ساختمان داده ها

هرم
(Heap)

مدرس: غیاثی‌شیرازی
دانشگاه فردوسی مشهد

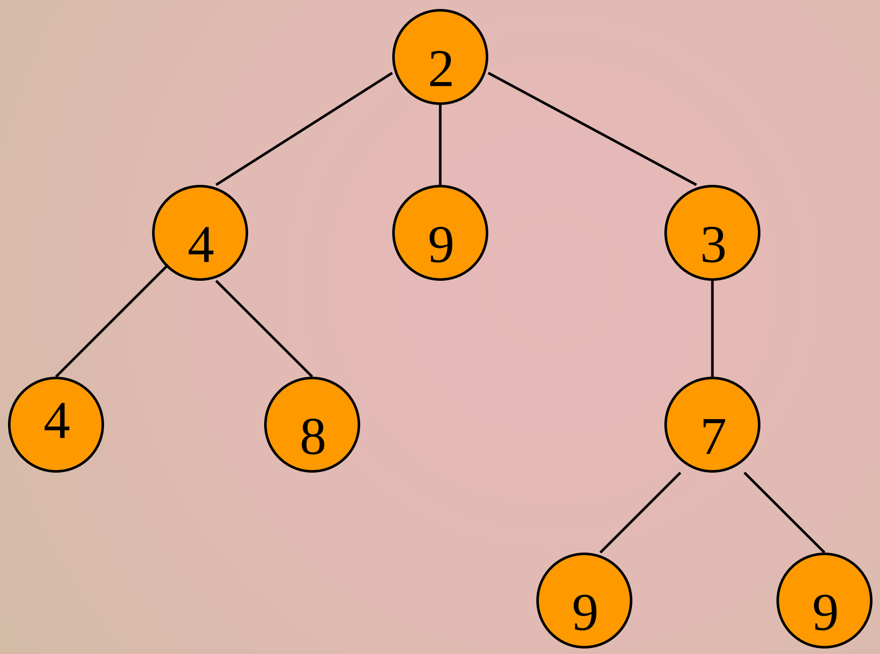
Min Tree Definition

Each tree node has a value.

Value in any node is the minimum value in the subtree for which that node is the root.

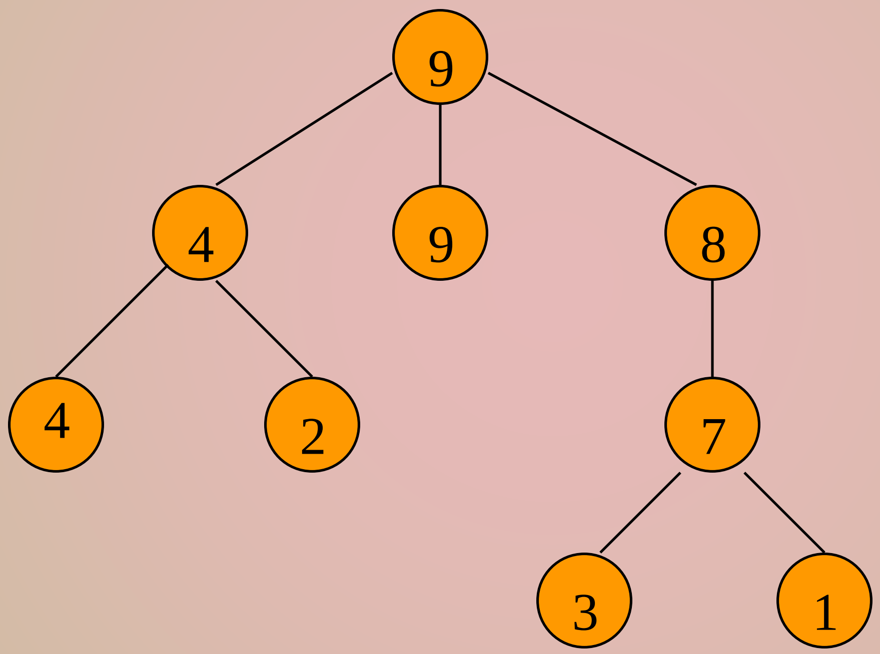
Equivalently, no descendent has a smaller value.

Min Tree Example



Root has minimum element.

Max Tree Example

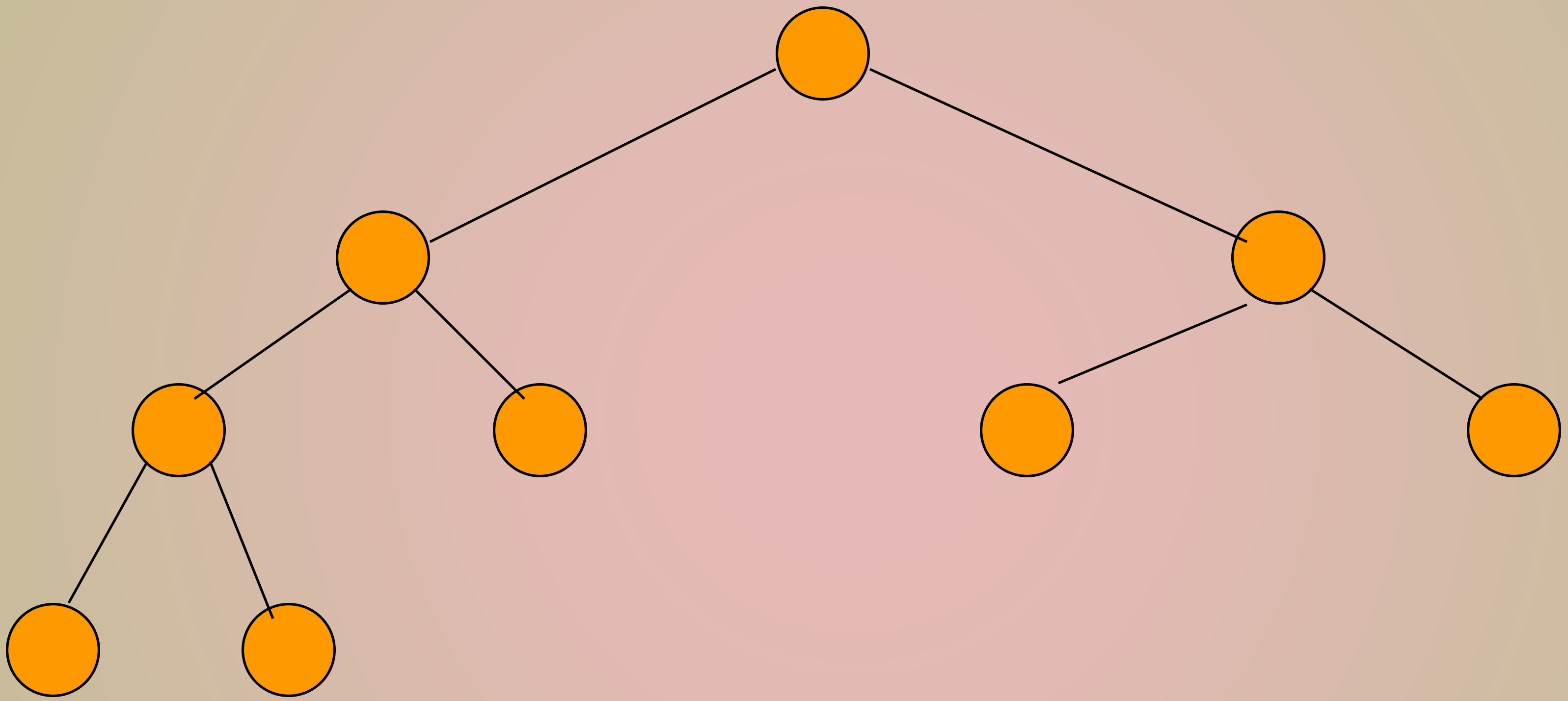


Root has maximum element.

Min Heap Definition

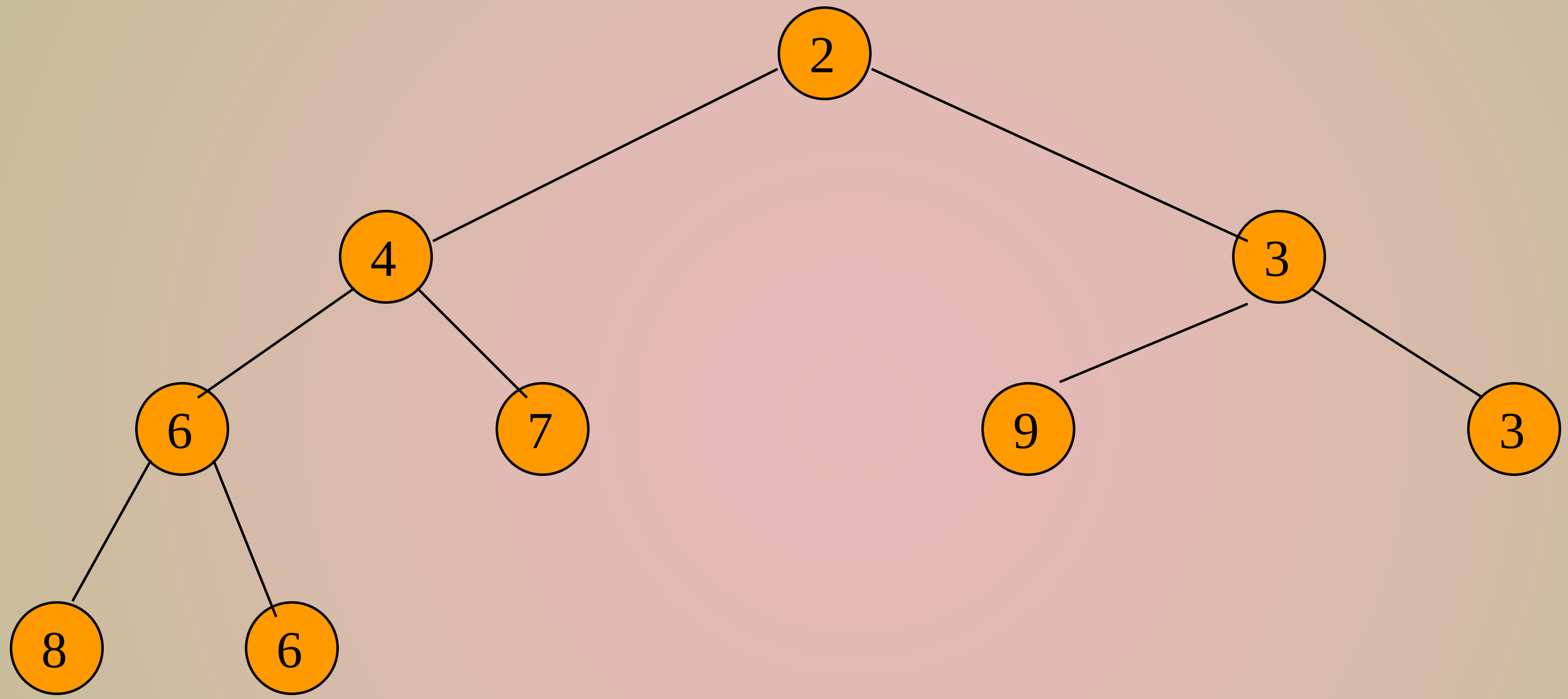
- complete binary tree
- min tree

Min Heap With 9 Nodes



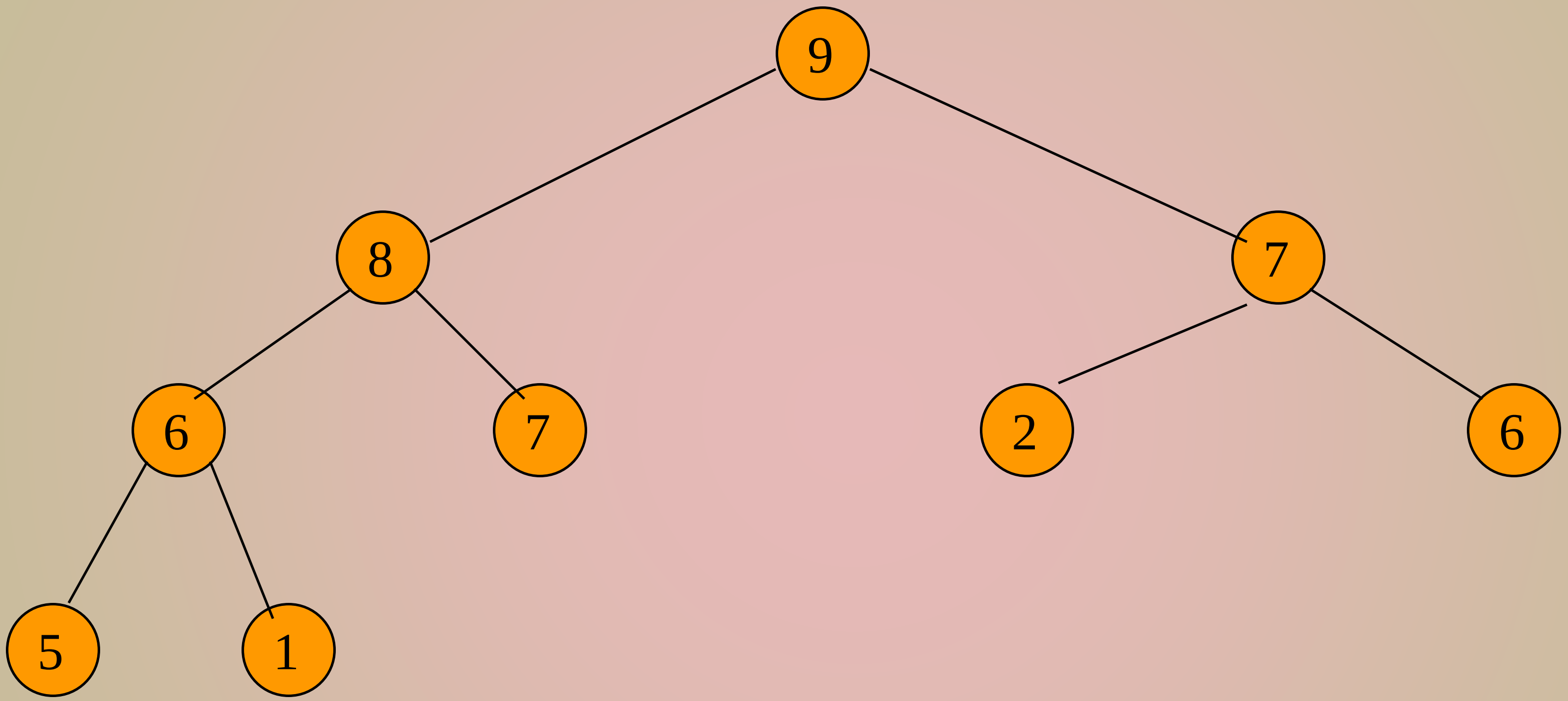
Complete binary tree with 9 nodes.

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.

