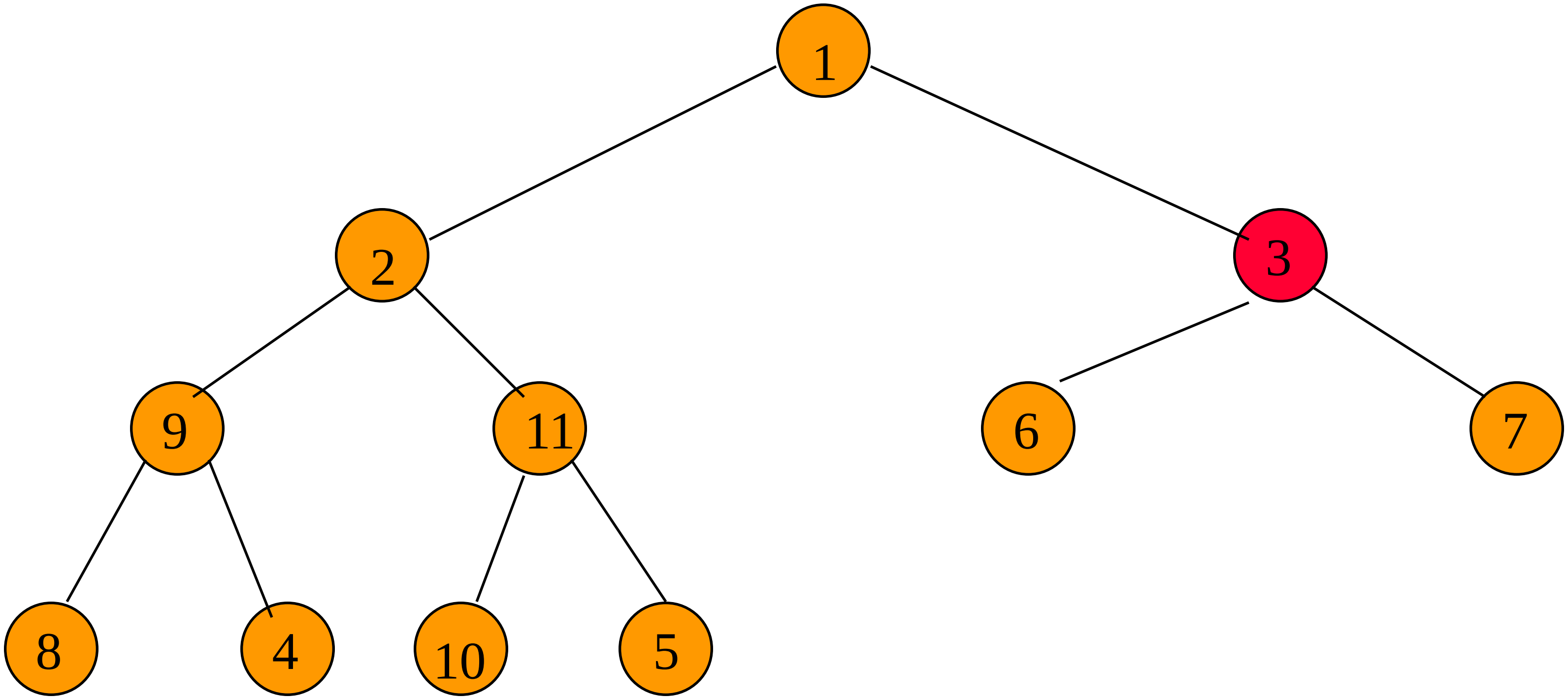
Initializing A Max Heap



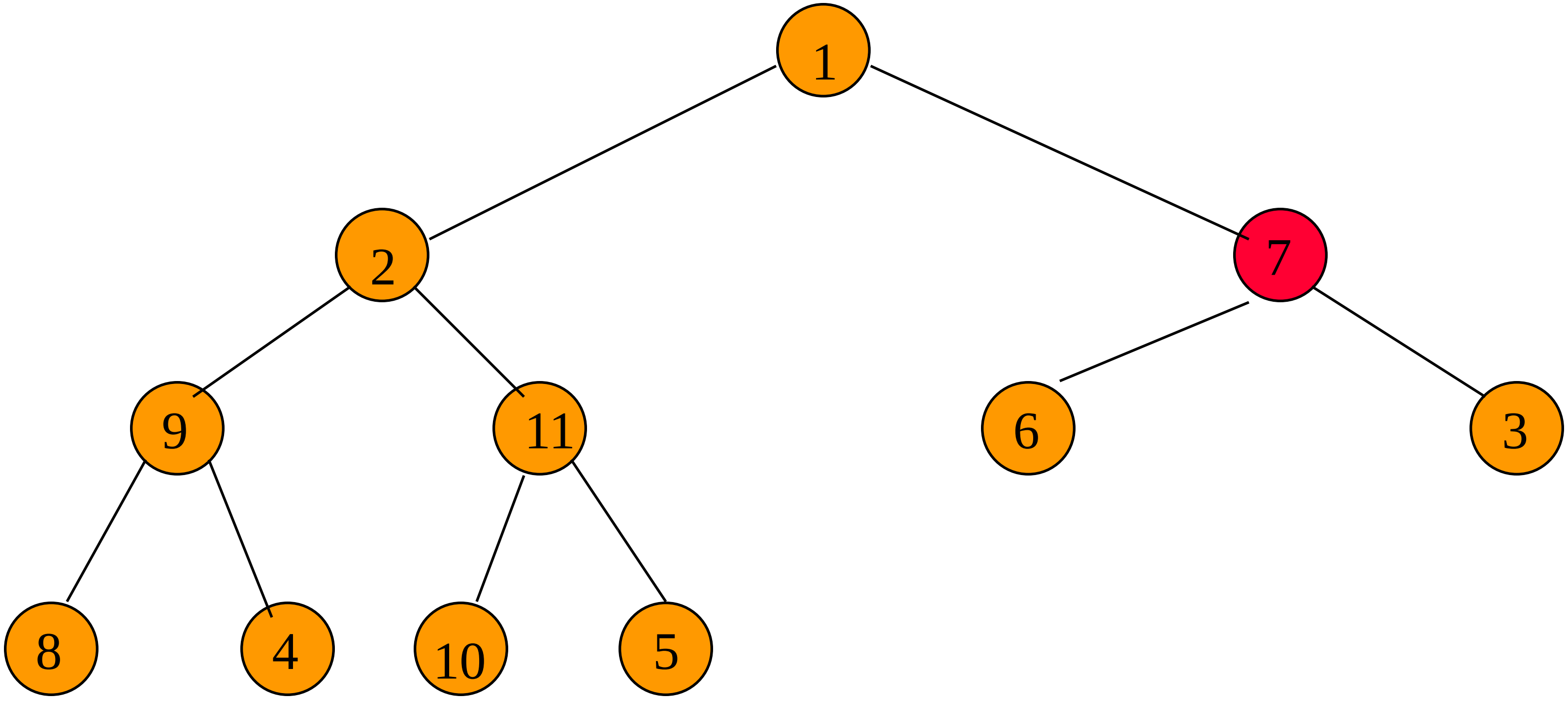
Initializing A Max Heap



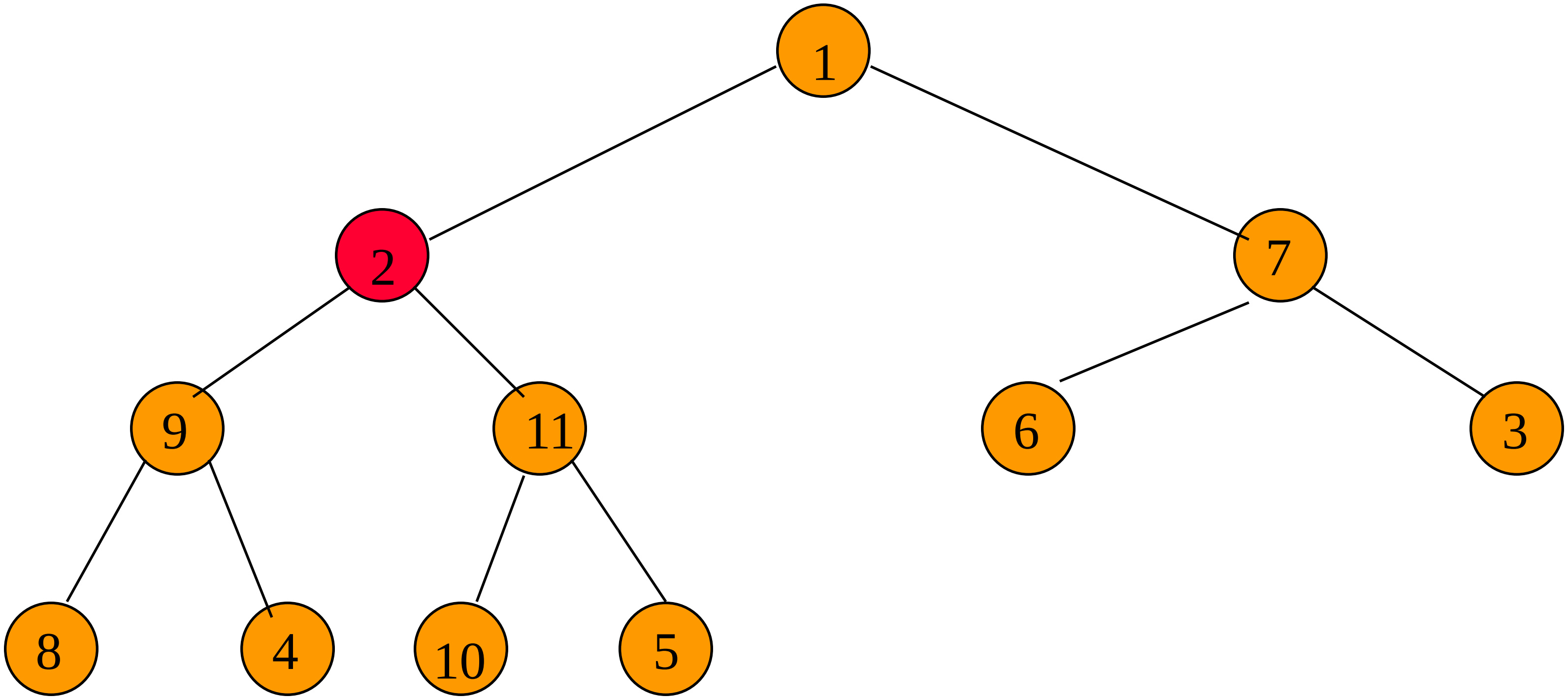
Initializing A Max Heap