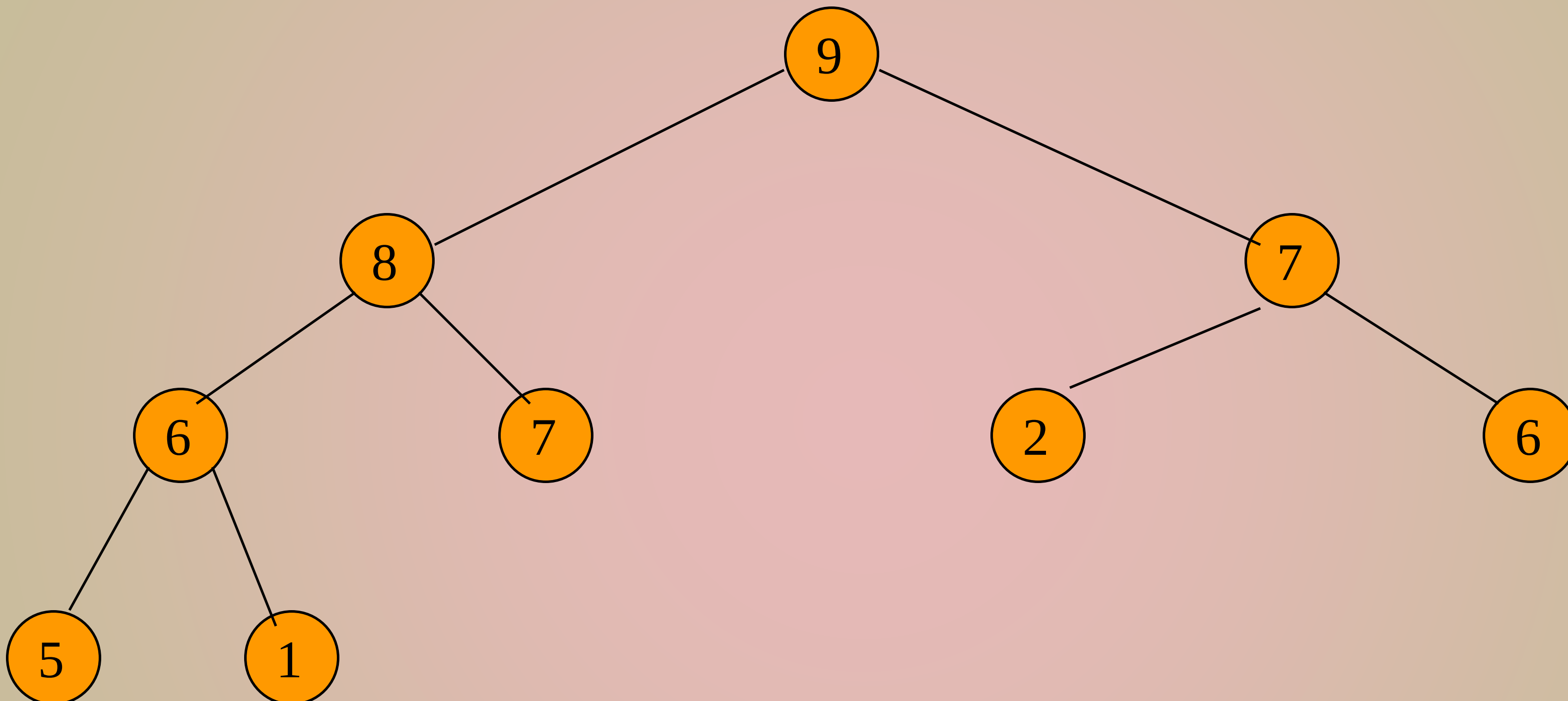
Heap Height

Since a heap is a complete binary tree, the height of an n node heap is ……………

Heap Height

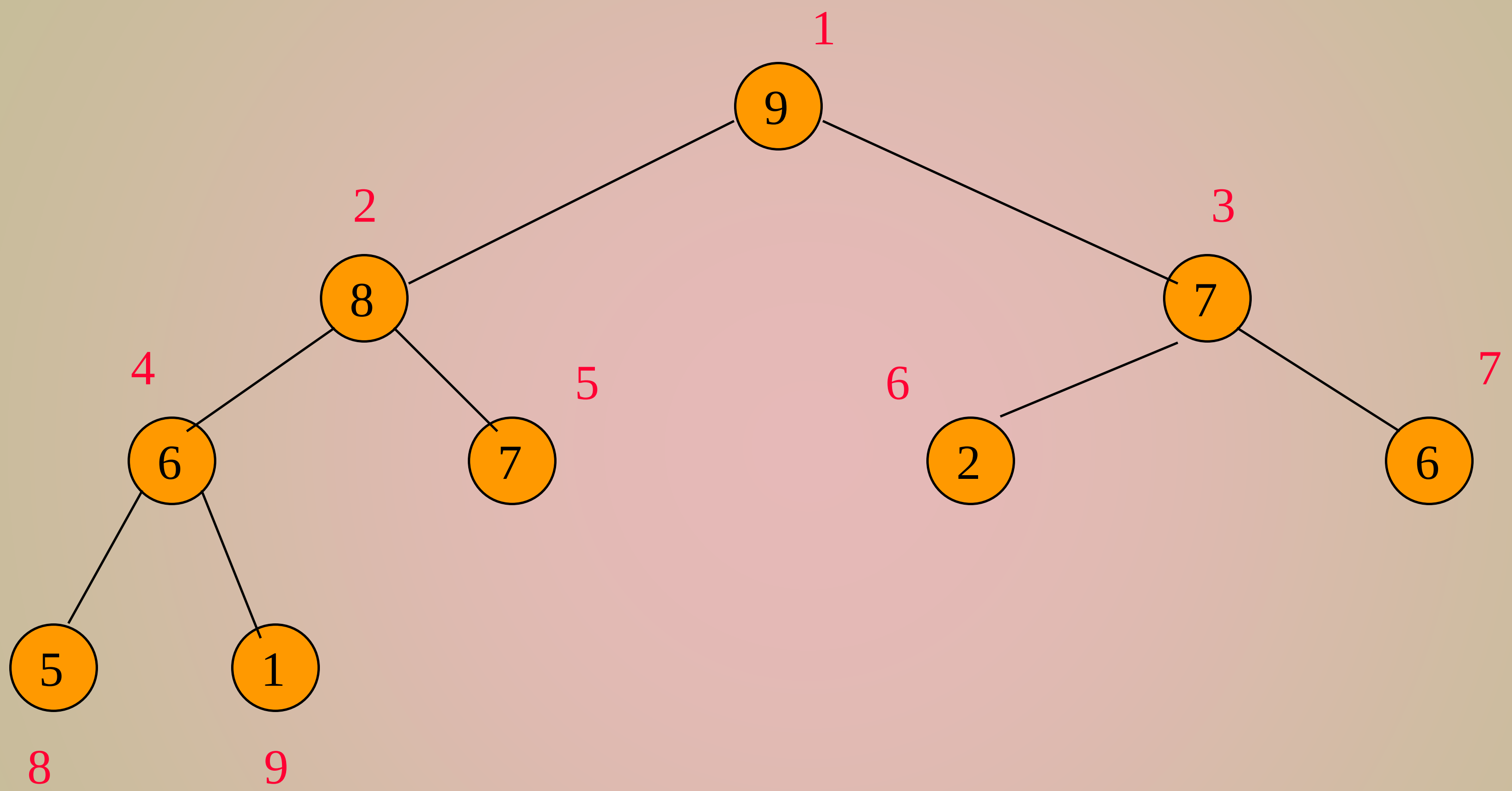
Since a heap is a complete binary tree, the height of an n node heap is $\lceil \log_2(n+1) \rceil$.

A Heap Is Efficiently Represented As An Array

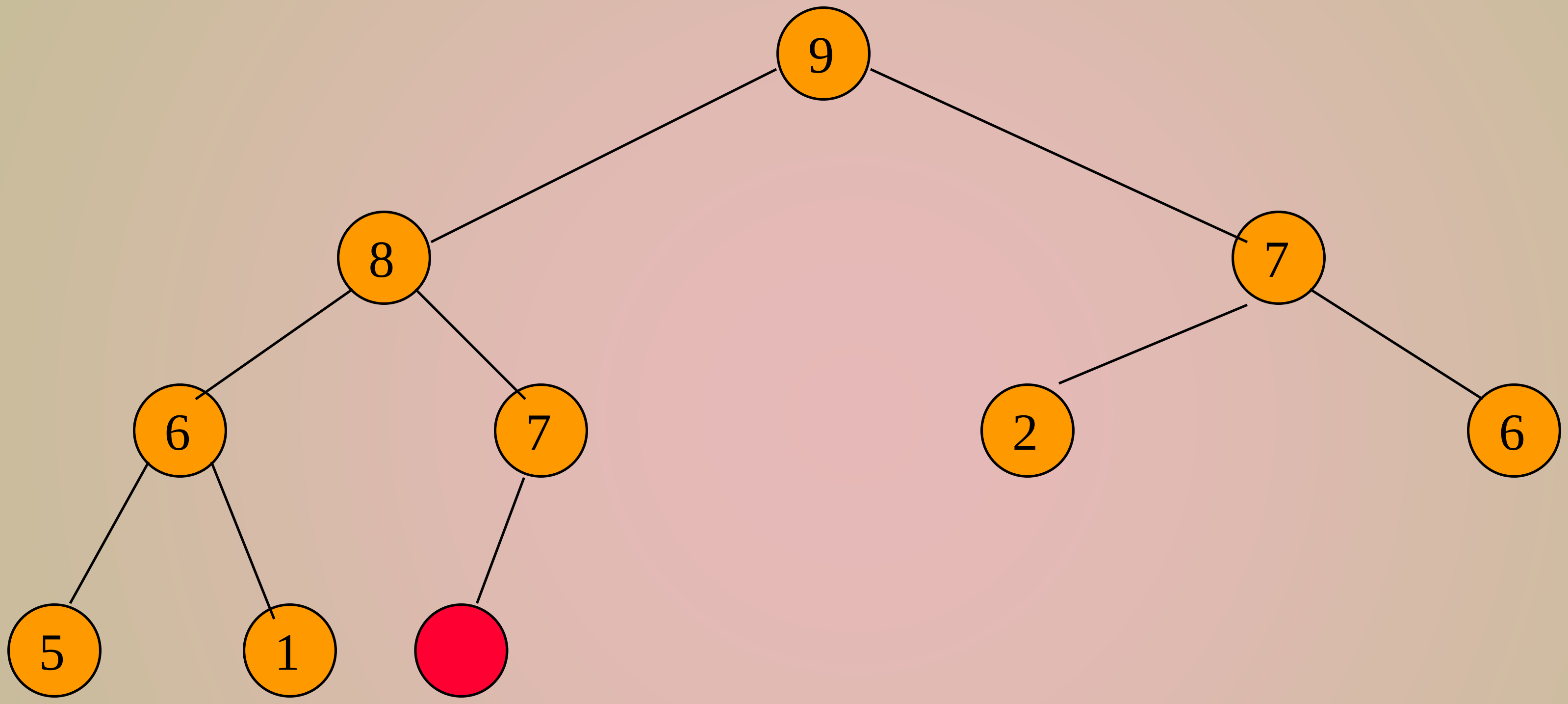


	9	8	7	6	7	2	6	5	1							
0	1	2	3	4	5	6	7	8	9	10						

Moving Up And Down A Heap

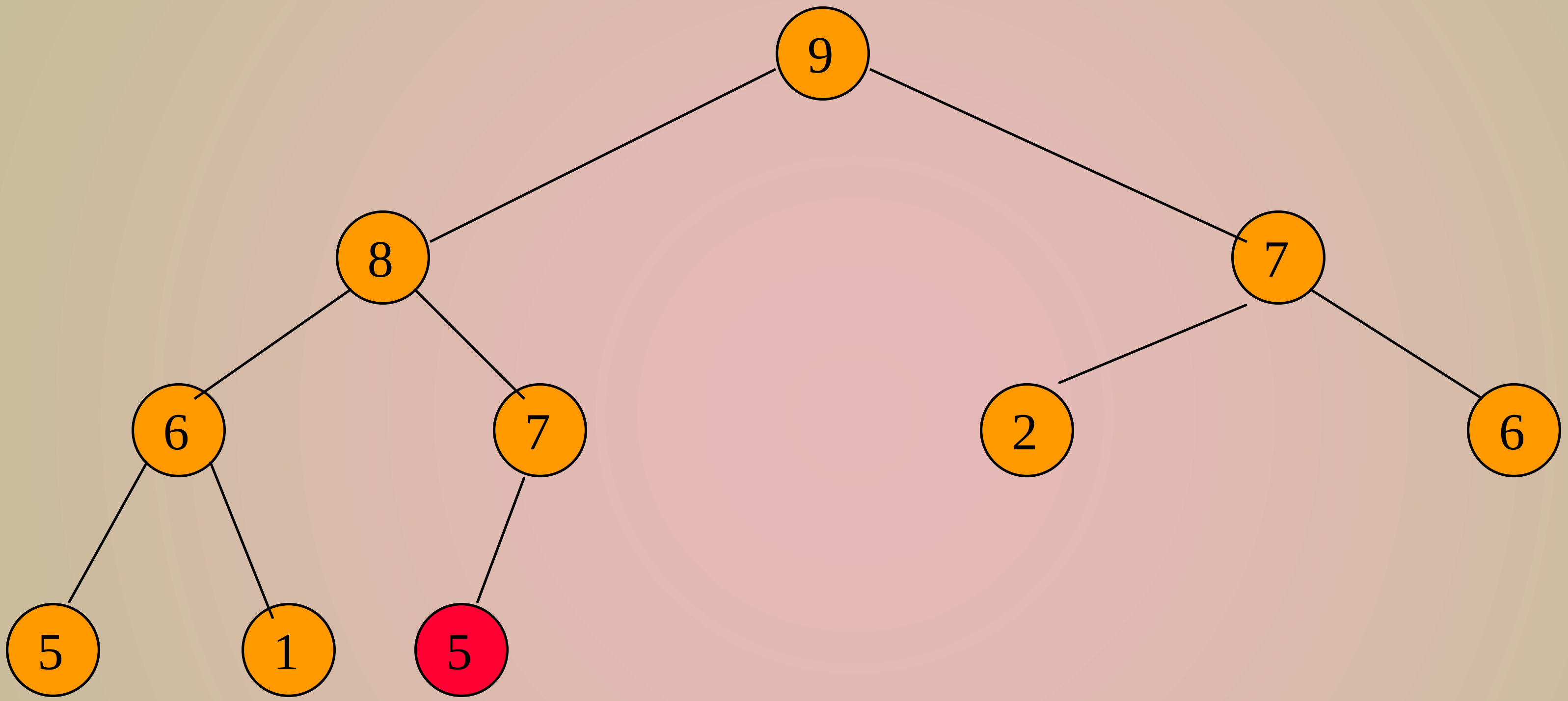


Inserting An Element Into A Max Heap



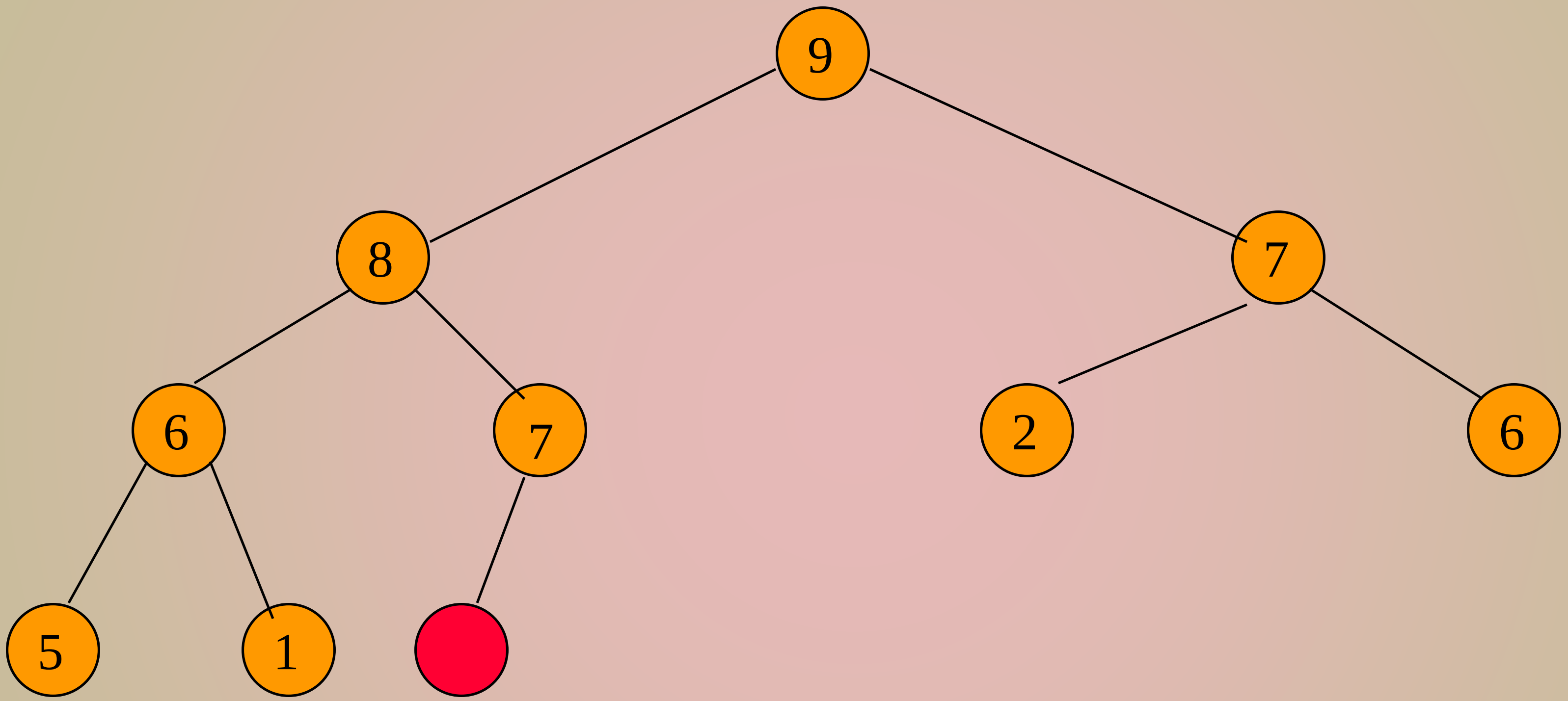
Complete binary tree with 10 nodes.

Inserting An Element Into A Max Heap



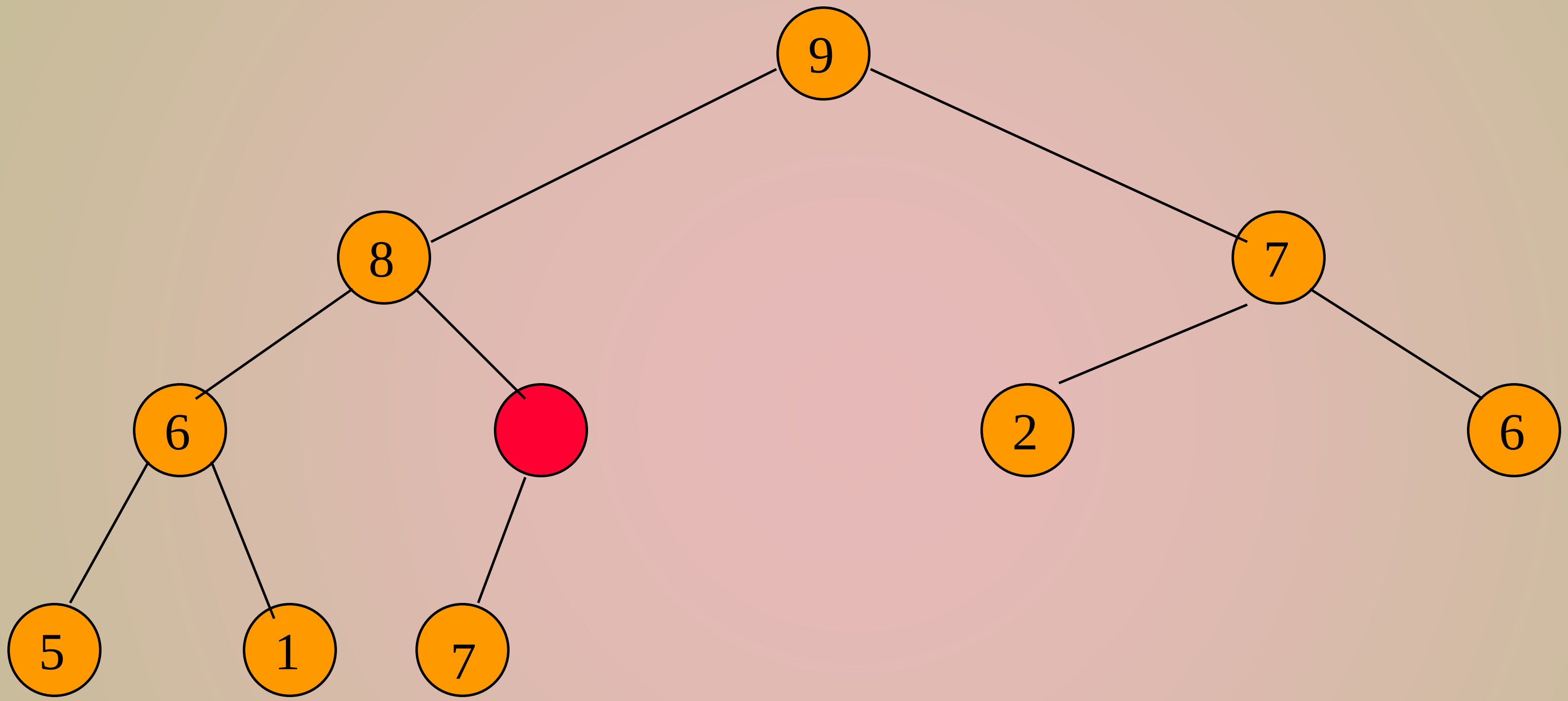
New element is 5.