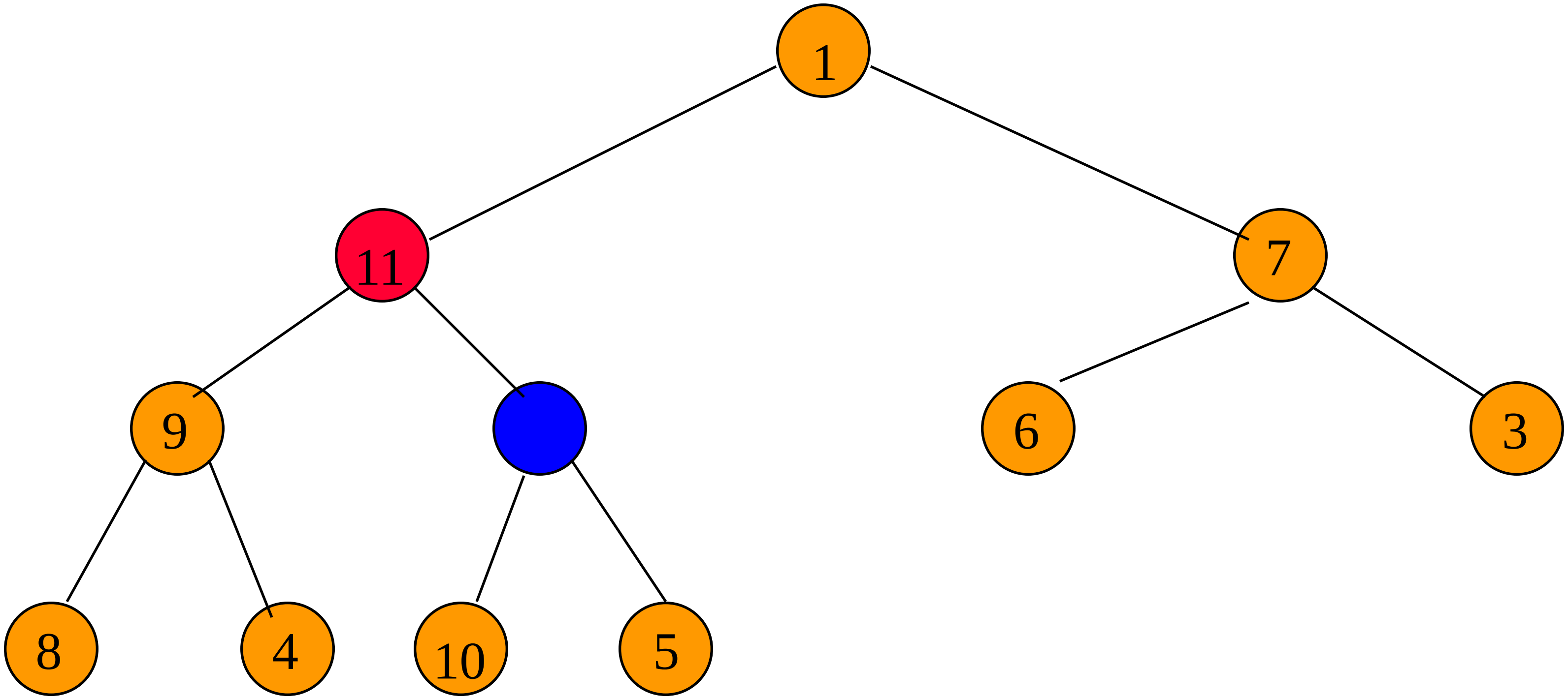
Initializing A Max Heap



Initializing A Max Heap

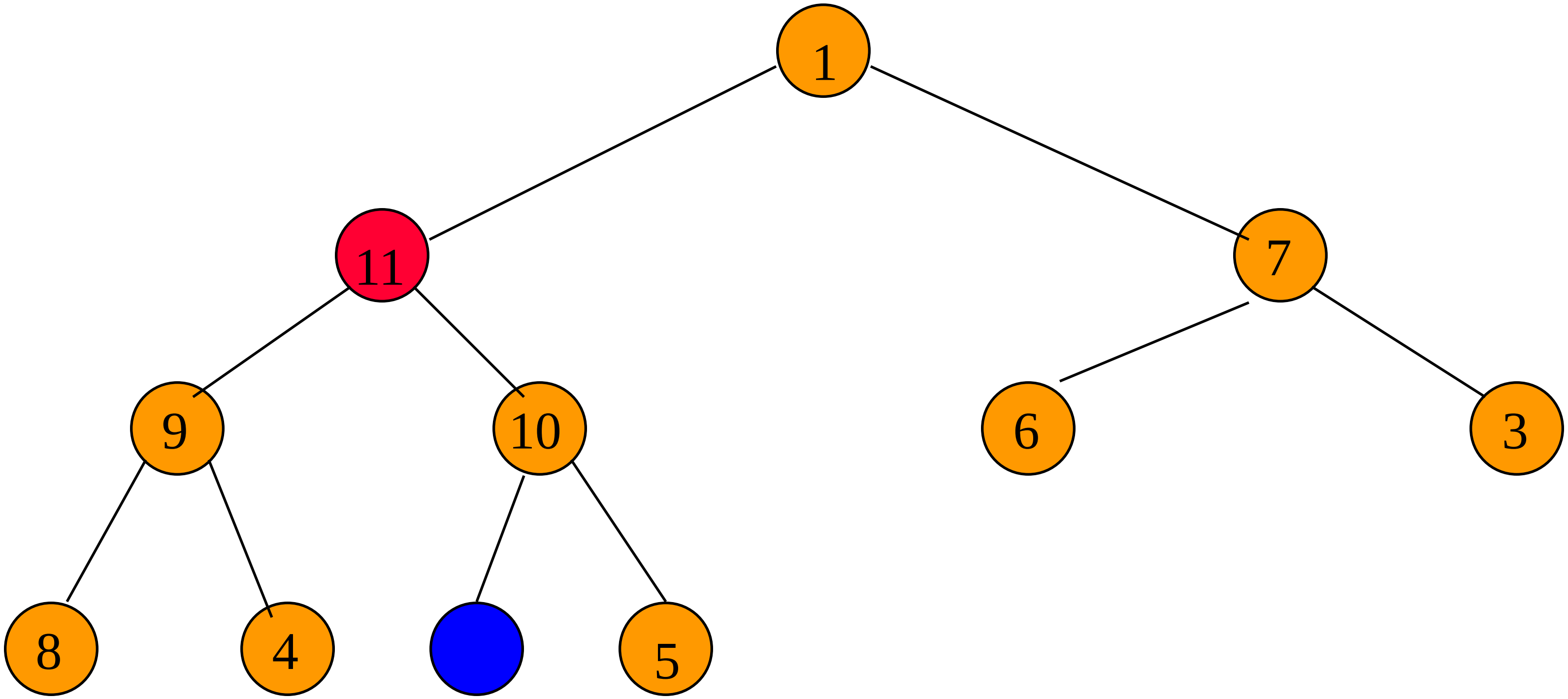


Initializing A Max Heap



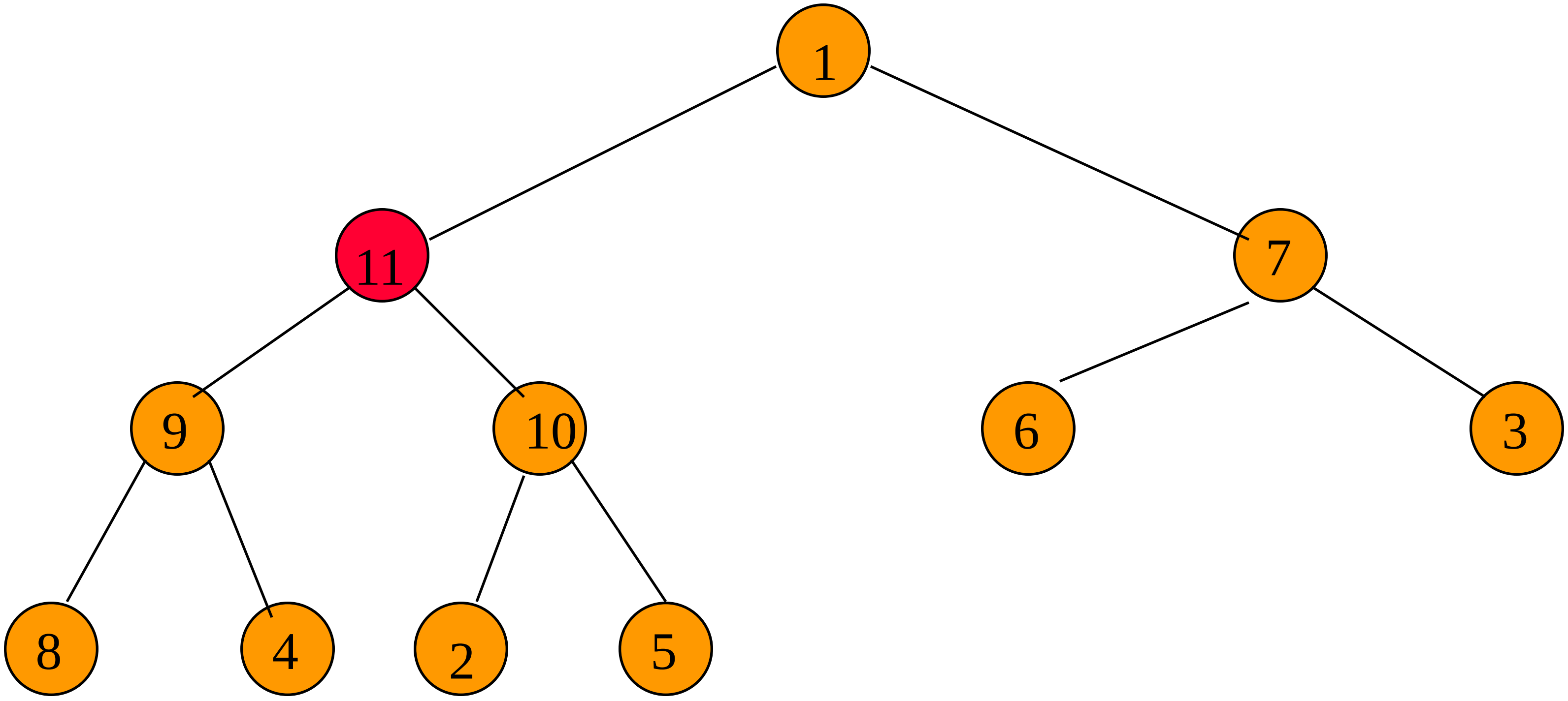
Find a home for 2.