Inserting An Element Into A Max Heap



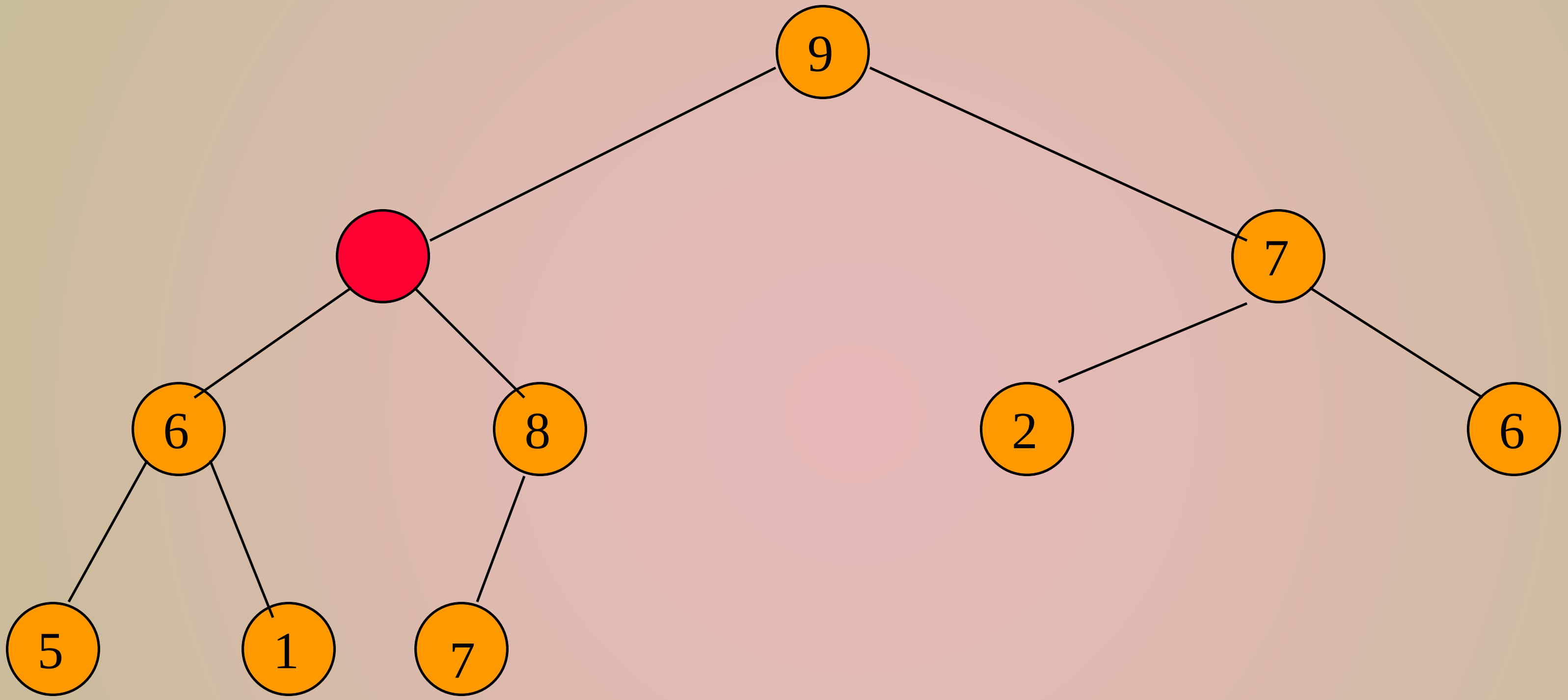
New element is 20.

Inserting An Element Into A Max Heap



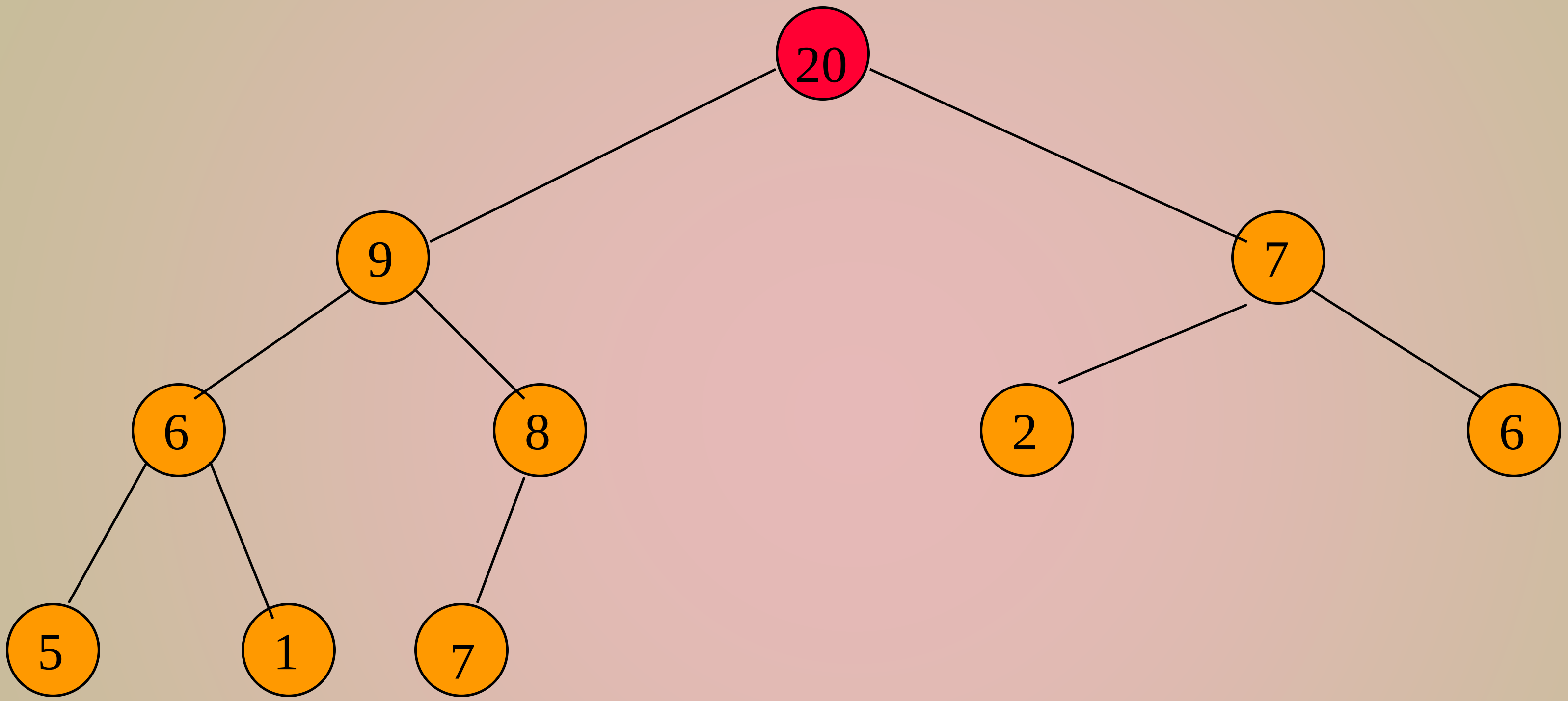
New element is 20.

Inserting An Element Into A Max Heap



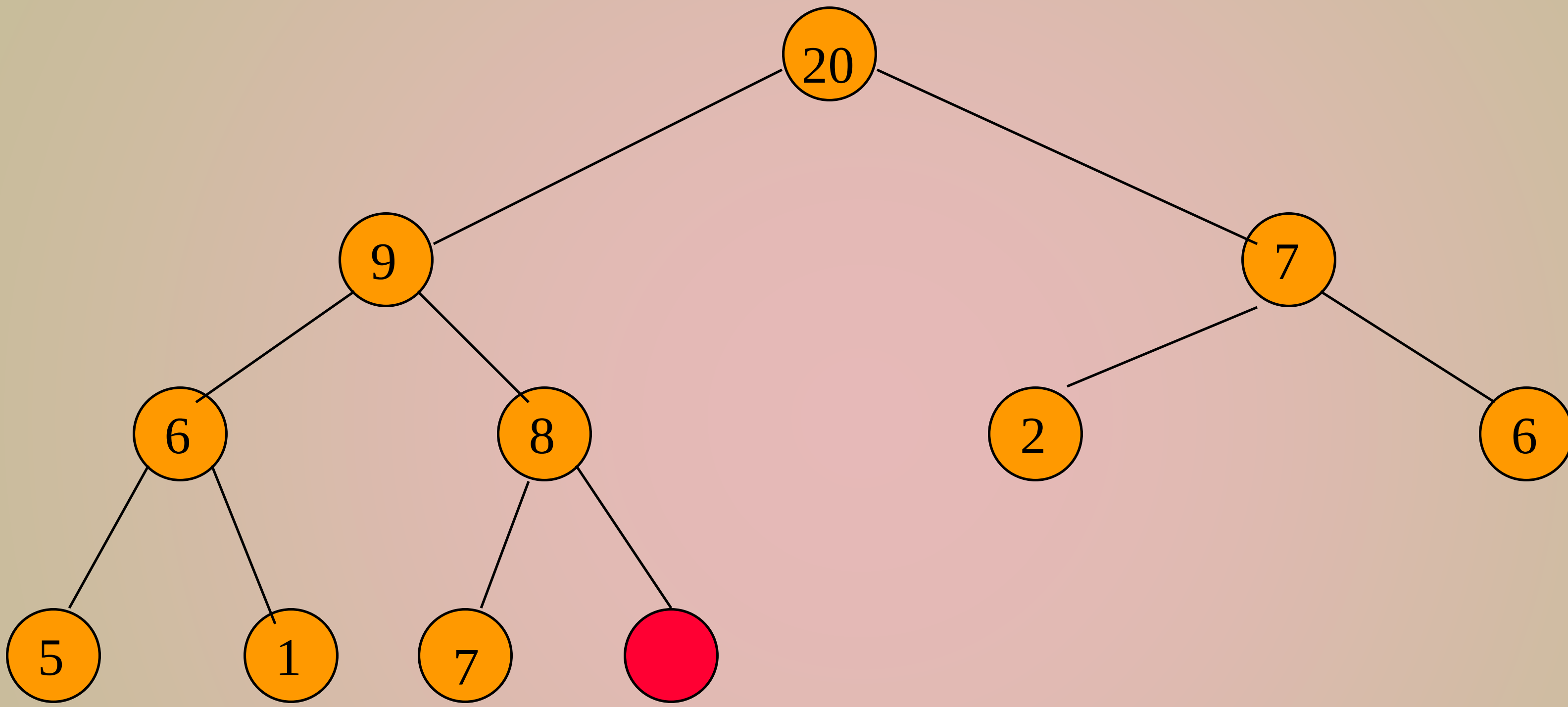
New element is 20.

Inserting An Element Into A Max Heap



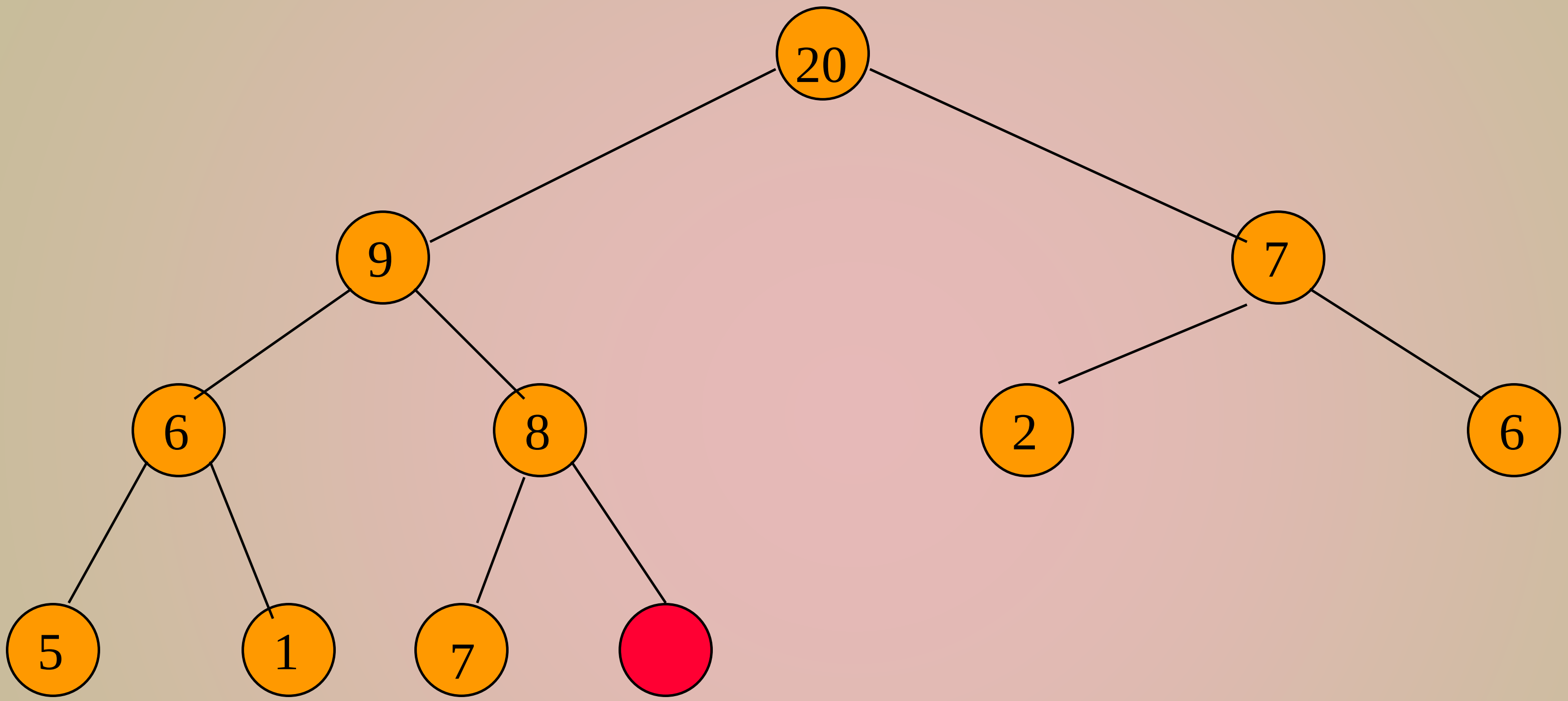
New element is 20.

Inserting An Element Into A Max Heap



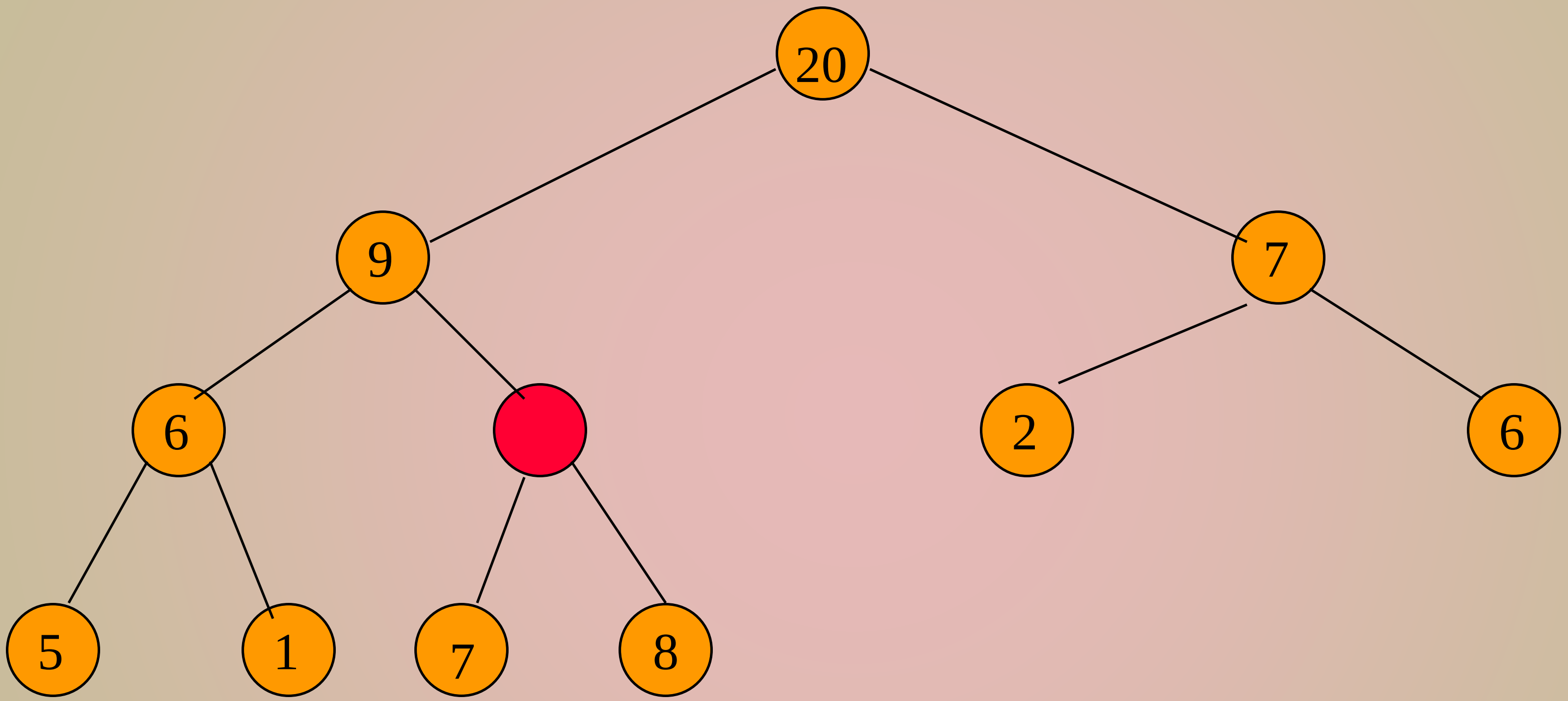
Complete binary tree with 11 nodes.

Inserting An Element Into A Max Heap



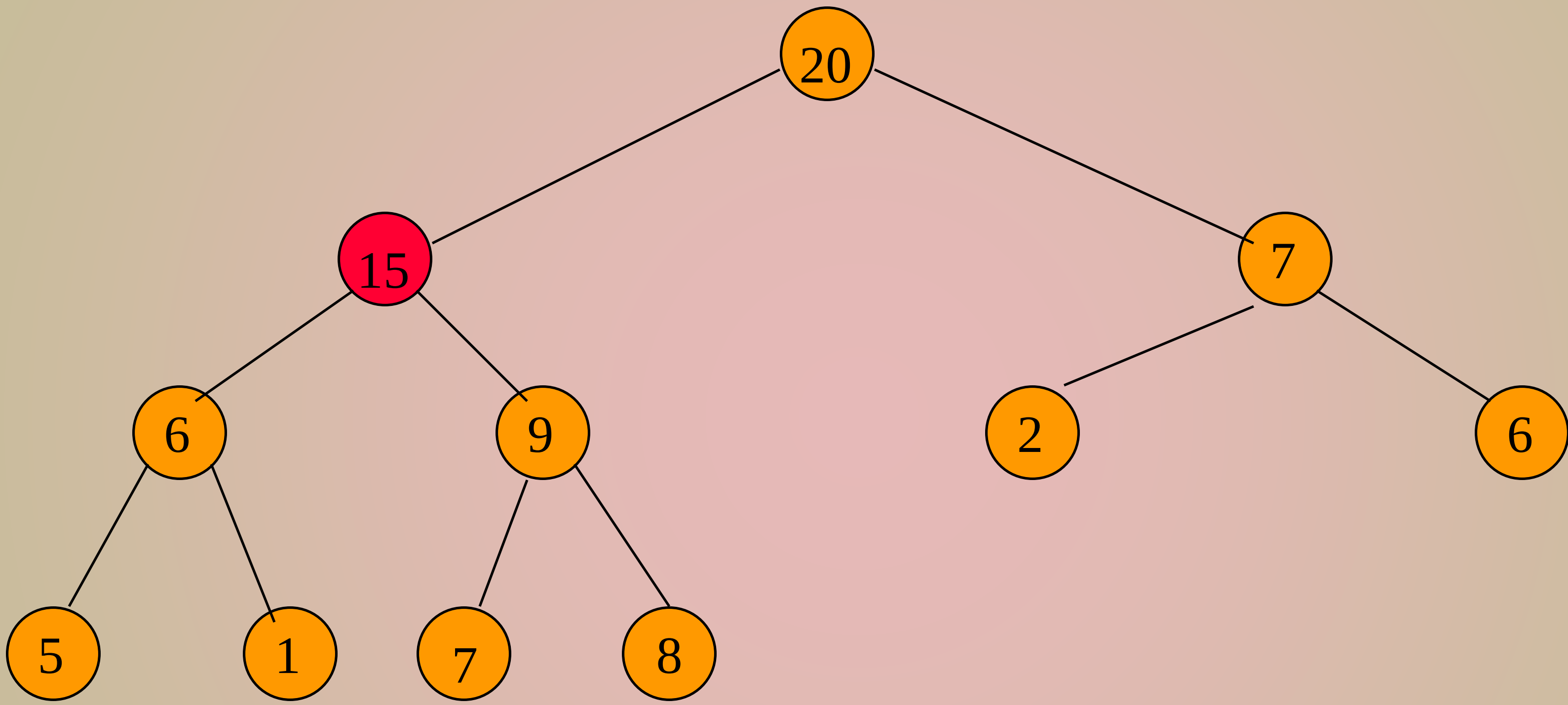
New element is 15.

Inserting An Element Into A Max Heap



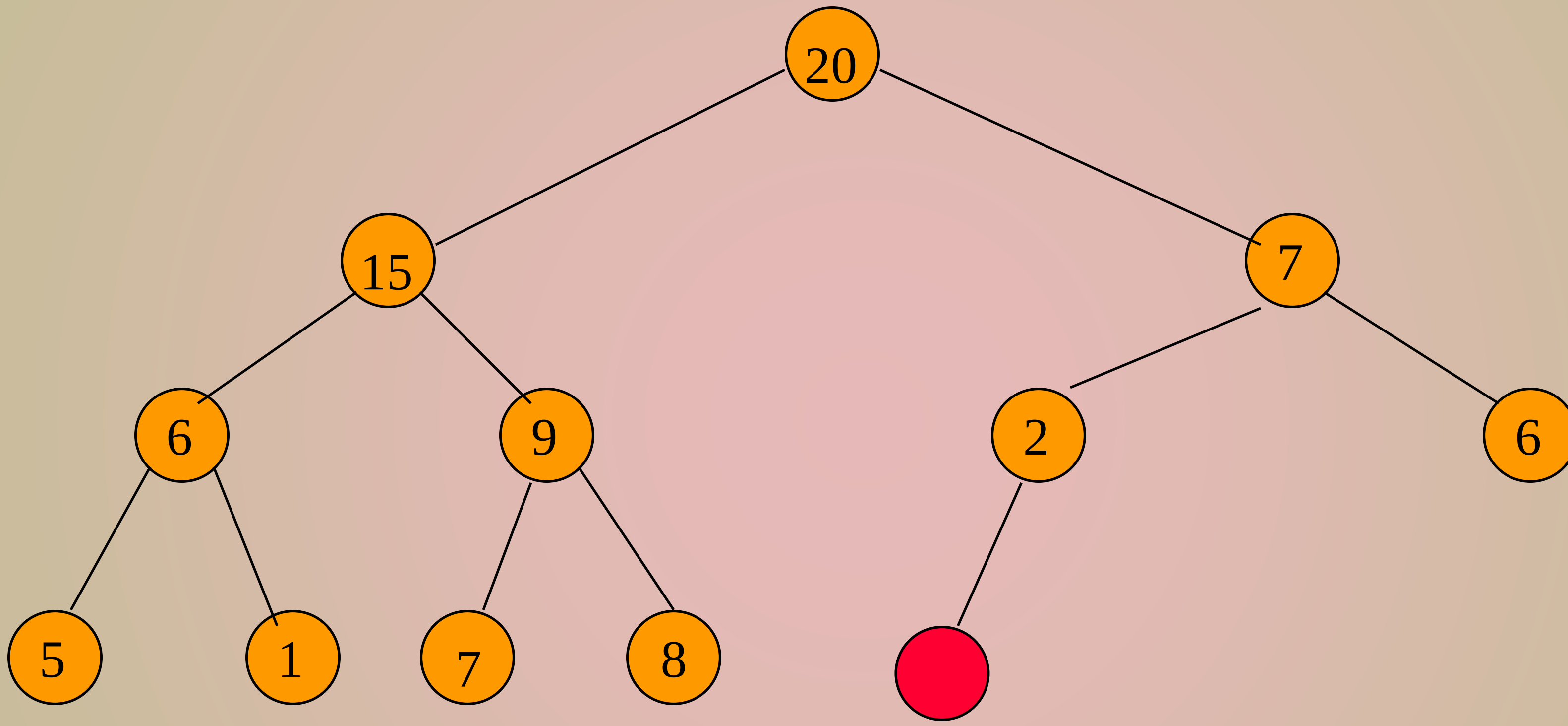
New element is 15.

Inserting An Element Into A Max Heap



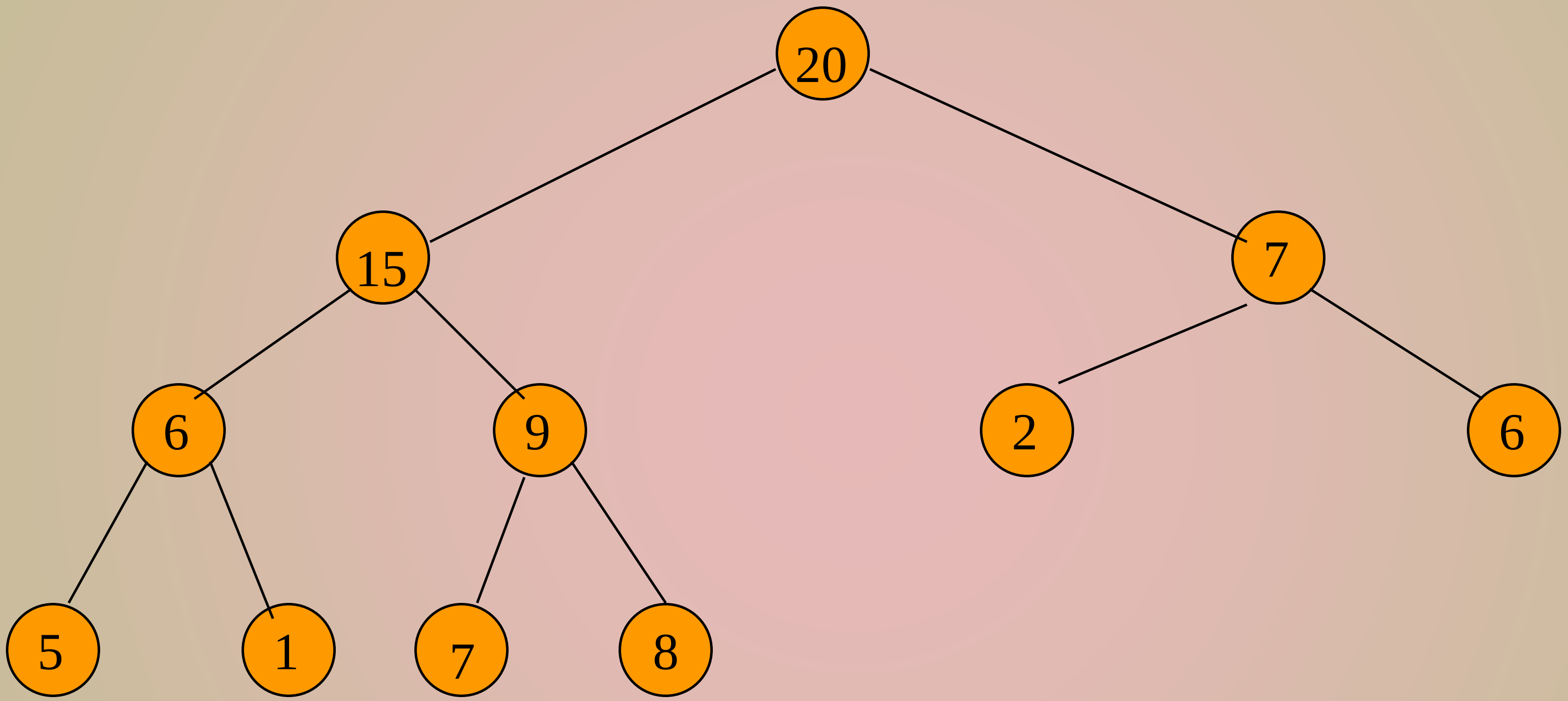
New element is 15.

Complexity Of Insert



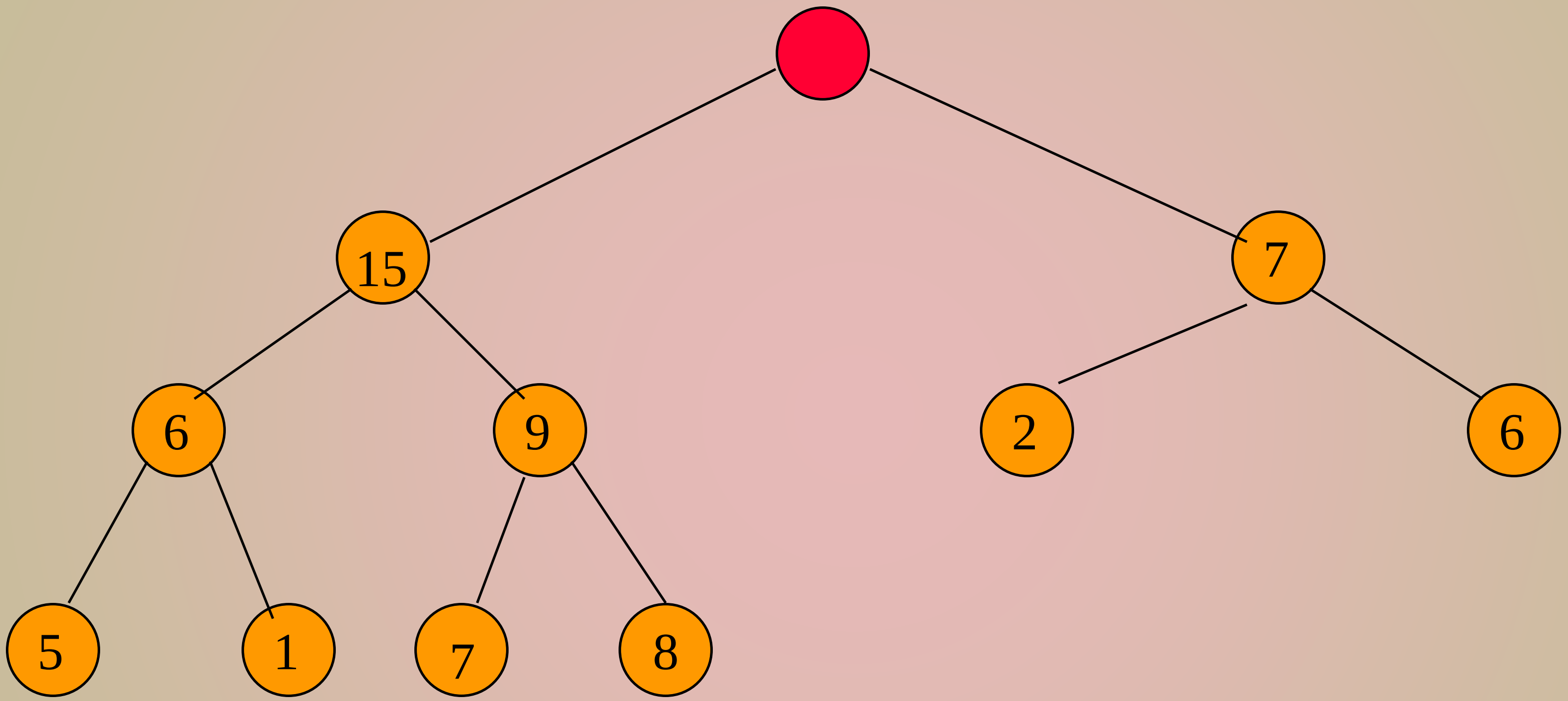
Complexity is O(log n), where n is heap size.

Removing The Max Element



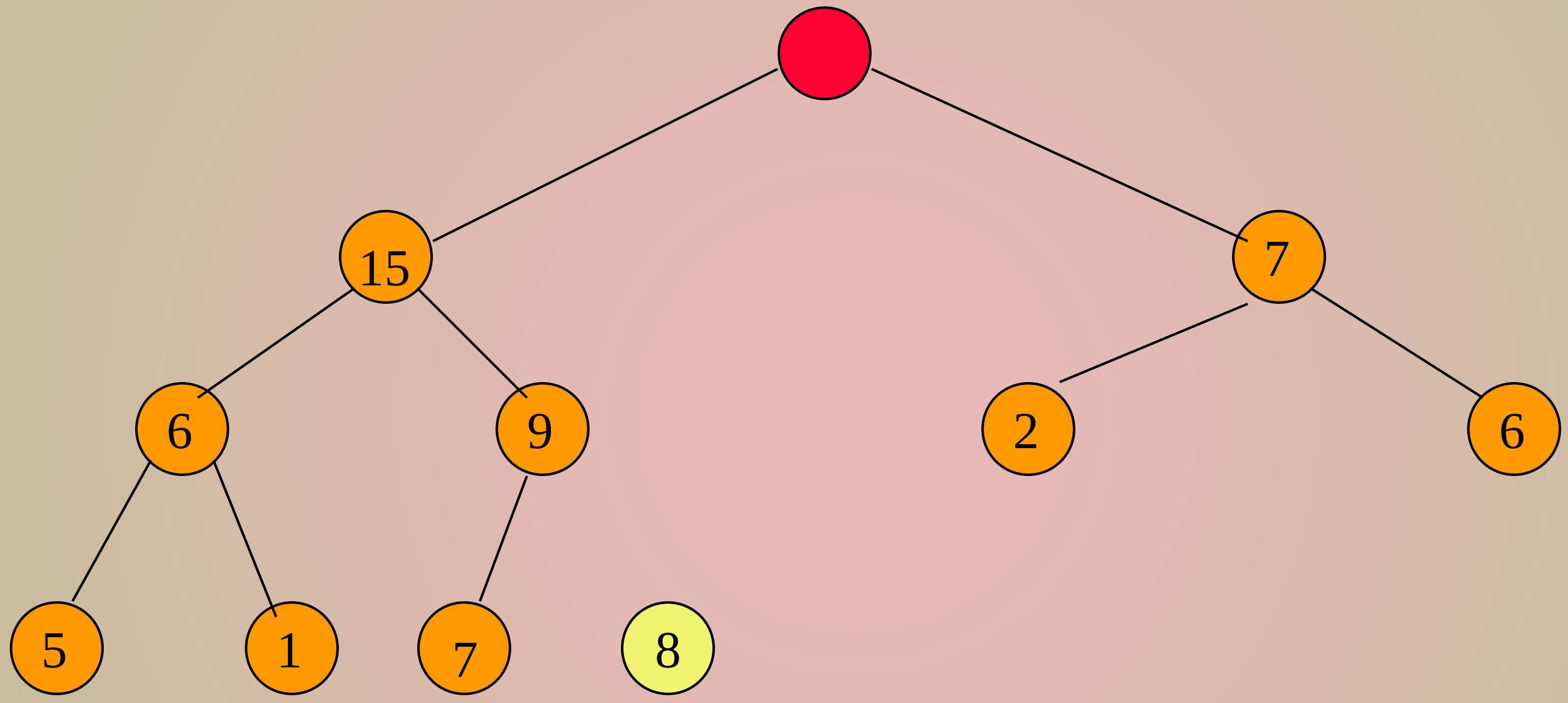
Max element is in the root.