Initializing A Max Heap



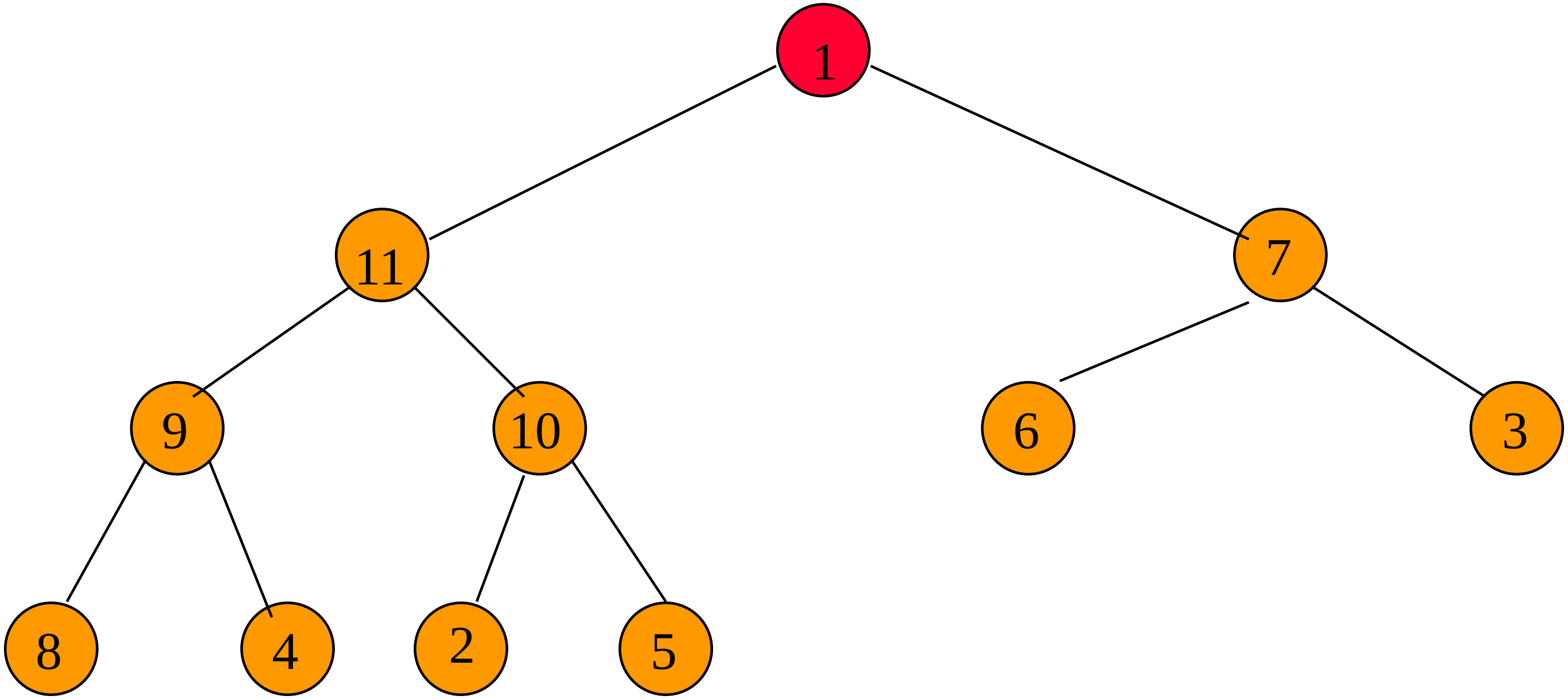
Find a home for 2.

Initializing A Max Heap



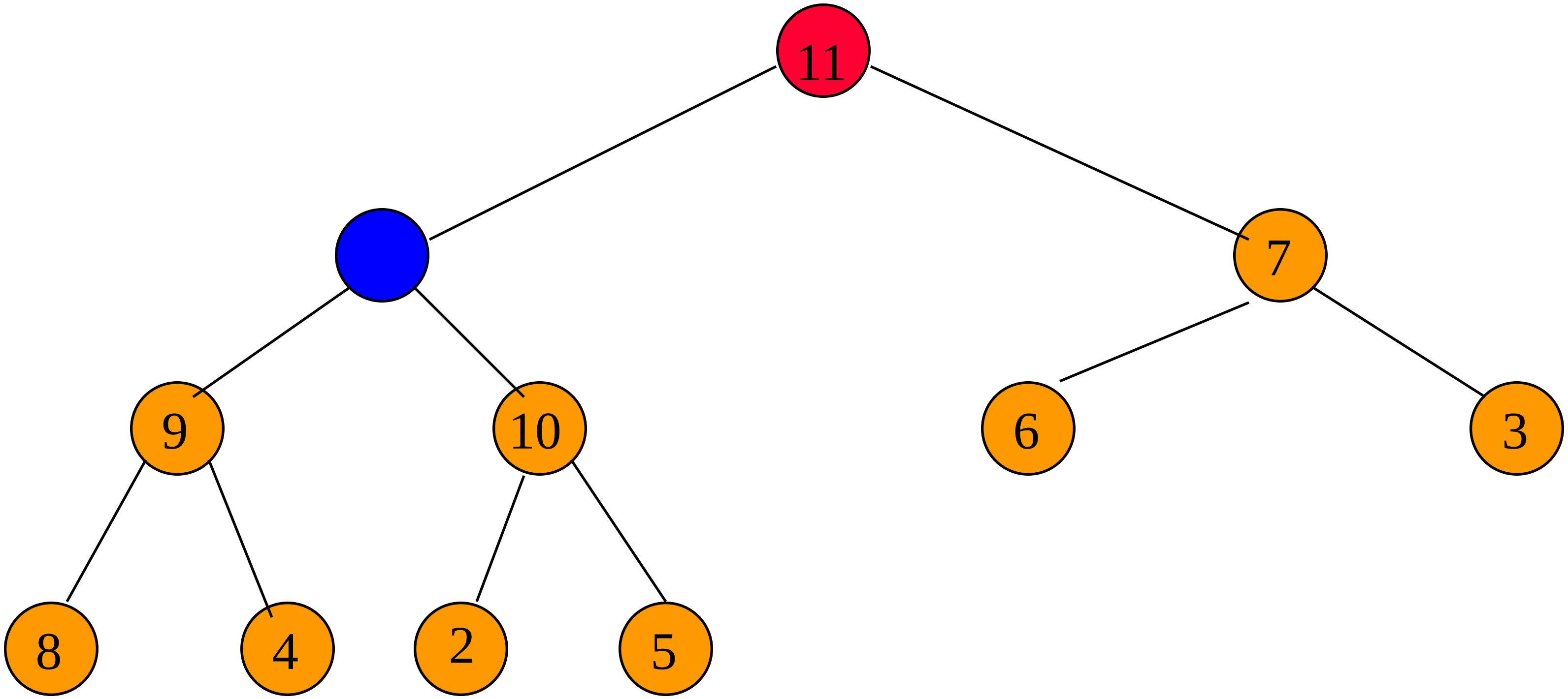
Done, move to next lower array position.

Initializing A Max Heap



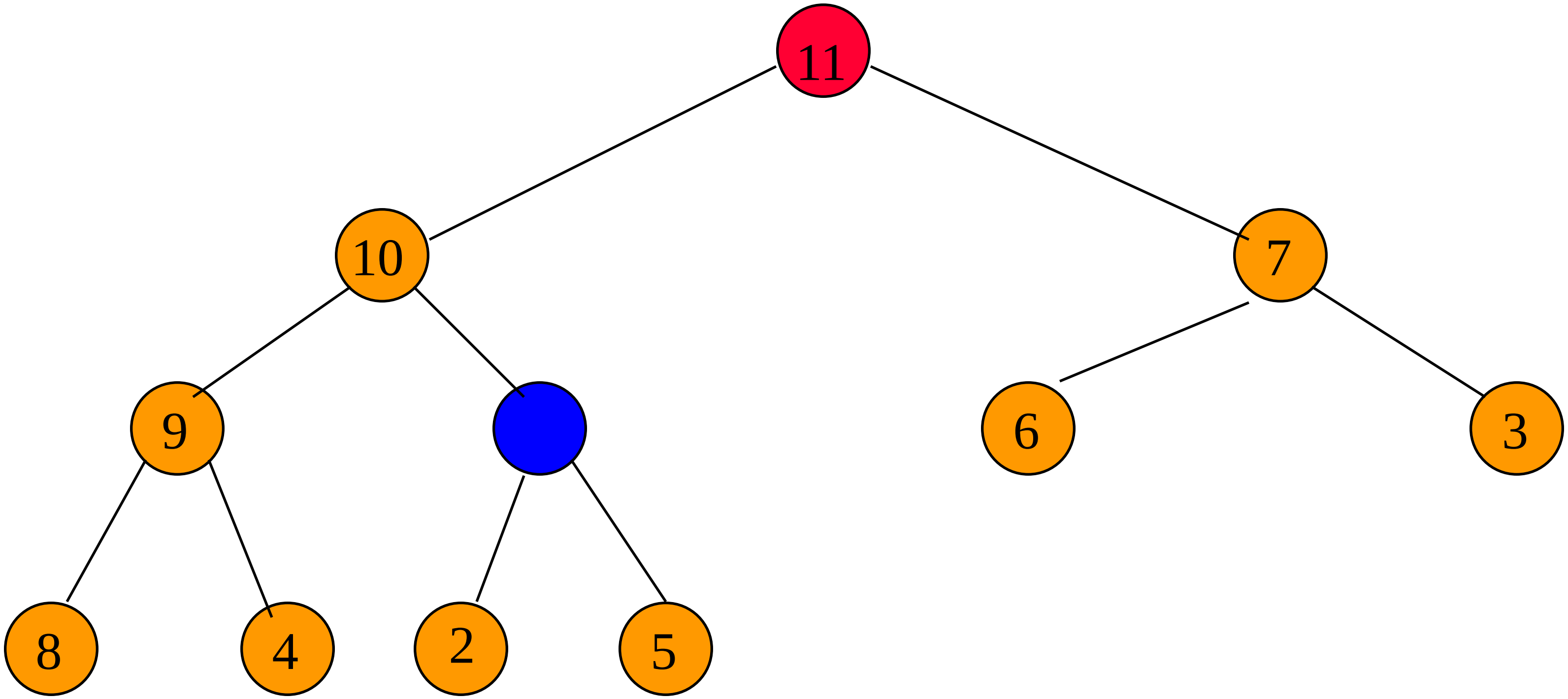
Find home for 1.

Initializing A Max Heap



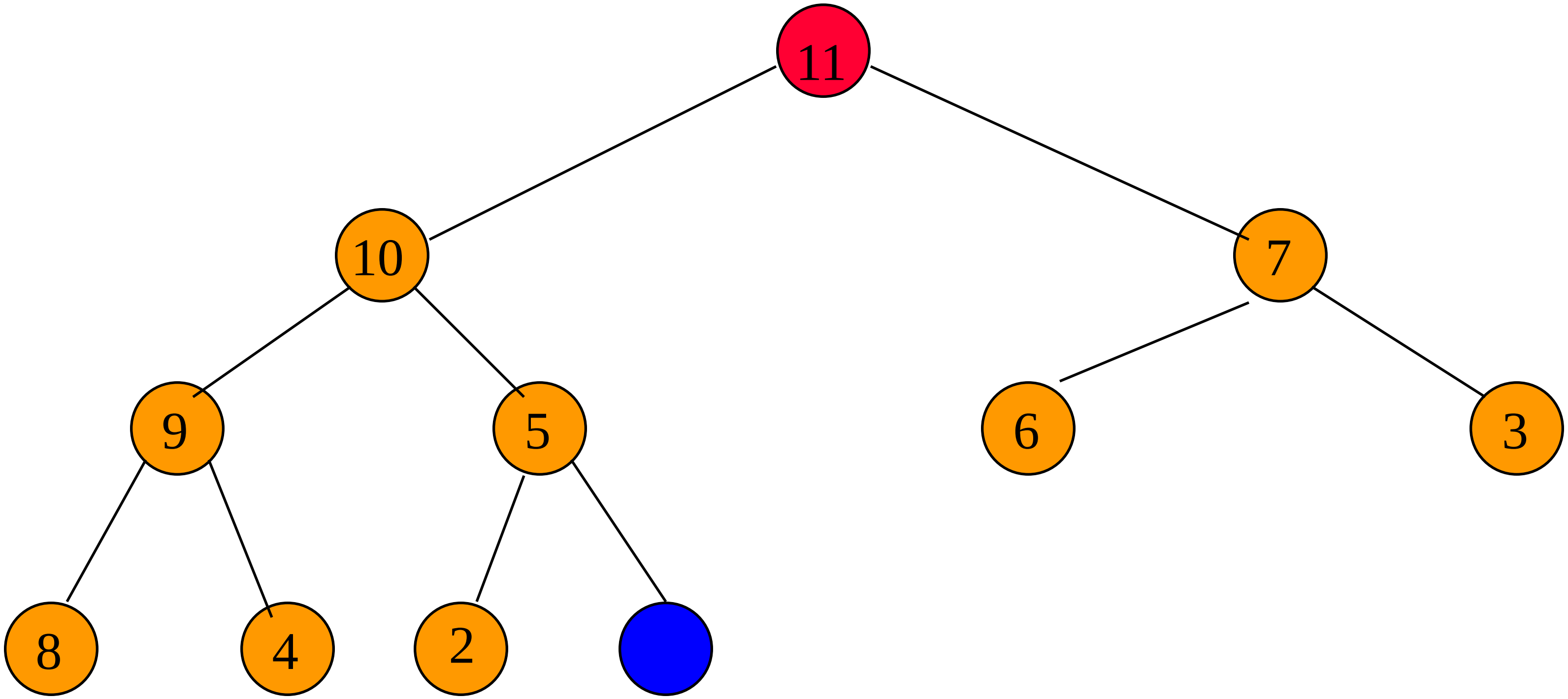
Find home for 1.