Removing The Max Element



After max element is removed.

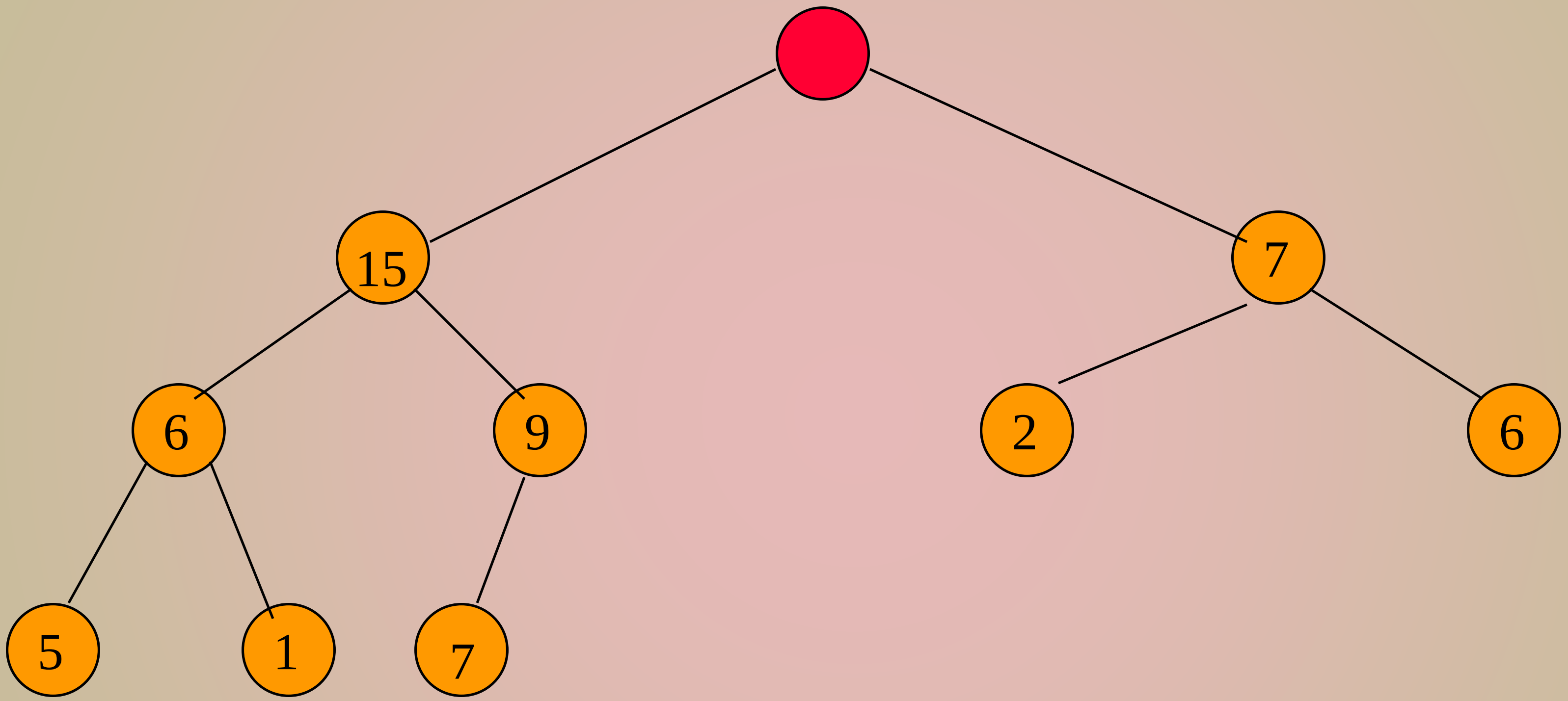
Removing The Max Element



Heap with 10 nodes.

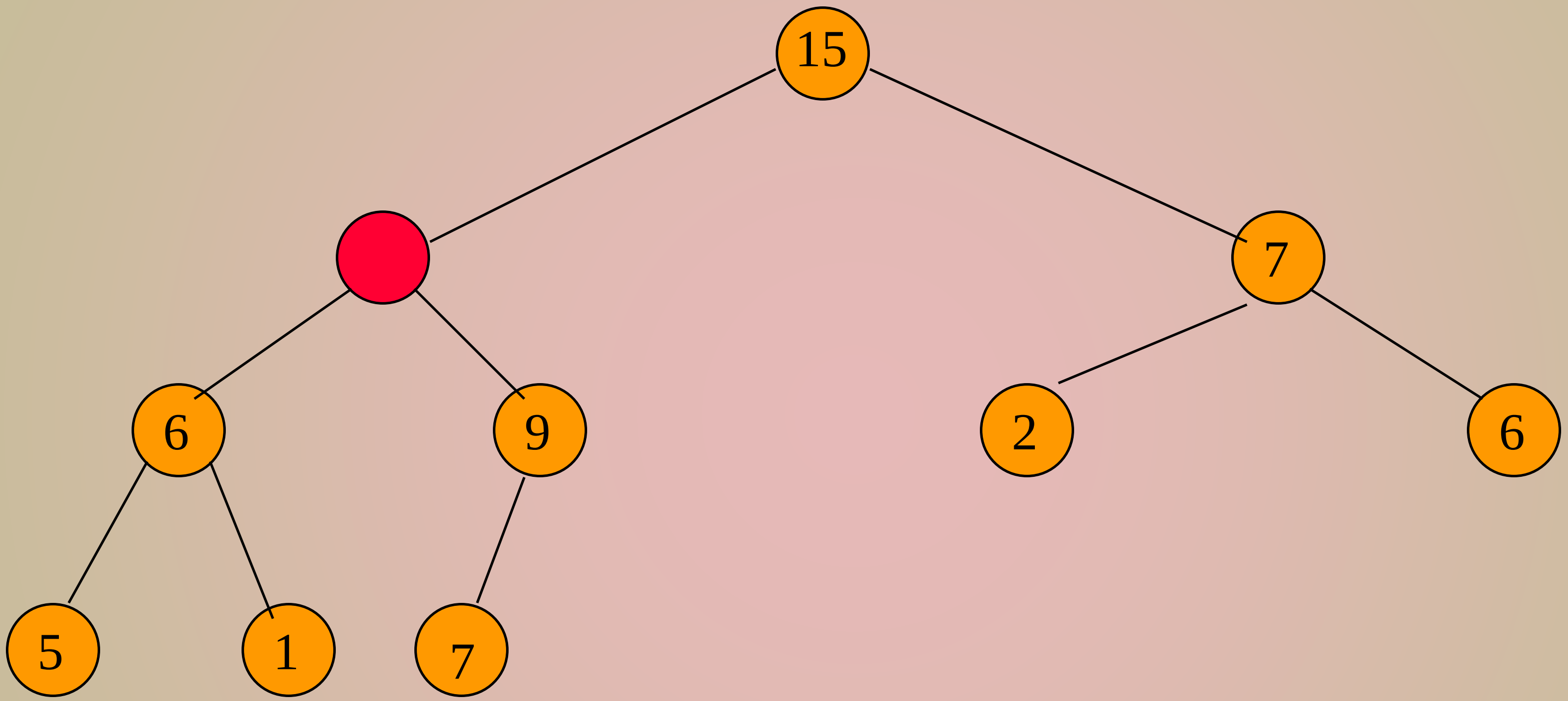
Reinsert 8 into the heap.

Removing The Max Element



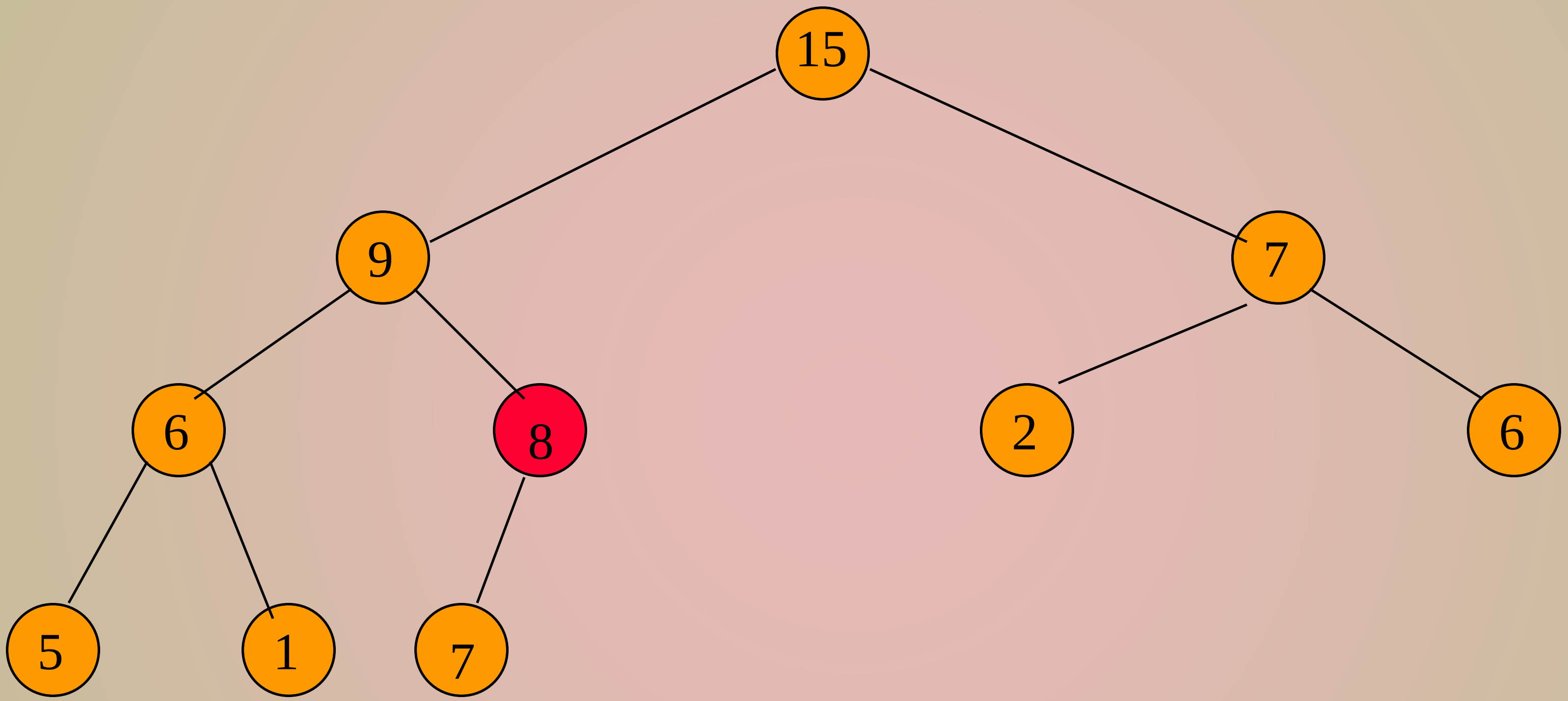
Reinsert 8 into the heap.

Removing The Max Element



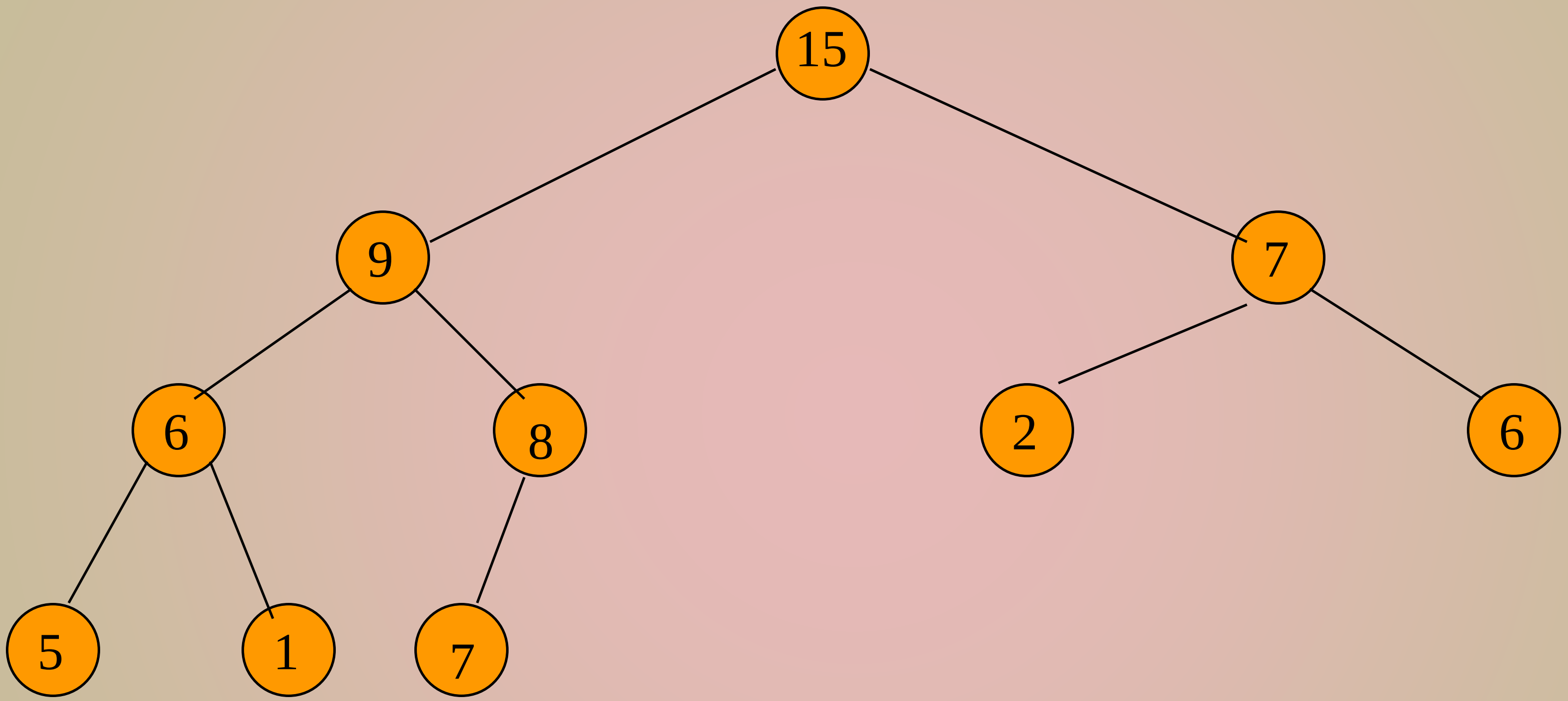
Reinsert 8 into the heap.

Removing The Max Element



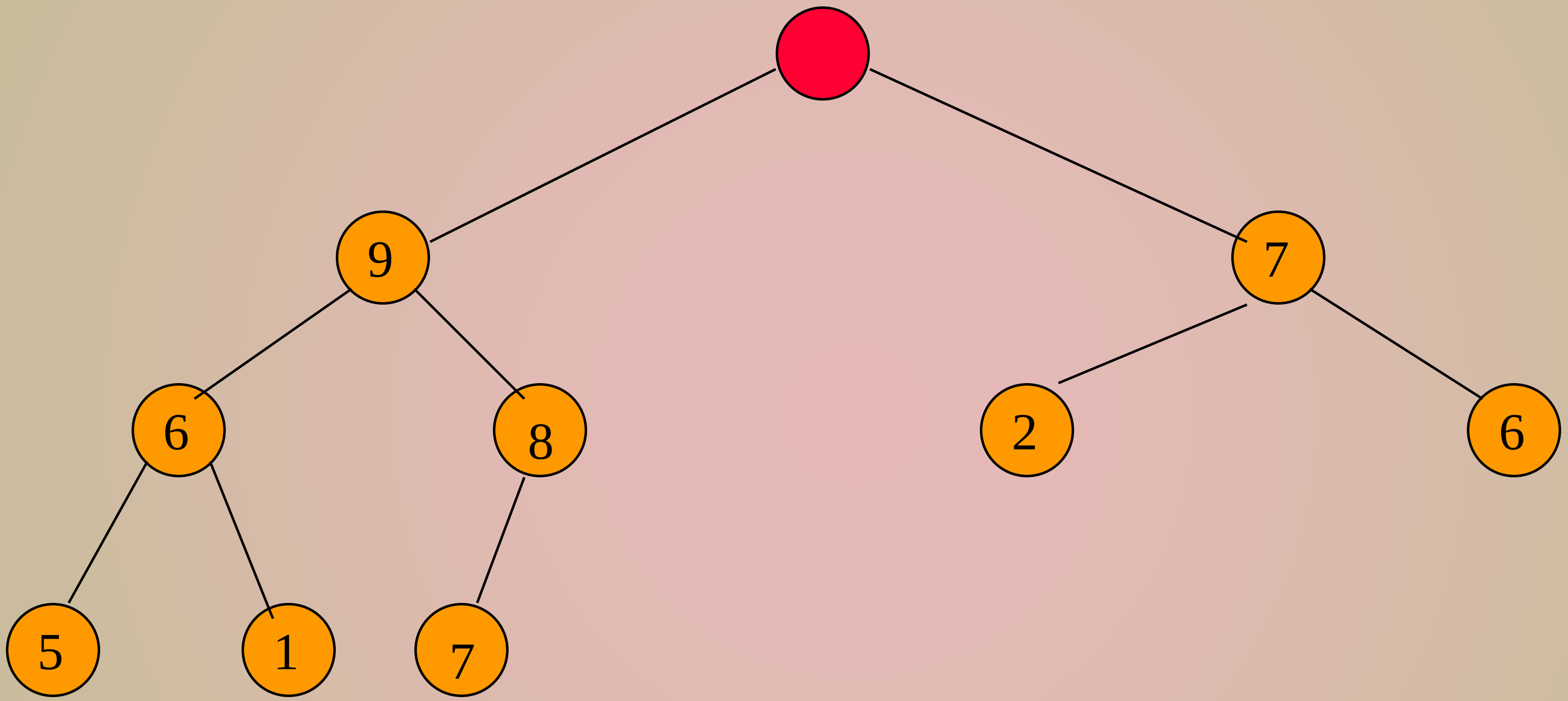
Reinsert 8 into the heap.

Removing The Max Element



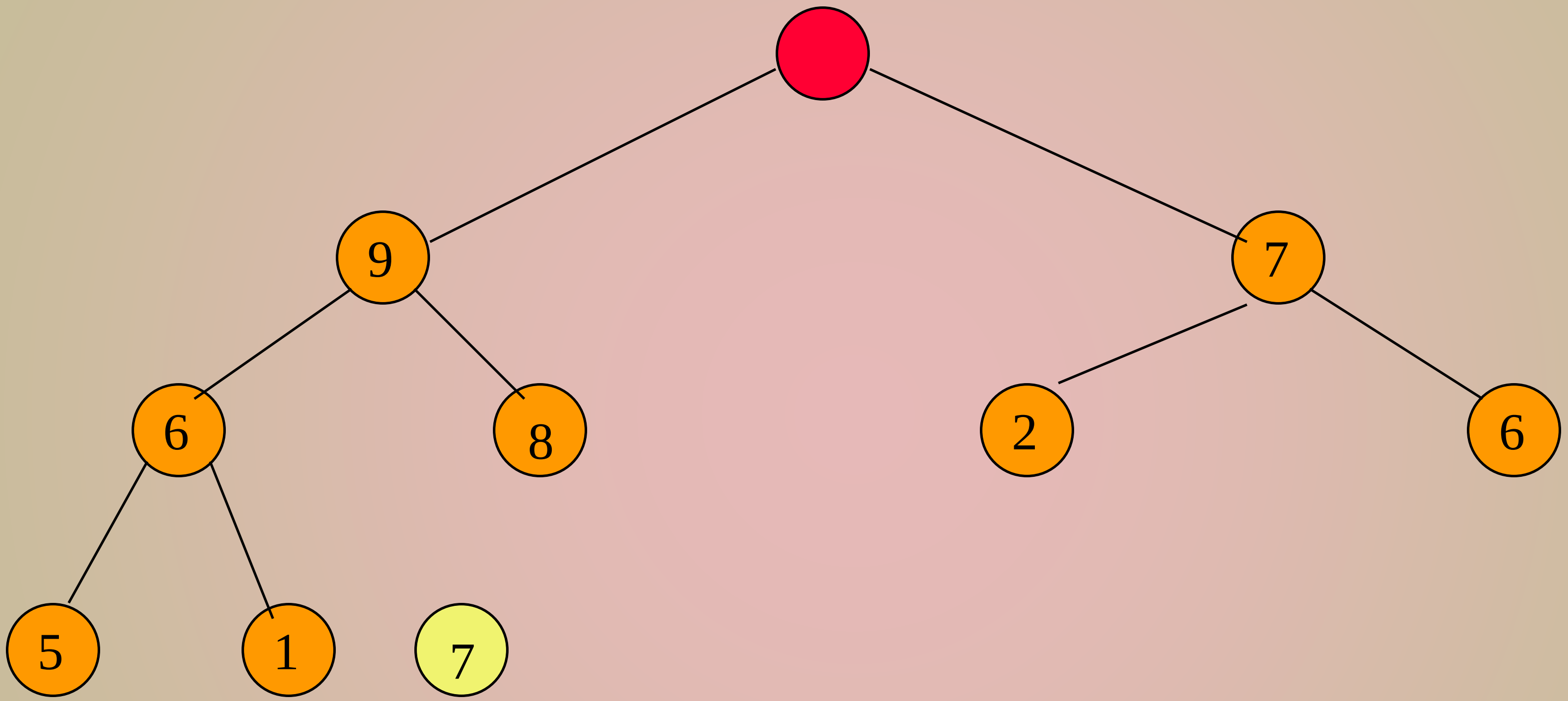
Max element is 15.

Removing The Max Element



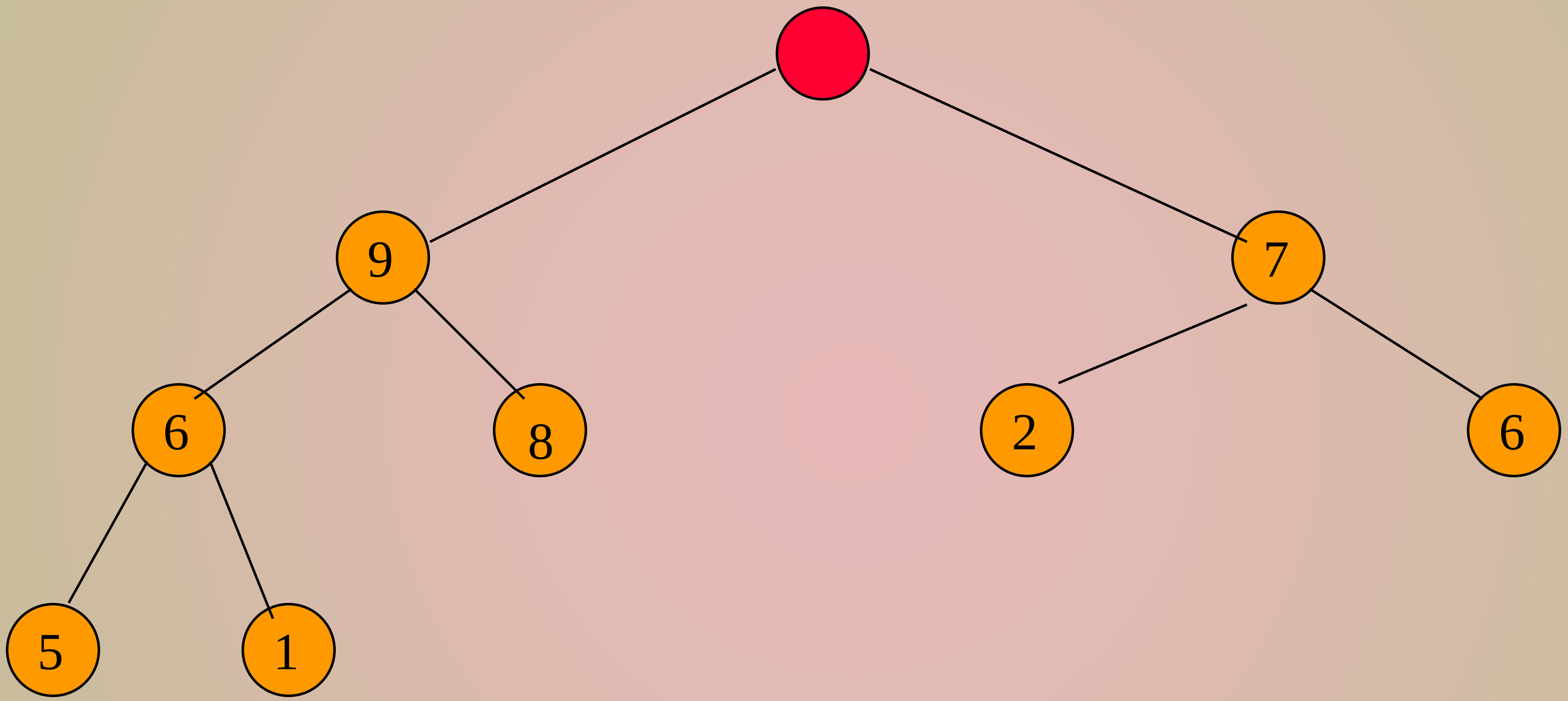
After max element is removed.

Removing The Max Element



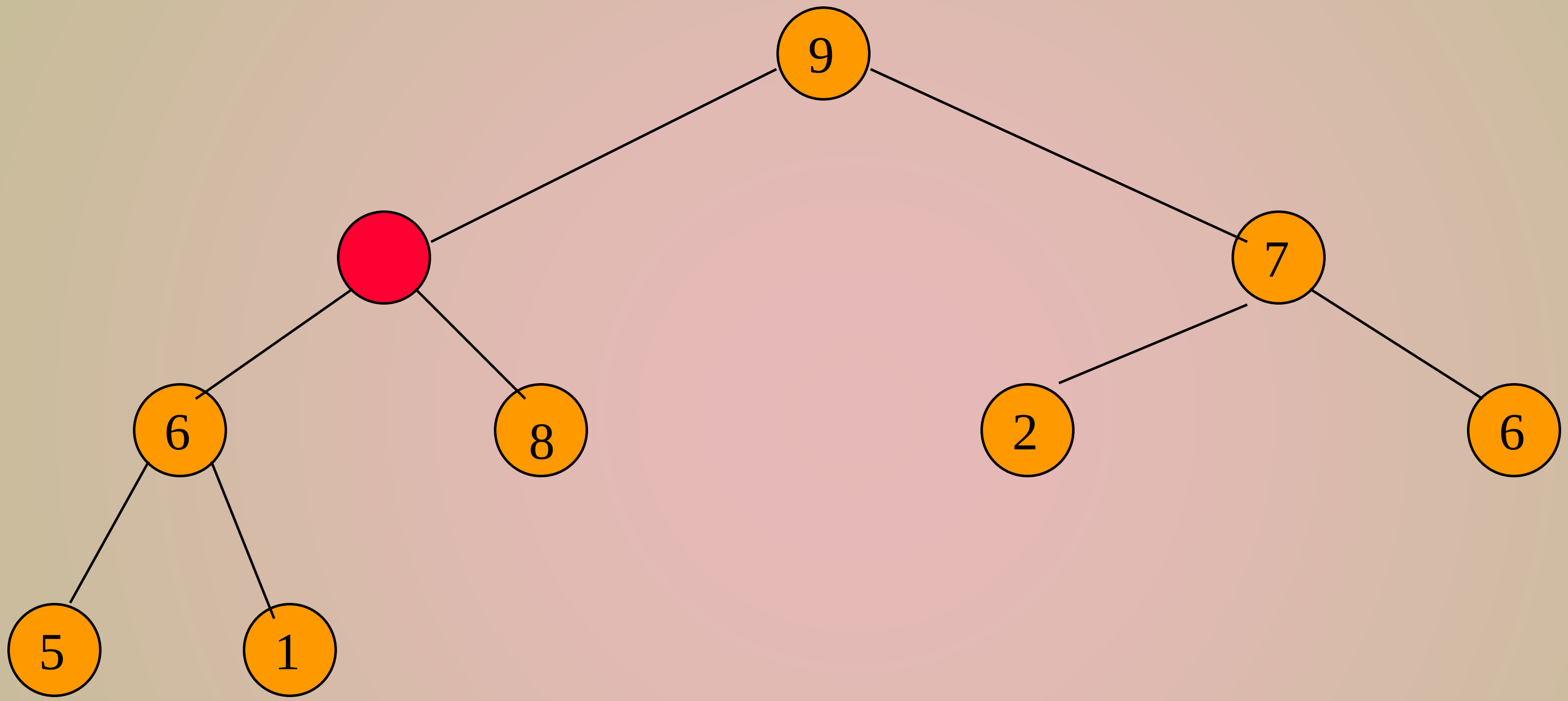
Heap with 9 nodes.

Removing The Max Element



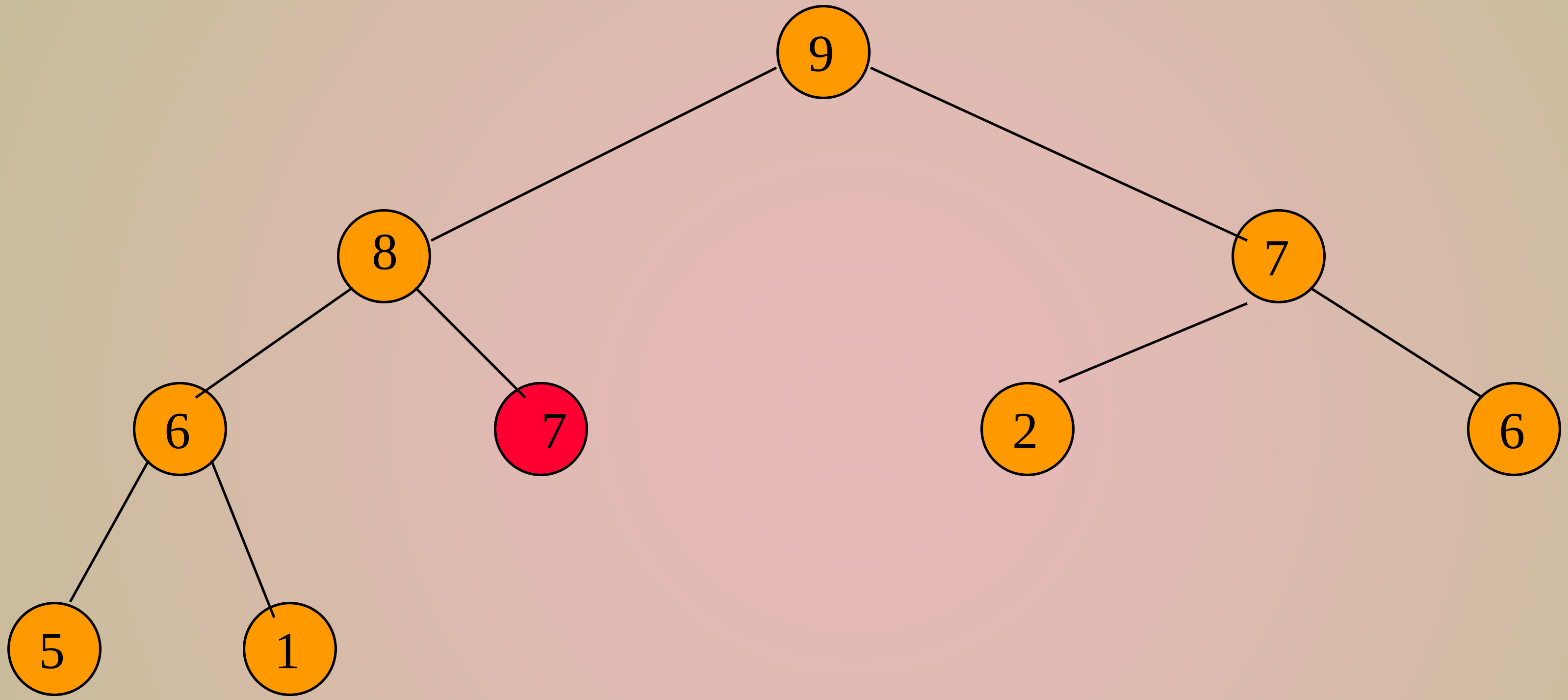
Reinsert 7.

Removing The Max Element



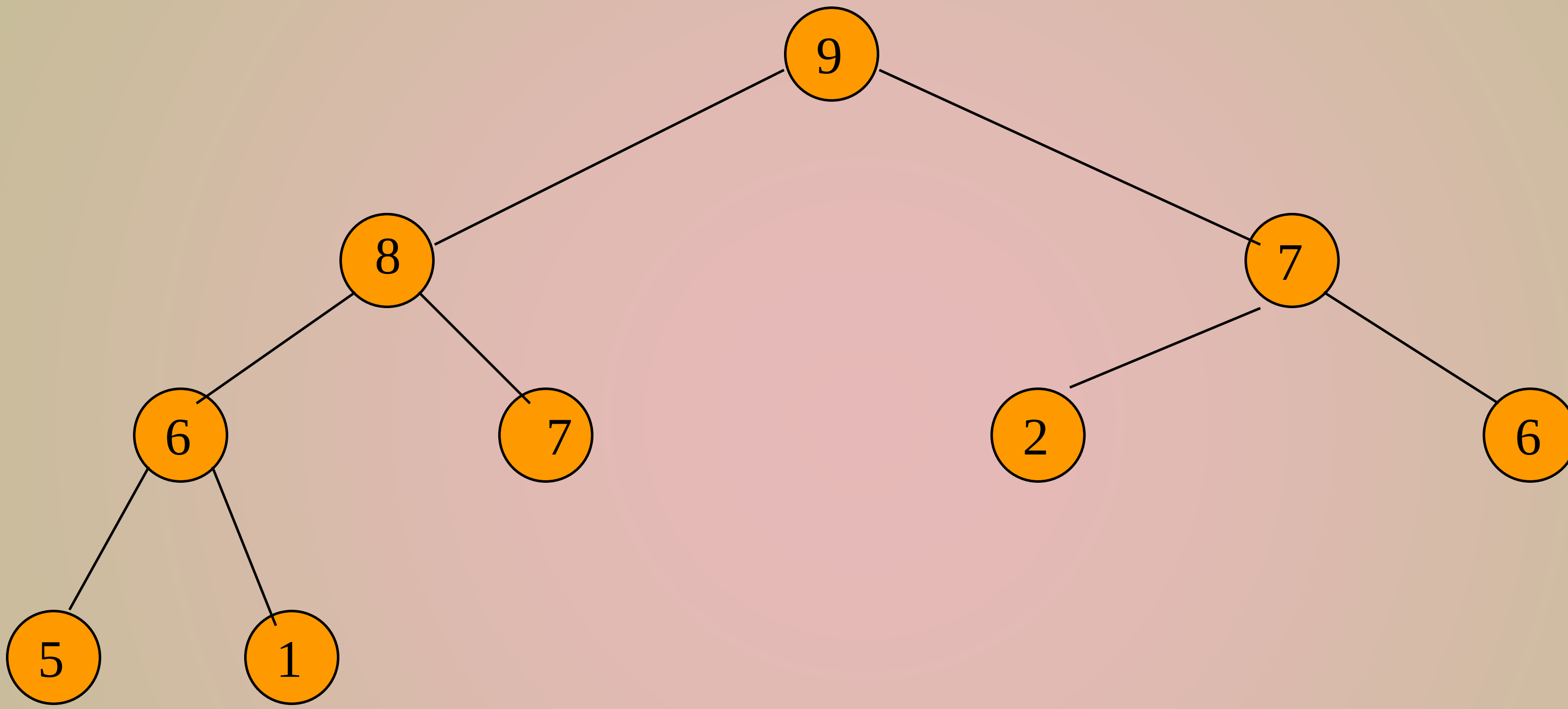
Reinsert 7.