Initializing A Max Heap



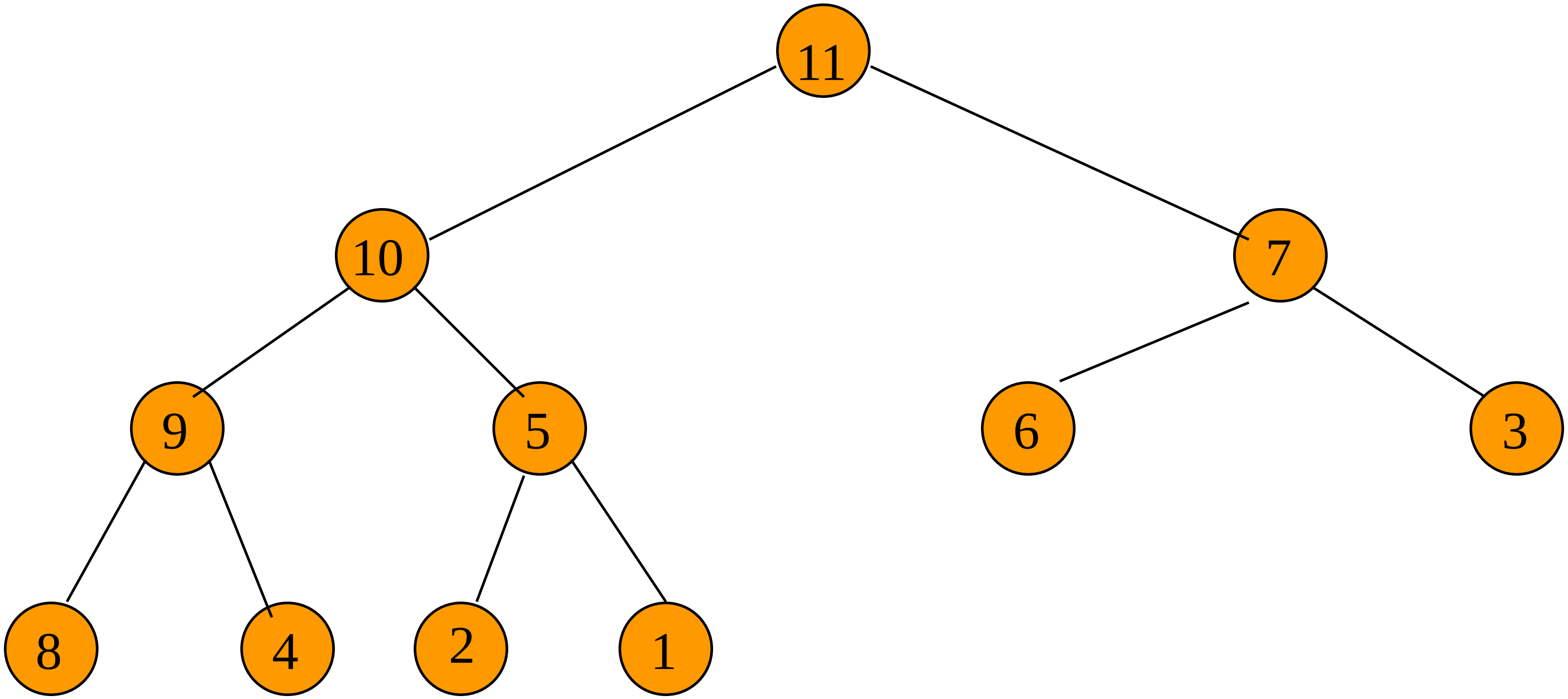
Find home for 1.

Initializing A Max Heap



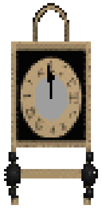
Find home for 1.

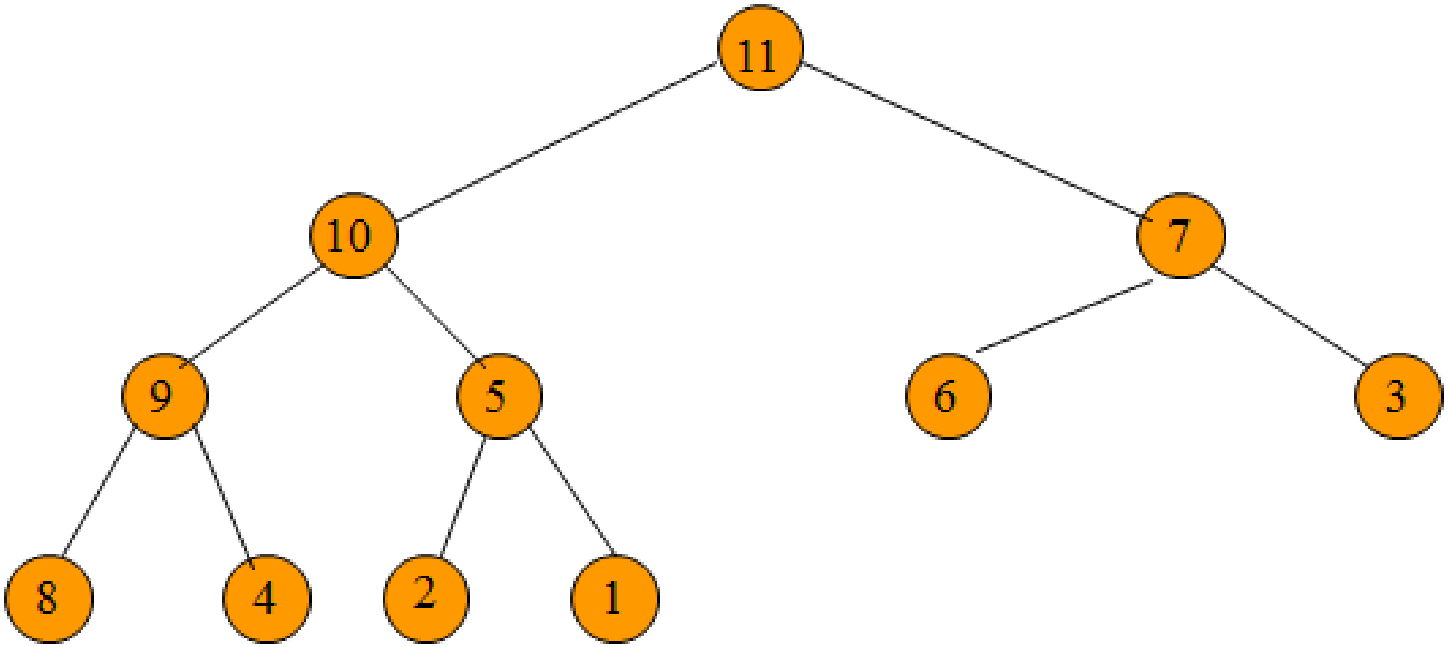
Initializing A Max Heap



Done.

Time Complexity





Cost of Max-Heapify (A, i) is O(log n)

Number of node/elements to be processed is n.

Total Time Complexity is **O(n log n)**.

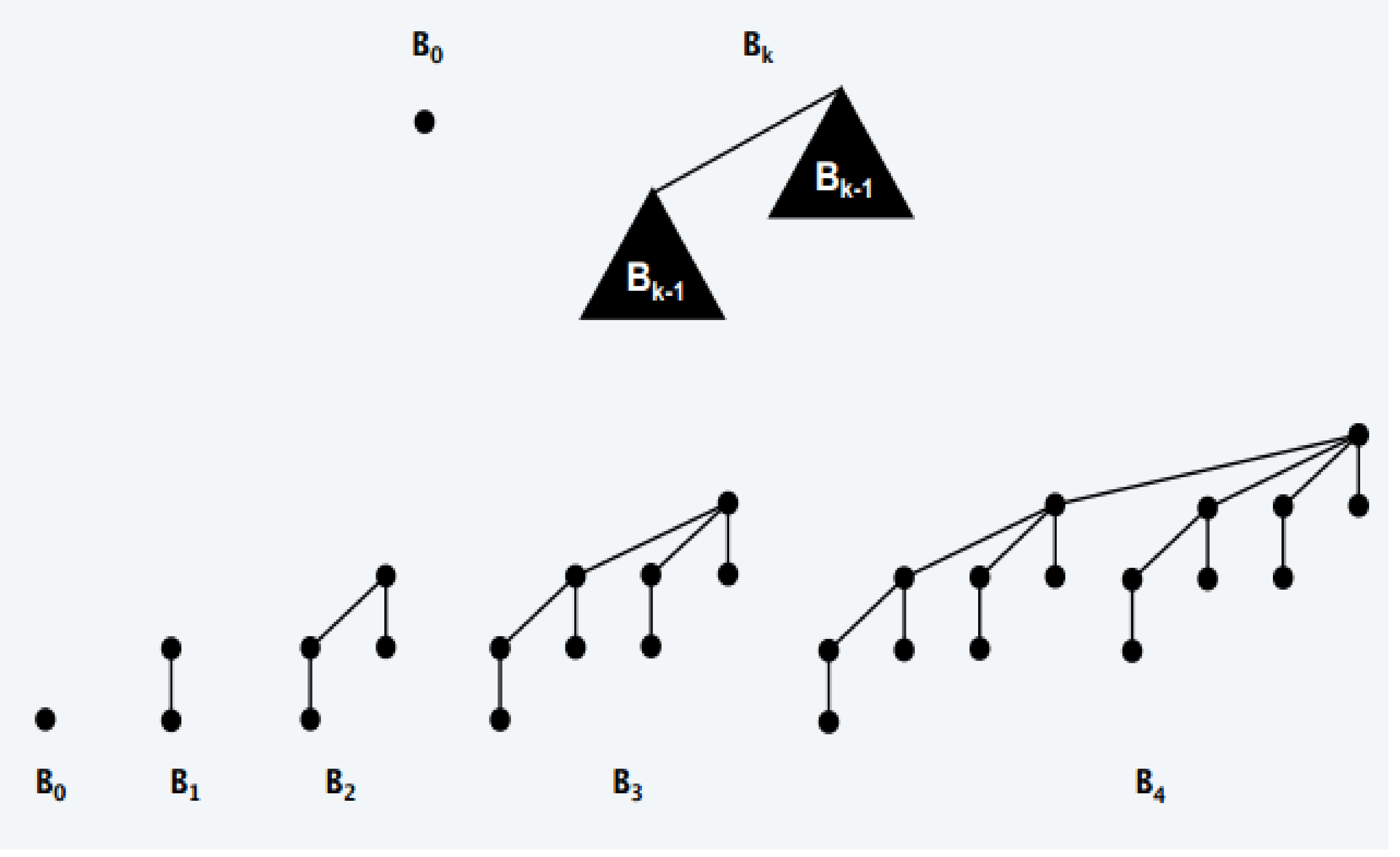
BINOMIAL HEAPS

Binomial Tree

Def. A binomial tree of order k is defined recursively:

- Order 0: single node.
- Order k: one binomial tree of order k –1 linked to another of order k – 1.

Binomial Tree

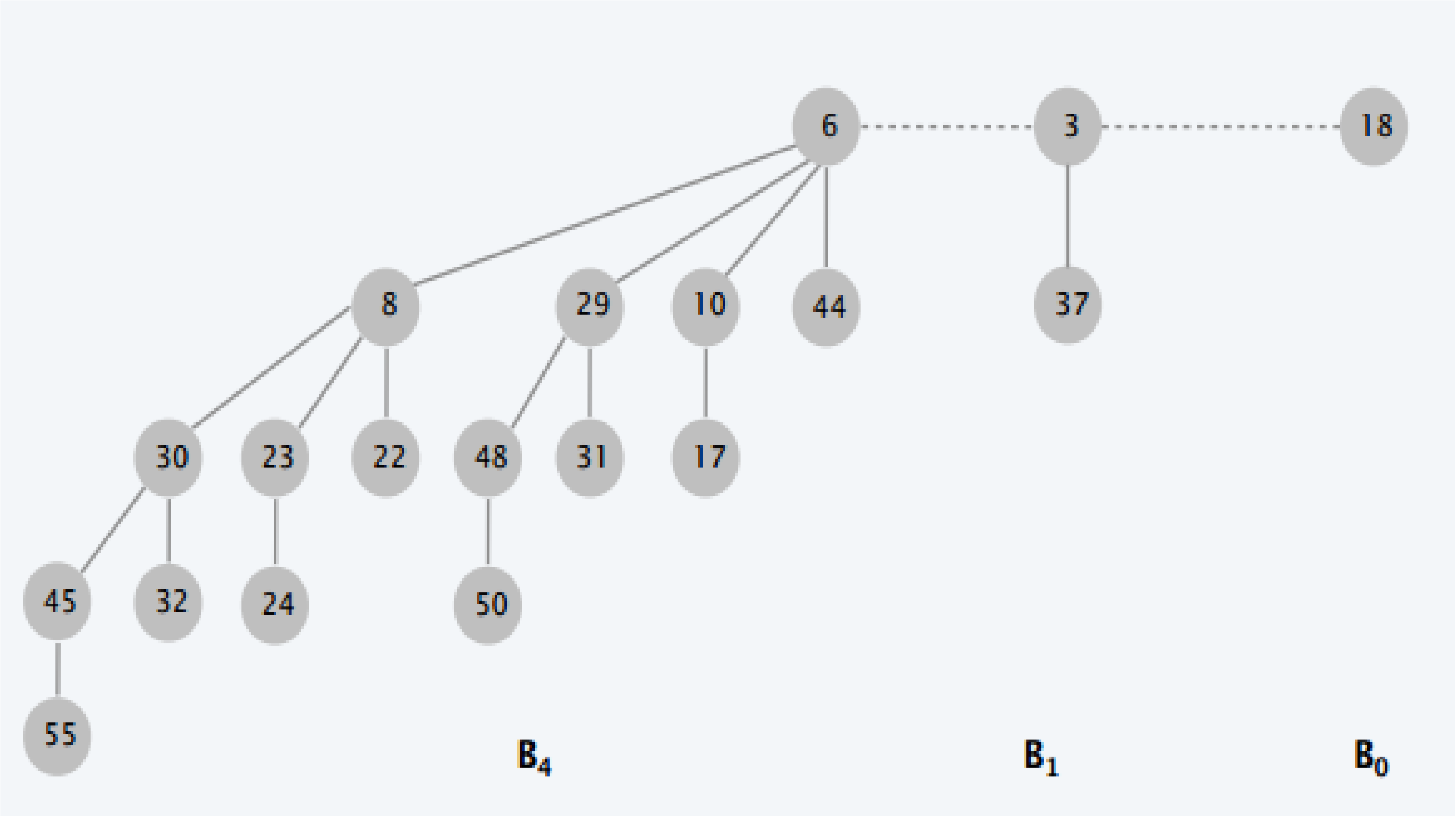


Binomial Heap

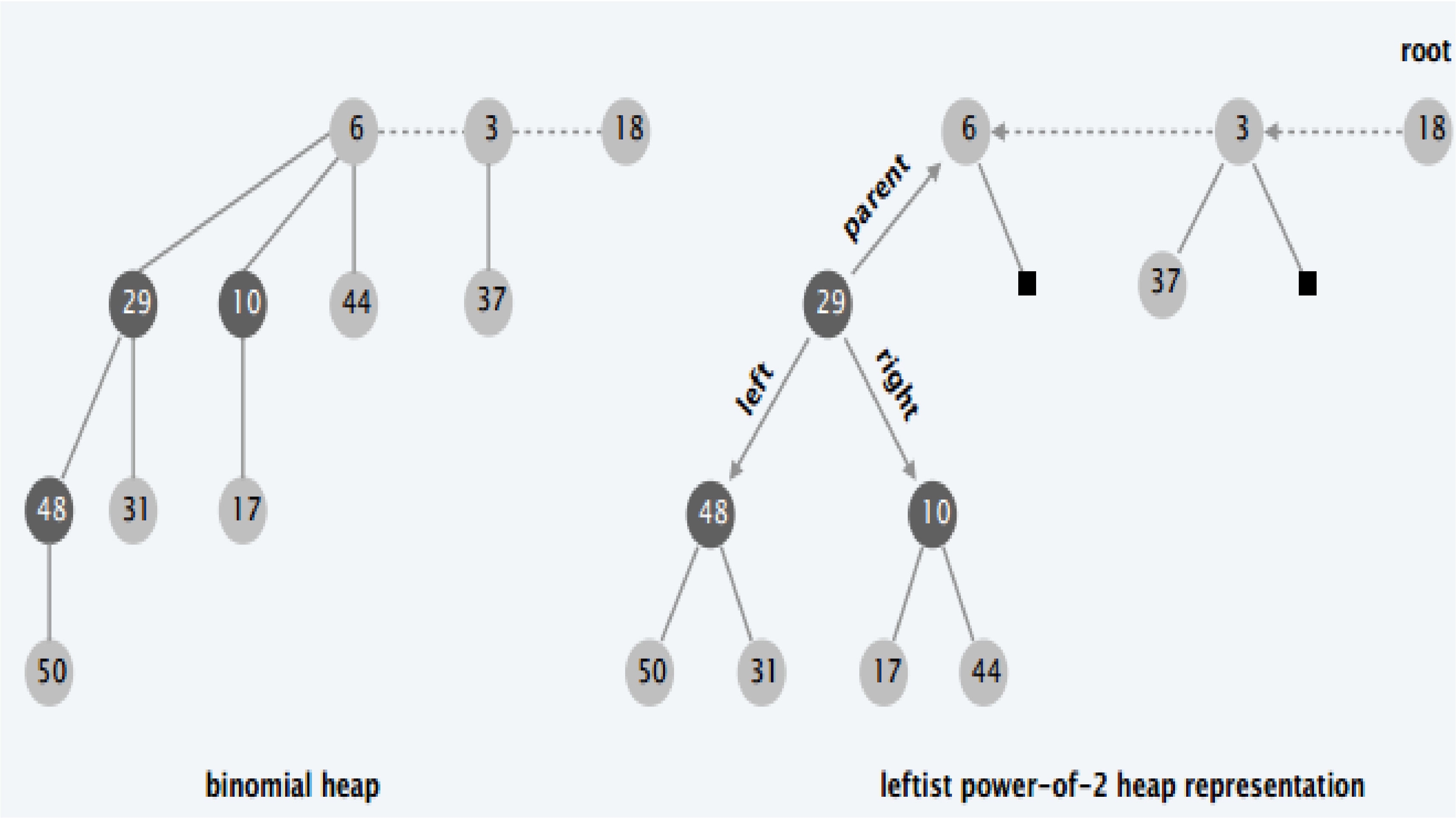
Def. A binomial heap is a sequence of binomial trees such that:

- Each tree is heap-ordered
- There is either 0 or 1 binomial tree of order k

Binomial Heap



Binomial Heap

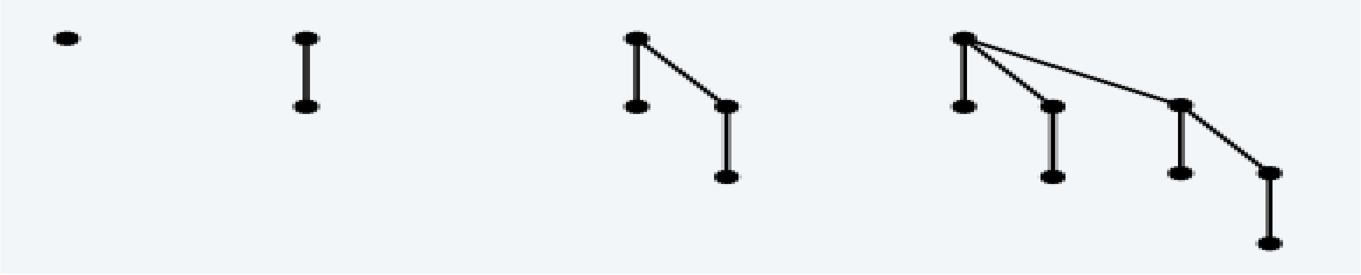


FIBONACCI HEAPS

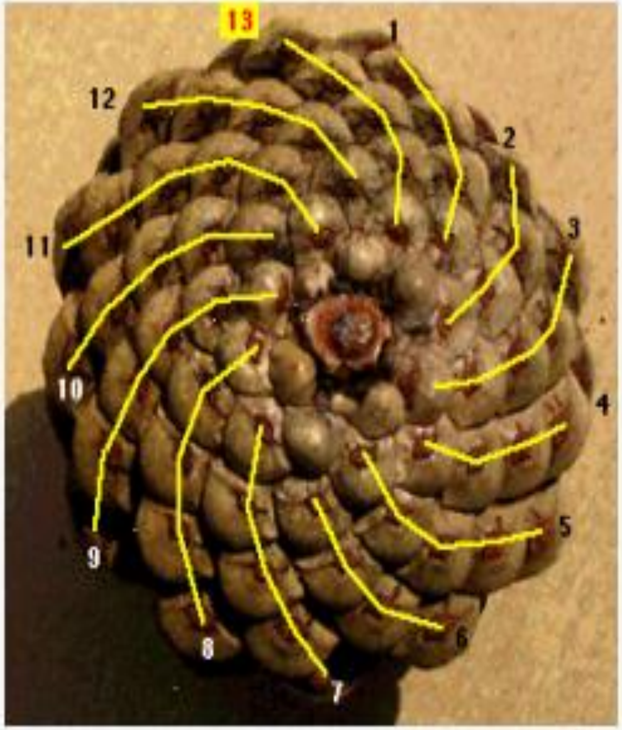
Fibonacci Heap

Basic Idea

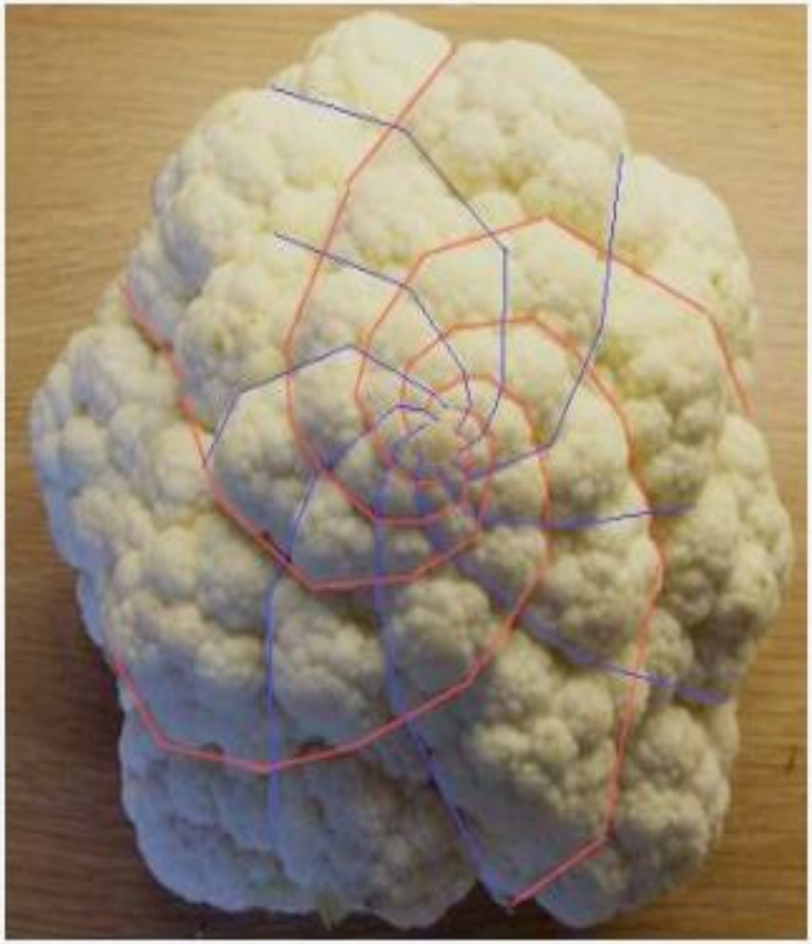
- Similar to binomial heaps, but less rigid structure
- **Binomial heap:** eagerly consolidate trees after each INSERT; implement DECREASE-KEY by repeatedly exchanging node with its parent



FIBONACCI HEAPS IN NATURE



pinecone



cauliflower

Application of Heap

- Sorting(Heap Sort)
- Priority Queues

• Event-driven simulation.	[customers in a line, colliding particles]
• Numerical computation.	[reducing roundoff error]
• Data compression.	[Huffman codes]
• Graph searching.	[Dijkstra's algorithm, Prim's algorithm]
• Number theory.	[sum of powers]
• Artificial intelligence.	[A* search]
• Statistics.	[maintain largest M values in a sequence]
• Operating systems.	[load balancing, interrupt handling]
• Discrete optimization.	[bin packing, scheduling]
• Spam filtering.	[Bayesian spam filter]

Heap Sort

- Algorithm for Heap Sort

```
HEAPSORT(A)
BUILD-MAX-HEAP(A)
for i = A.length downto 2
    exchange A[1] with A[i]
    A.heap-size = A.heap-size - 1
    MAX-HEAPIFY(A, 1)
```

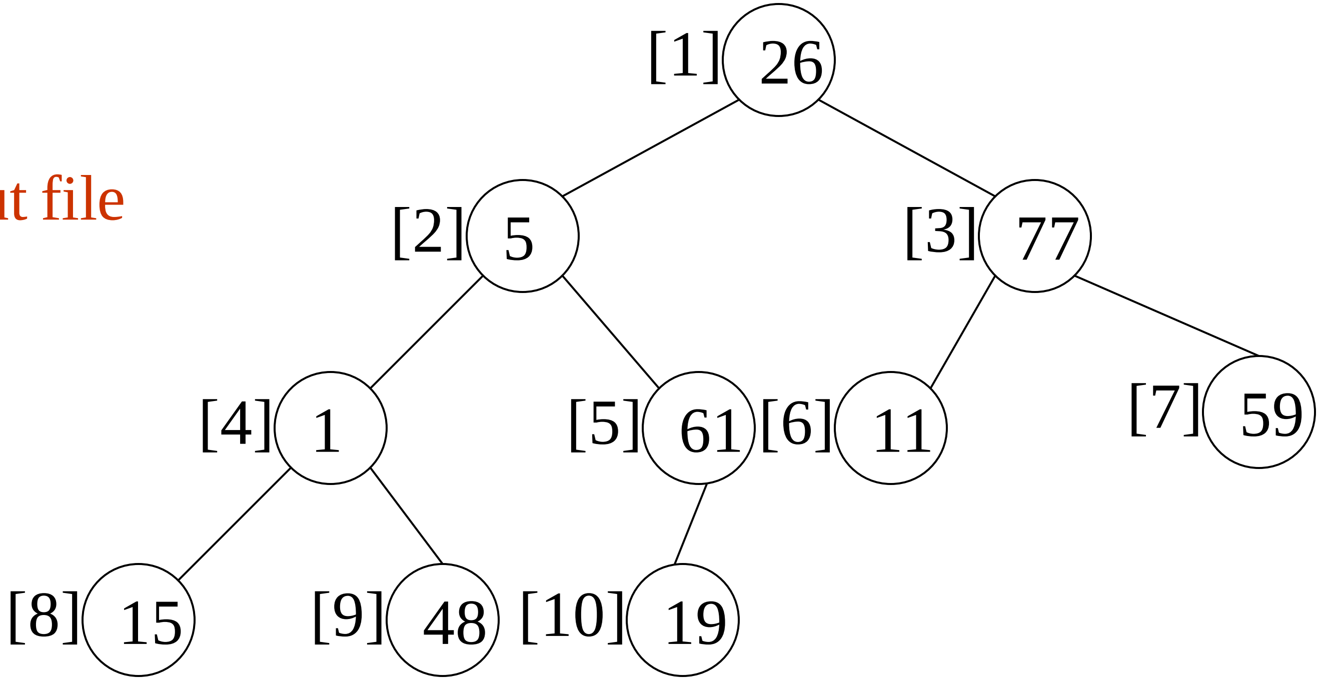
- Time Complexity is **O(n log n)**.

Heap Sort

- Array interpreted as a binary tree

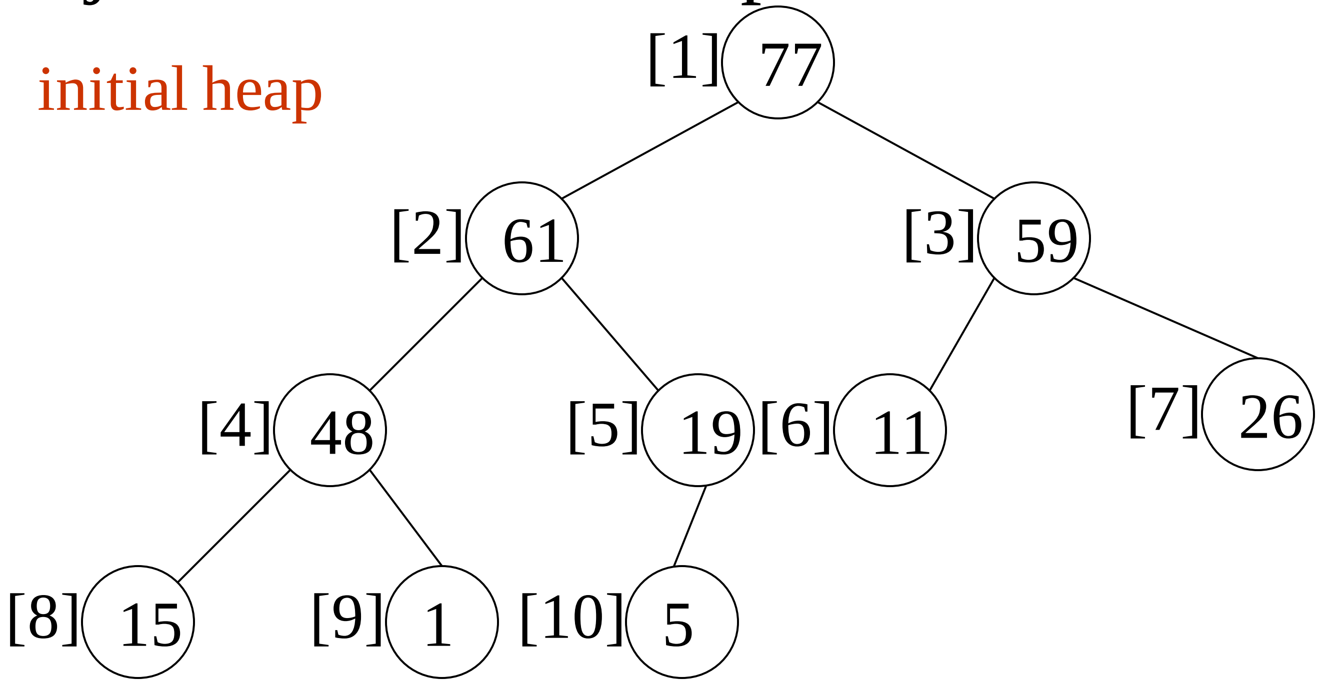
1	2	3	4	5	6	7	8	9	10
26	5	77	1	61	11	59	15	48	19