Removing The Max Element



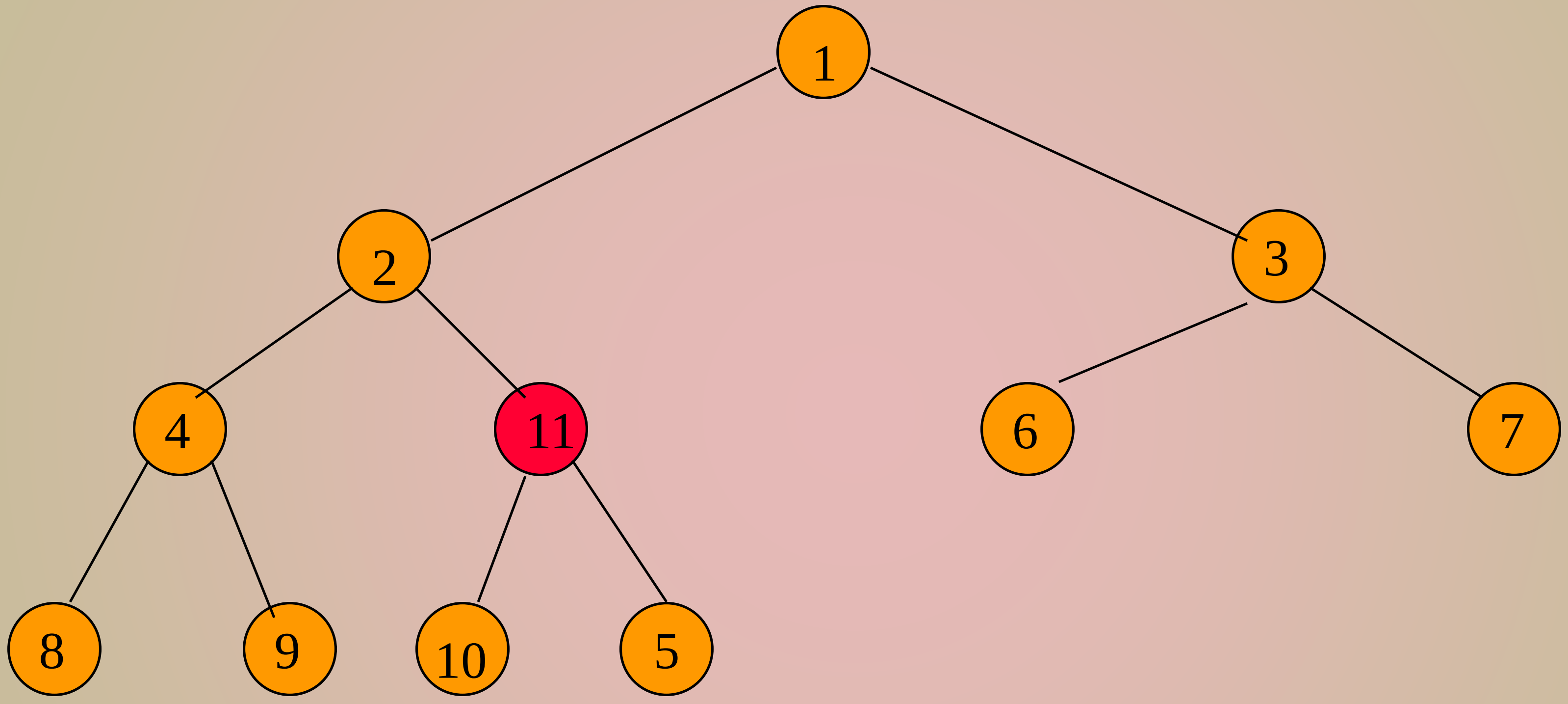
Reinsert 7.

Complexity Of Remove Max Element



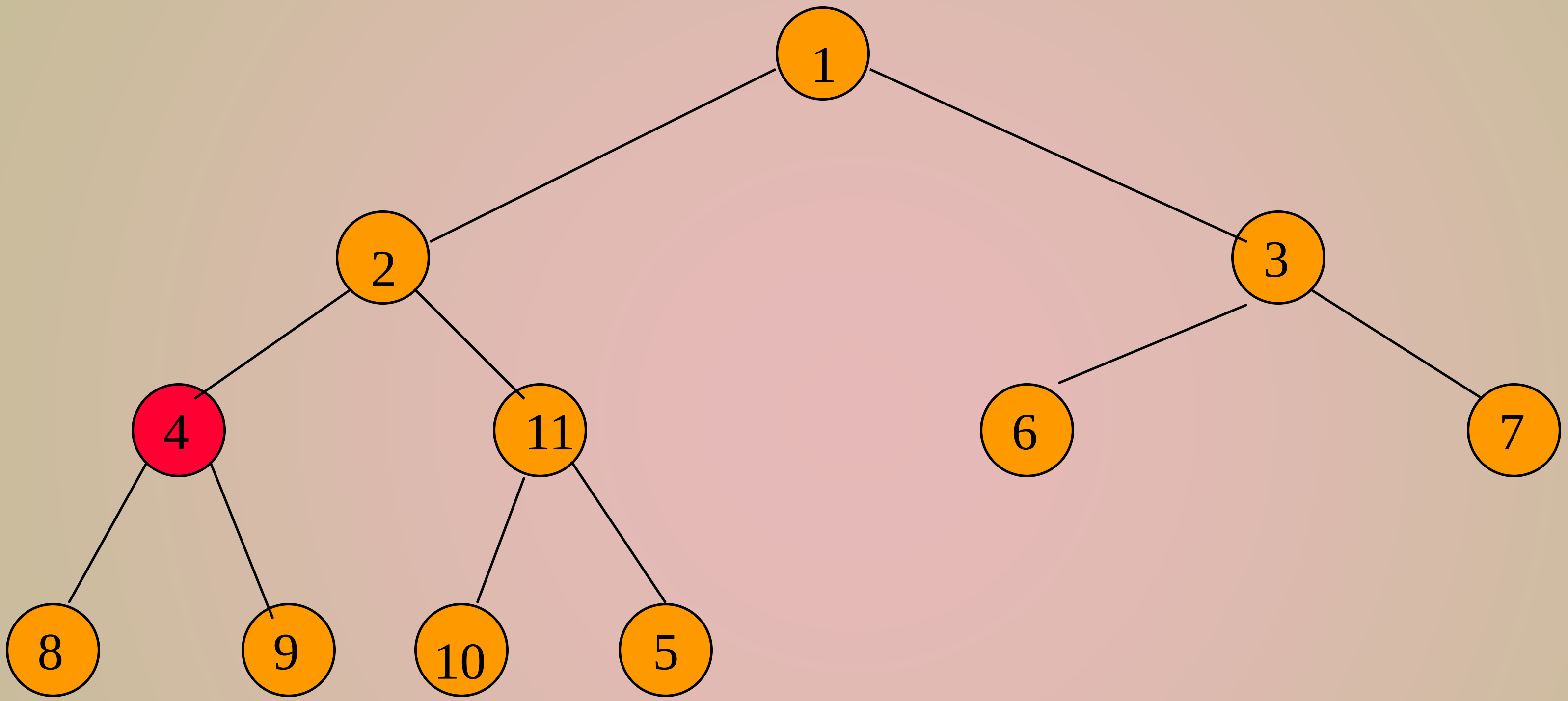
Complexity is O(log n).

Initializing A Max Heap

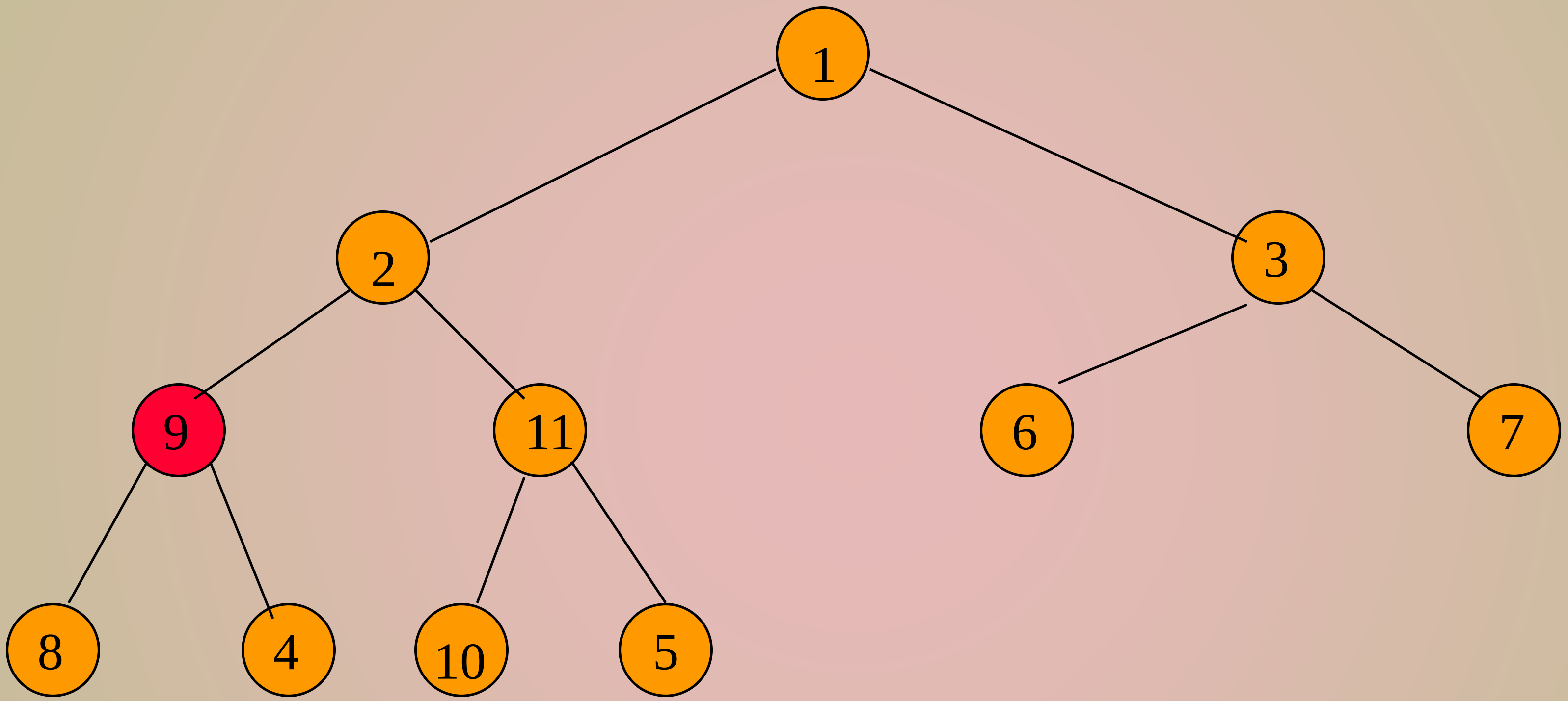


Move to next lower array position.