input file



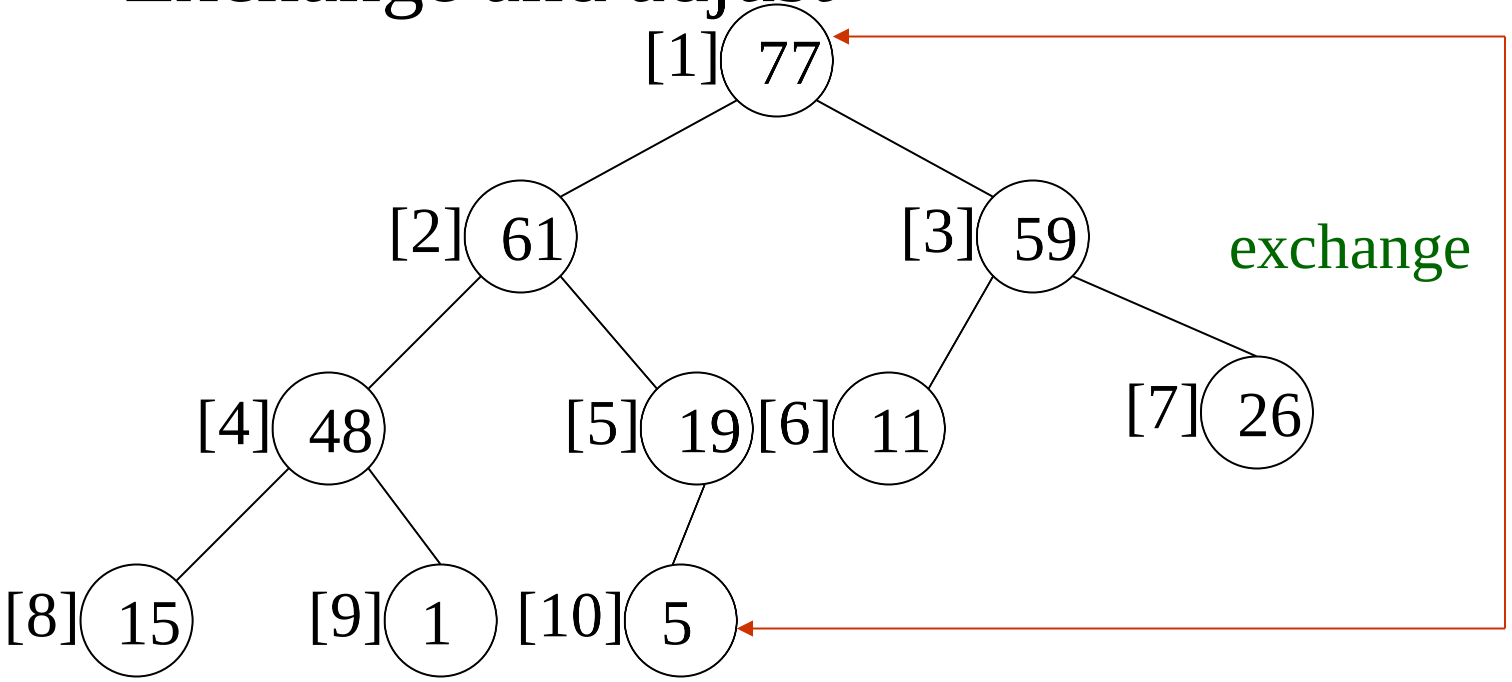
Heap Sort

- Adjust it to a MaxHeap

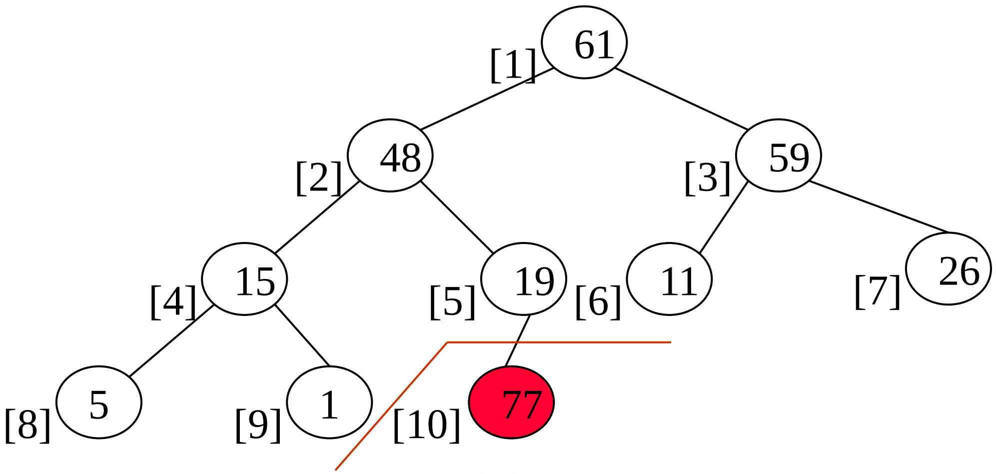


Heap Sort

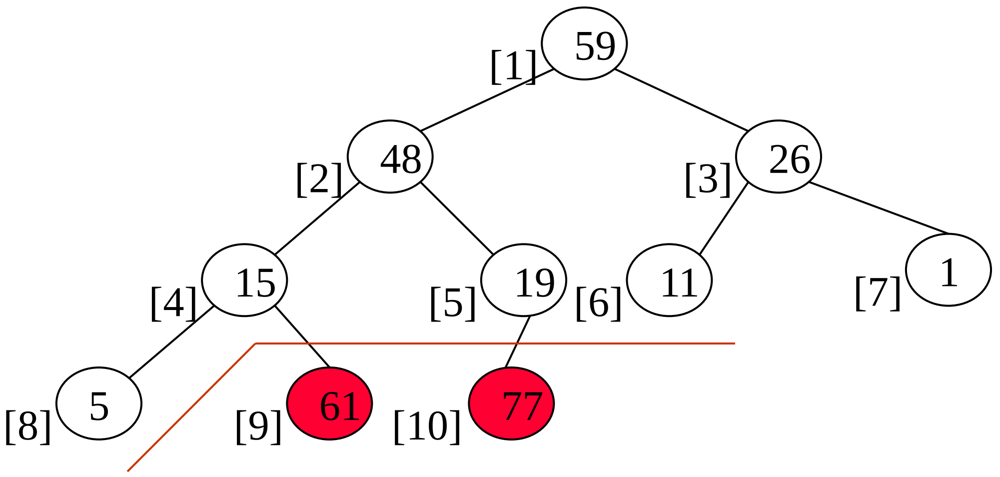
- Exchange and adjust



Heap Sort

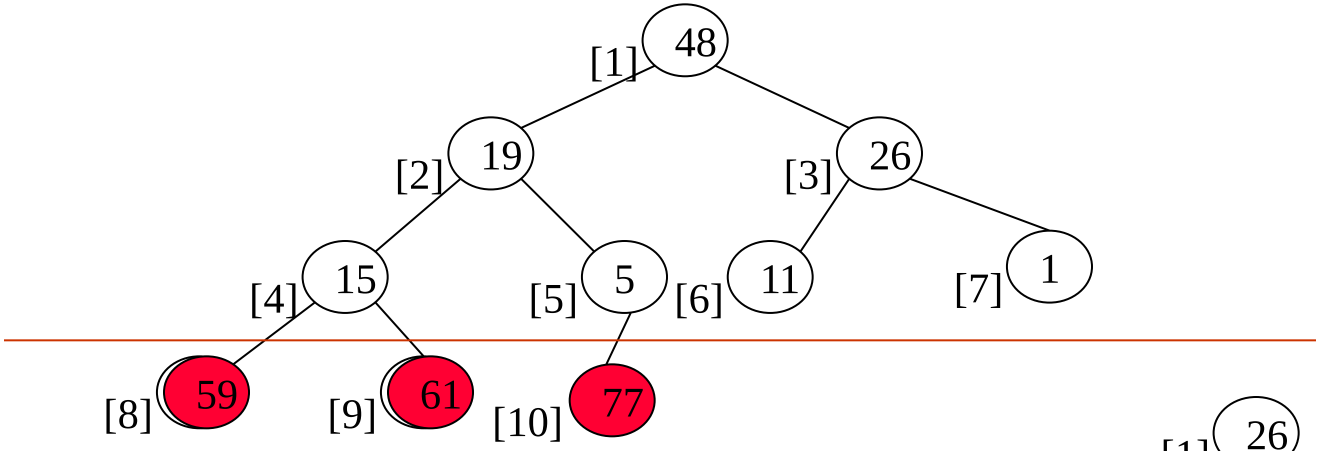

[1] 61
[2] 48
[3] 59
[4] 15
[5] 19
[6] 11
[7] 26
[8] 5
[9] 1
[10] 77

(a)

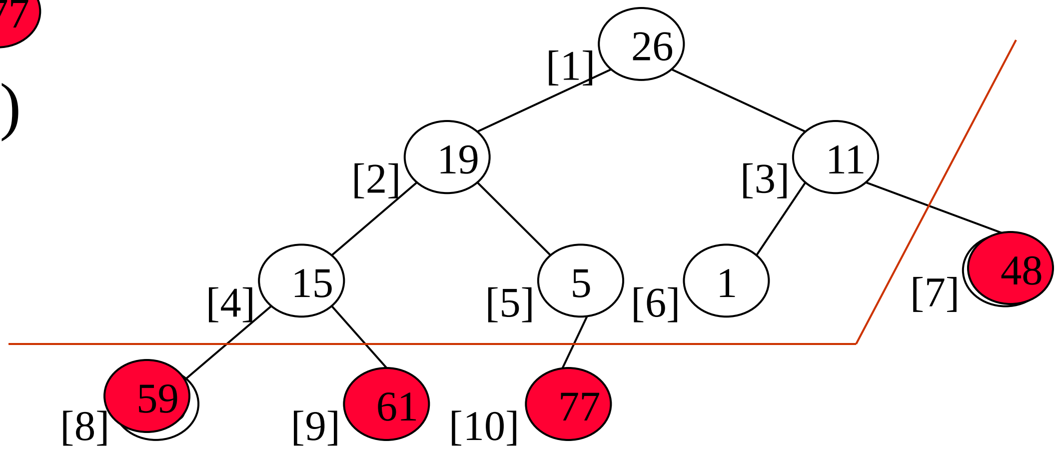

[1] 59
[2] 48
[3] 26
[4] 15
[5] 19
[6] 11
[7] 1
[8] 5
[9] 61
[10] 77

(b)

Heap Sort

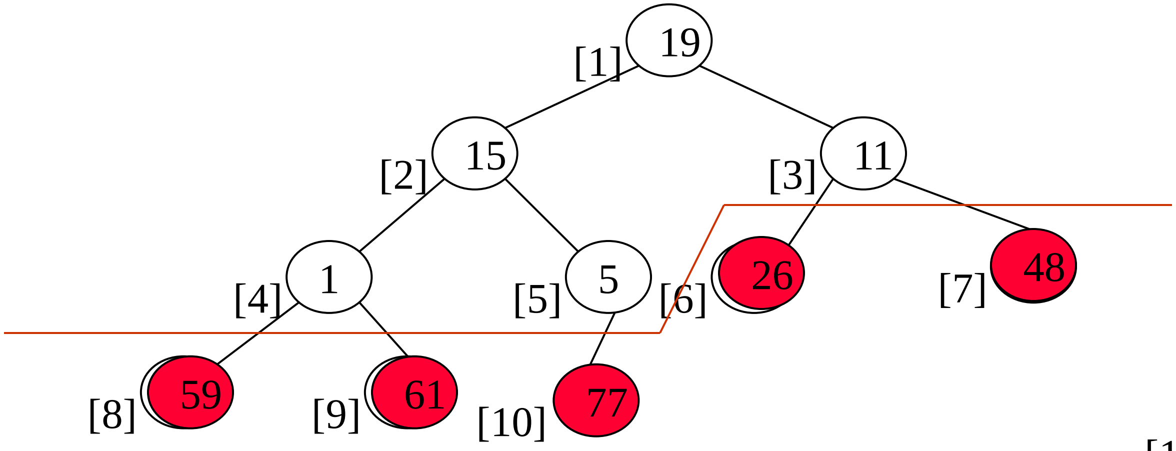

48
[1]
19
[2]
26
[3]
15
[4]
5
[5]
11
[6]
1
[7]
59
[8]
61
[9]
77
[10]

(c)


26
[1]
19
[2]
11
[3]
15
[4]
5
[5]
1
[6]
48
[7]
59
[8]
61
[9]
77
[10]

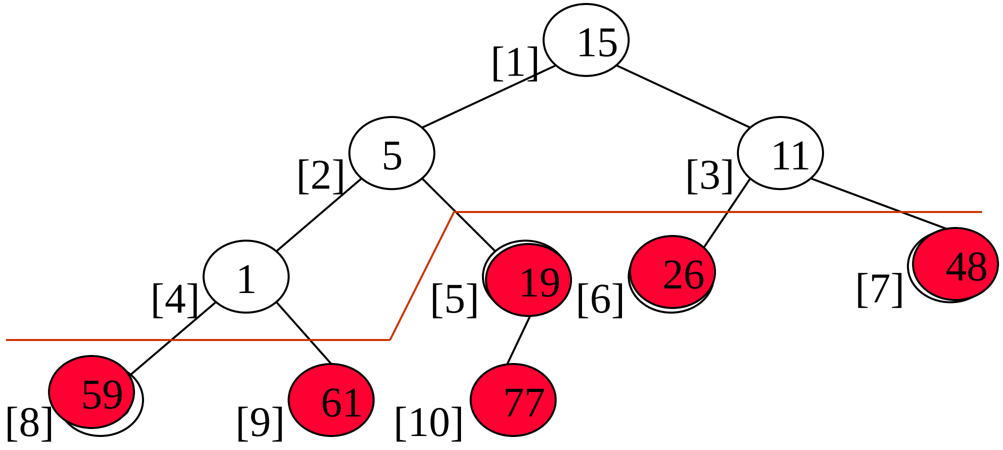
(d)

Heap Sort



19
[1]
15
[2]
11
[3]
1
[4]
5
[5]
26
[6]
48
[7]
59
[8]
61
[9]
77
[10]

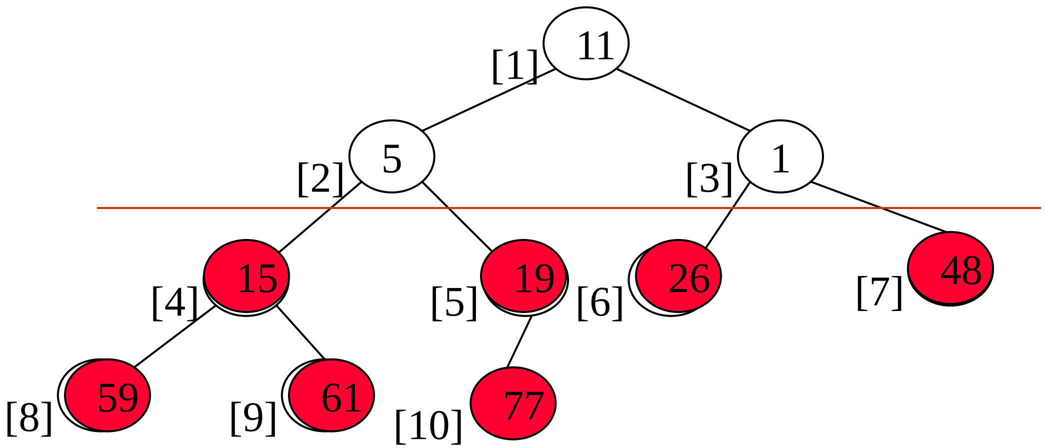
(e)



15
[1]
5
[2]
11
[3]
1
[4]
19
[5]
26
[6]
48
[7]
59
[8]
61
[9]
77
[10]

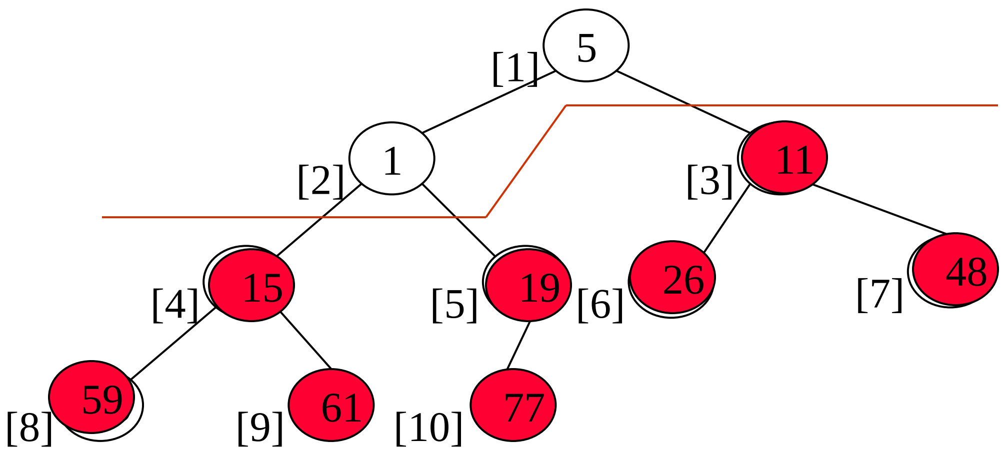
(f)

Heap Sort



11
[1]
5
[2]
1
[3]
15
[4]
19
[5]
26
[6]
48
[7]
59
[8]
61
[9]
77
[10]

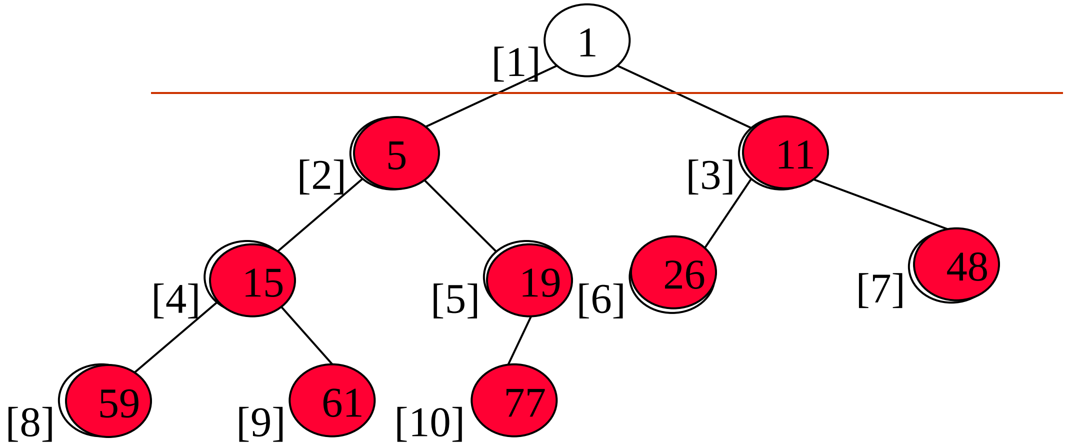
(g)



5
[1]
1
[2]
11
[3]
15
[4]
19
[5]
26
[6]
48
[7]
59
[8]
61
[9]
77
[10]

(h)

Heap Sort



(i)

- So results

77 61 59 48 26 19 15 11 5 1

Priority Queue

- A priority queue is a data structure for maintaining a set S of elements, each with an associated value called a key.
- Two kinds of priority queues:
  - Min priority queue
  - Max priority queue

Min Priority Queue

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

Max Priority Queue

- Collection of elements.
- Each element has a priority or key.
- 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)

Priority Queue

- Algorithm for Priority Queue

```
HEAP-EXTRACT-MAX(A)
if A.heap-size < 1
    error "heap underflow"
max = A[1]
A[1] = A[A.heap-size]
A.heap-size = A.heap-size - 1
MAX-HEAPIFY(A, 1)
return max
```

Complexity Of Operations

Using a heap:

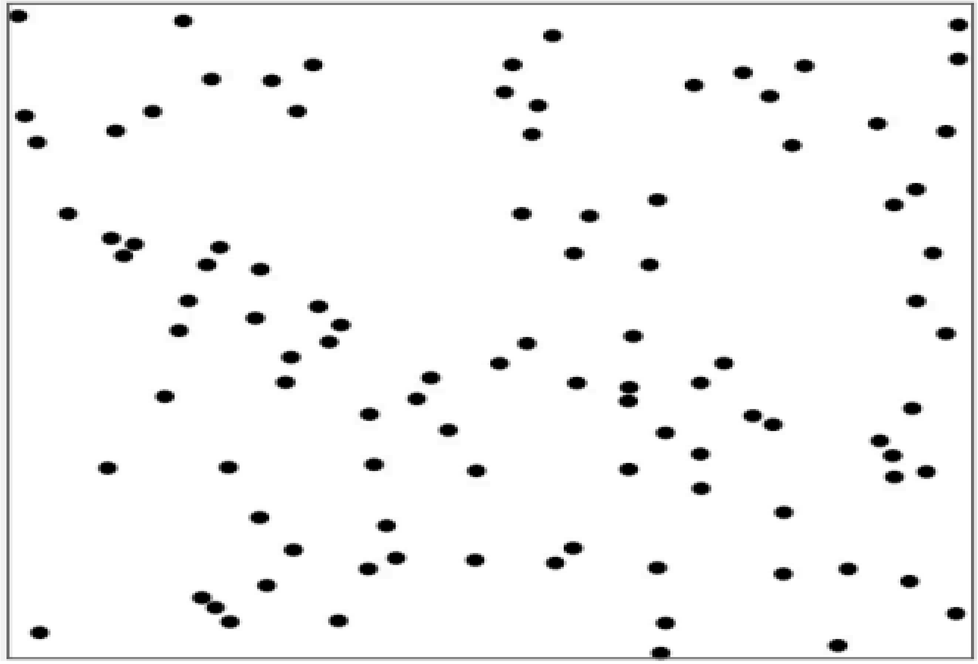
- empty, size, and top => O(1) time
- insert (push) and remove (pop) => O(log n) time where n is the size of the priority queue

Priority Queue

- Use max-priority queues to schedule jobs on a shared computer
- The max-priority queue keeps track of the jobs to be performed and their relative priorities
- When a job is finished or interrupted, the scheduler selects the highest-priority job from among those pending by calling EXTRACT-MAX
- The scheduler can add a new job to the queue at any time by calling INSERT

Event-Driven Simulation

- Goal: Simulate the motion of N moving particles that behave according to the laws of elastic collision.



Event-Driven Simulation

Significance: Relates macroscopic observables to microscopic dynamics

- Maxwell-Boltzmann: distribution of speeds as a function of temperature.
- Einstein: explain Brownian motion of pollen grains

Over-All Analysis of Heap

operation	linked list	binary heap	binomial heap	Fibonacci heap †
MAKE-HEAP	$O(1)$	$O(1)$	$O(1)$	$O(1)$
IS-EMPTY	$O(1)$	$O(1)$	$O(1)$	$O(1)$
INSERT	$O(1)$	$O(\log n)$	$O(\log n)$	$O(1)$
EXTRACT-MIN	$O(n)$	$O(\log n)$	$O(\log n)$	$O(\log n)$
DECREASE-KEY	$O(1)$	$O(\log n)$	$O(\log n)$	$O(1)$
DELETE	$O(1)$	$O(\log n)$	$O(\log n)$	$O(\log n)$
MELD	$O(1)$	$O(n)$	$O(\log n)$	$O(1)$
FIND-MIN	$O(n)$	$O(1)$	$O(\log n)$	$O(1)$

Some More Food

Heaps of heaps

- b-heaps.
- Fat heaps.
- 2-3 heaps.
- Leaf heaps.
- Thin heaps.
- Skew heaps.
- Splay heaps.
- Weak heaps.
- Leftist heaps.
- Quake heaps.
- Pairing heaps.
- Violation heaps.
- Run-relaxed heaps.
- Rank-pairing heaps.
- Skew-pairing heaps.
- Rank-relaxed heaps.
- Lazy Fibonacci heaps.