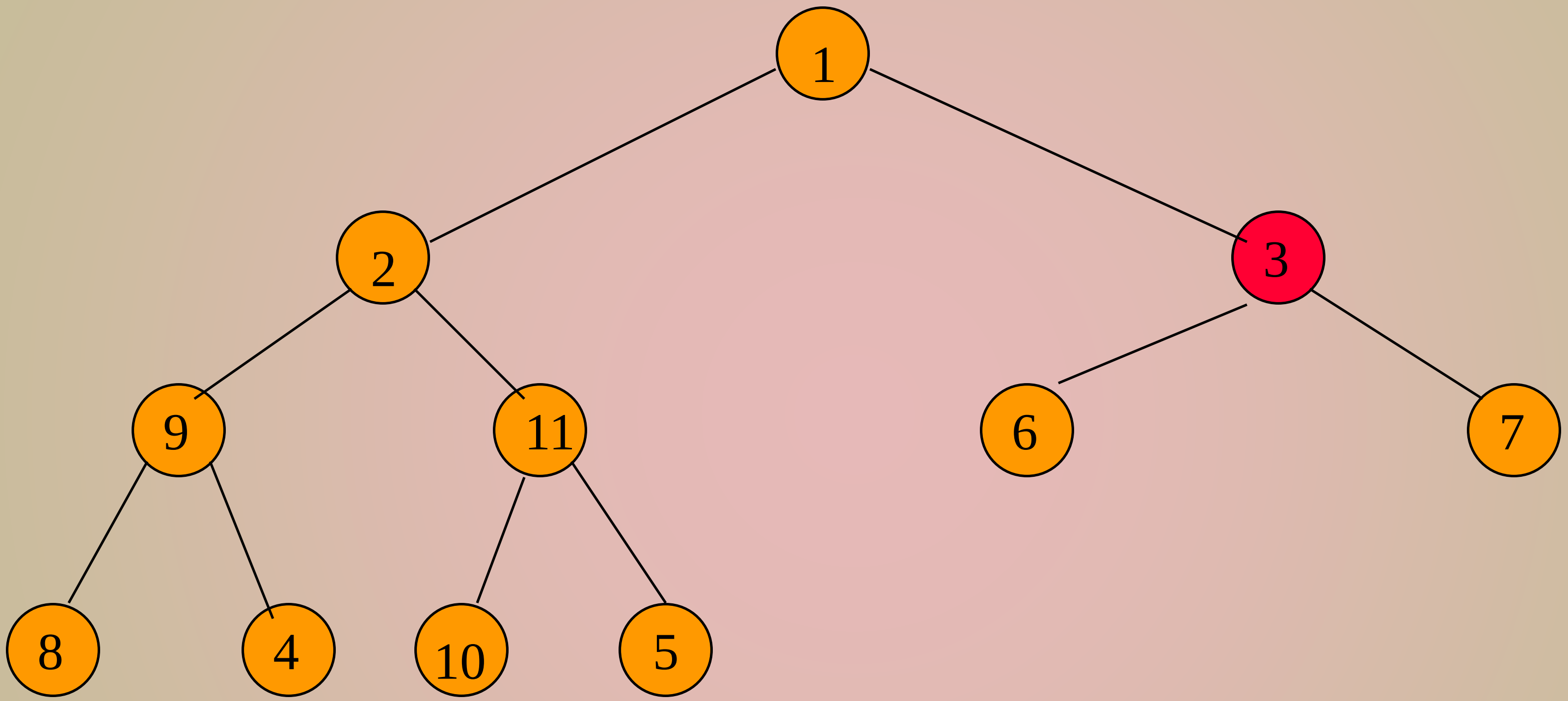
Initializing A Max Heap



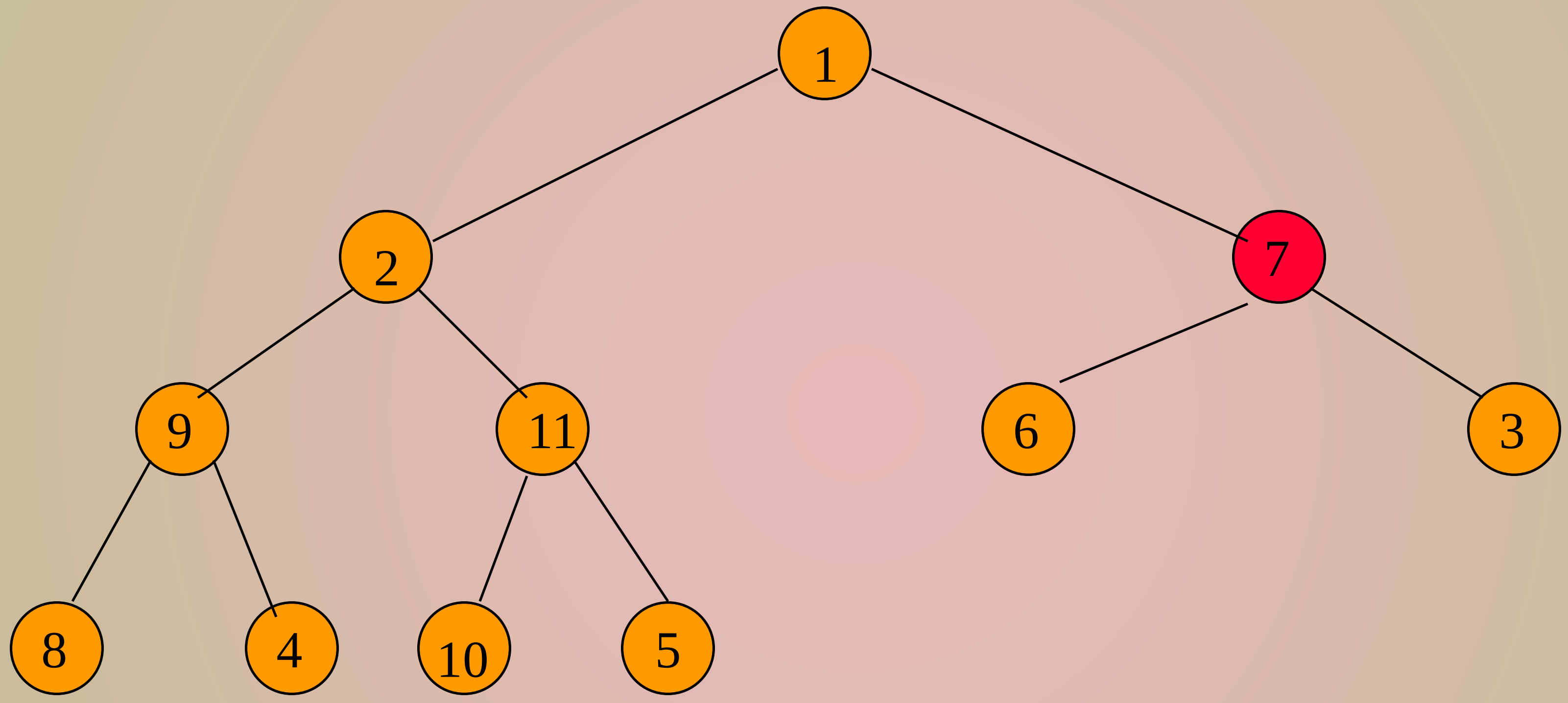
Initializing A Max Heap



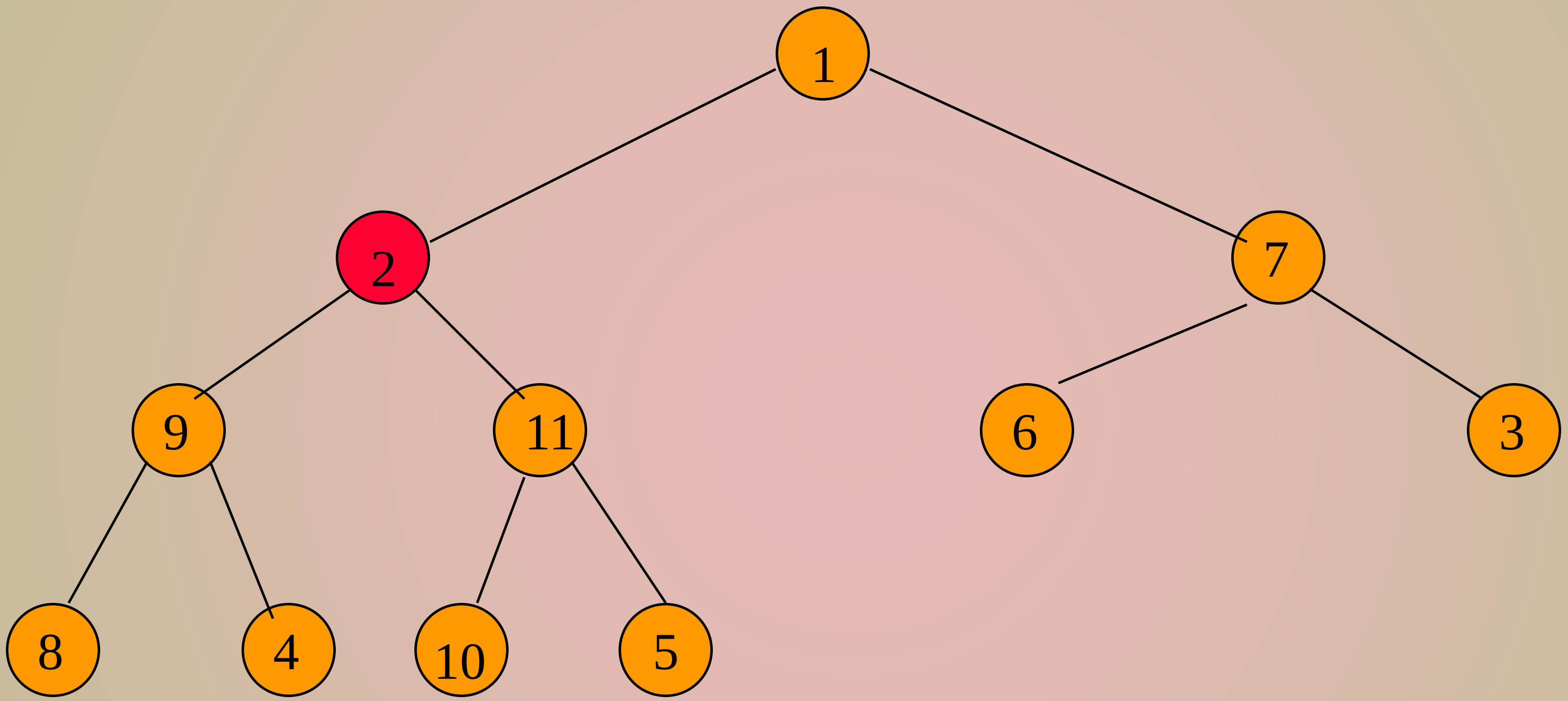
Initializing A Max Heap



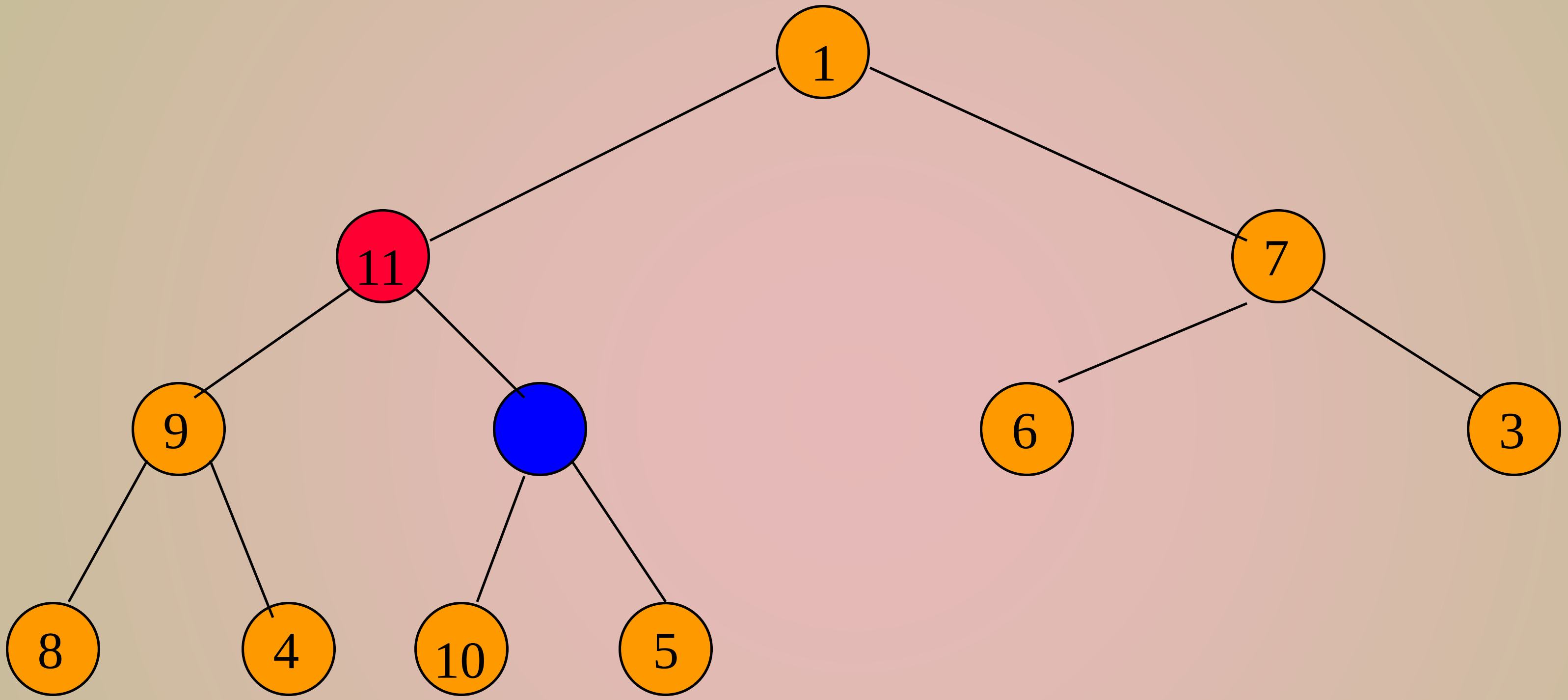
Initializing A Max Heap



Initializing A Max Heap

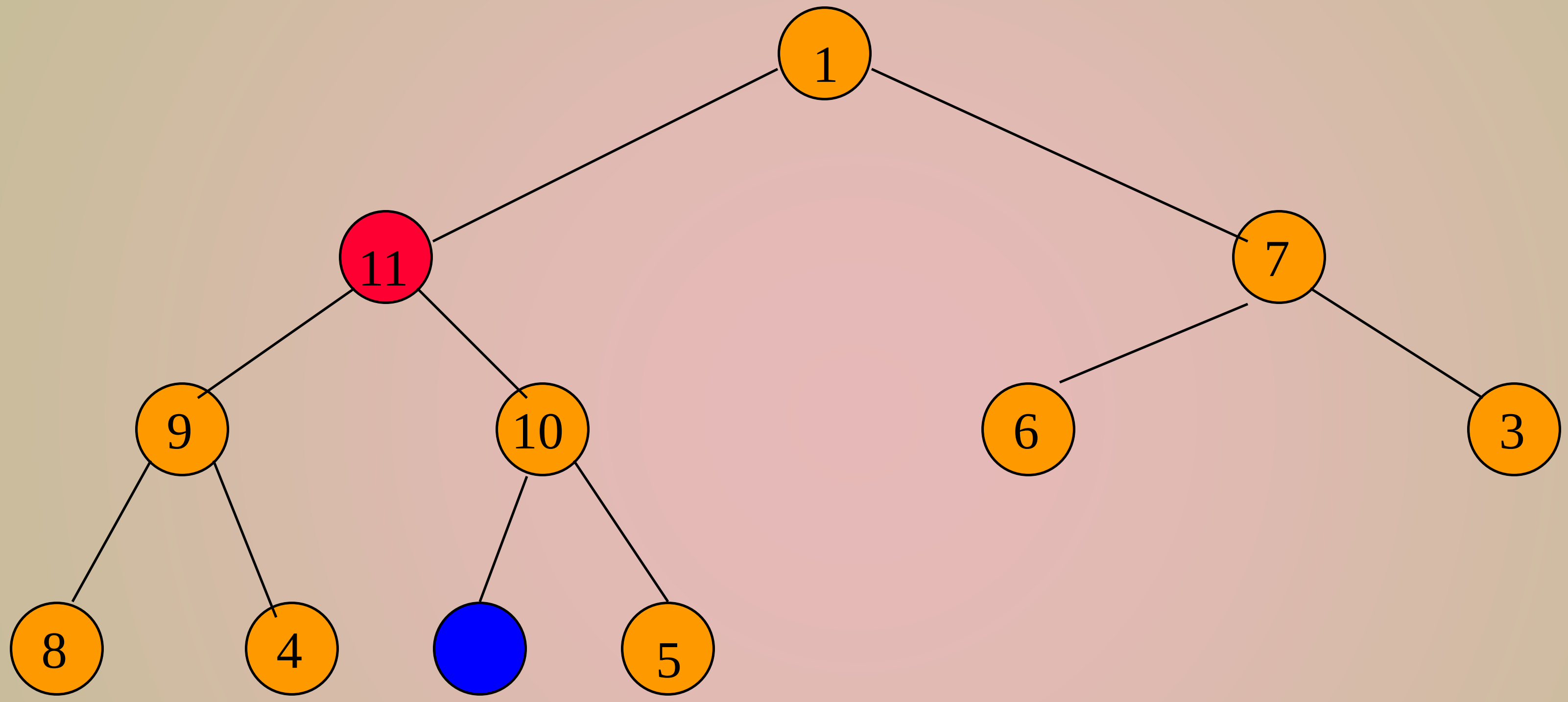


Initializing A Max Heap



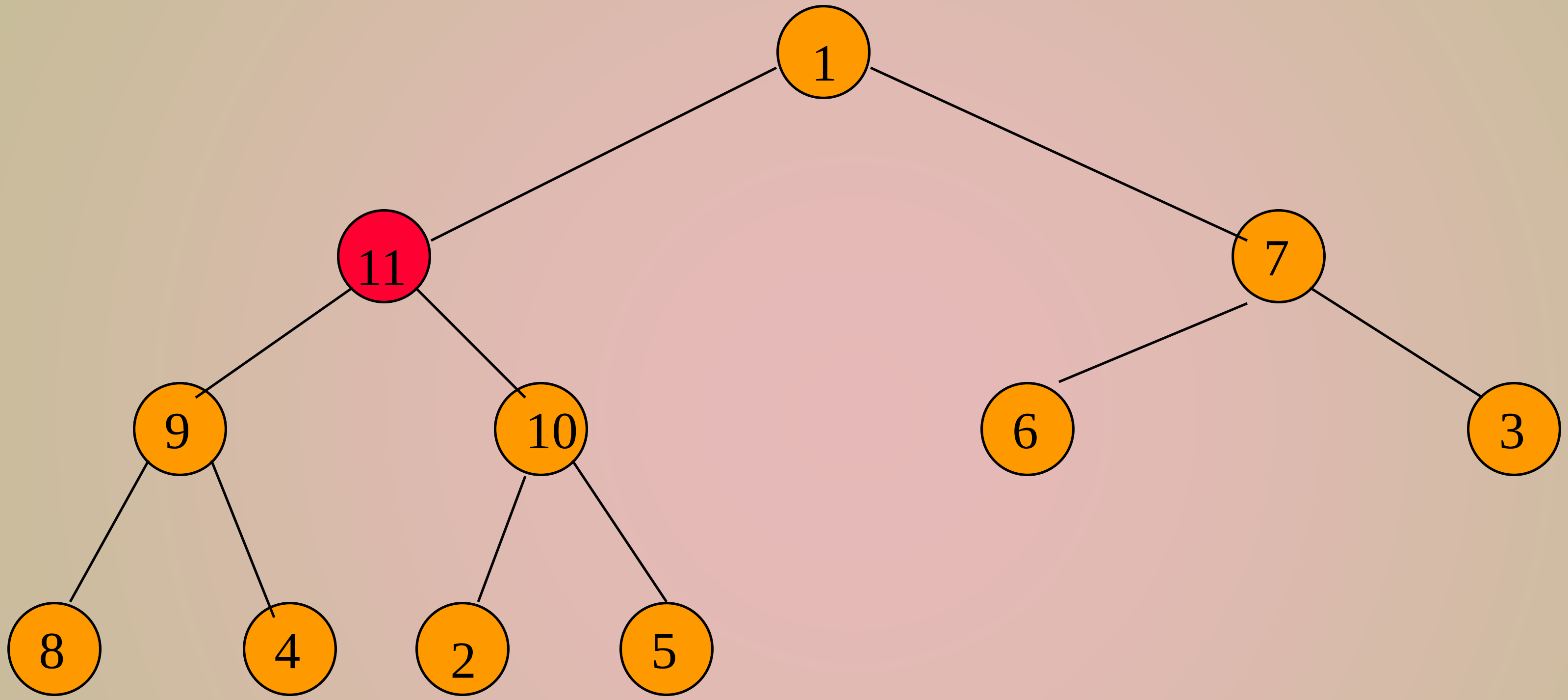
Find a home for 2.

Initializing A Max Heap



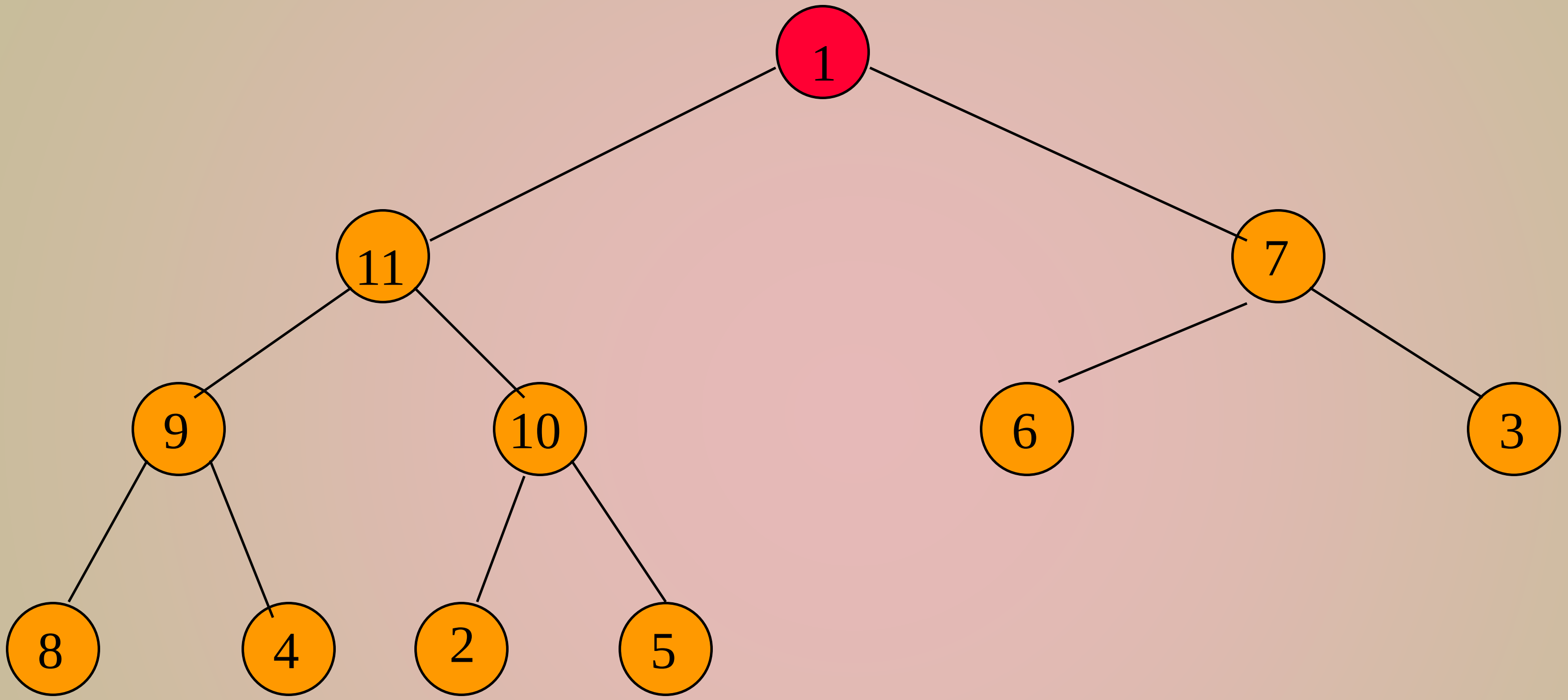
Find a home for 2.

Initializing A Max Heap



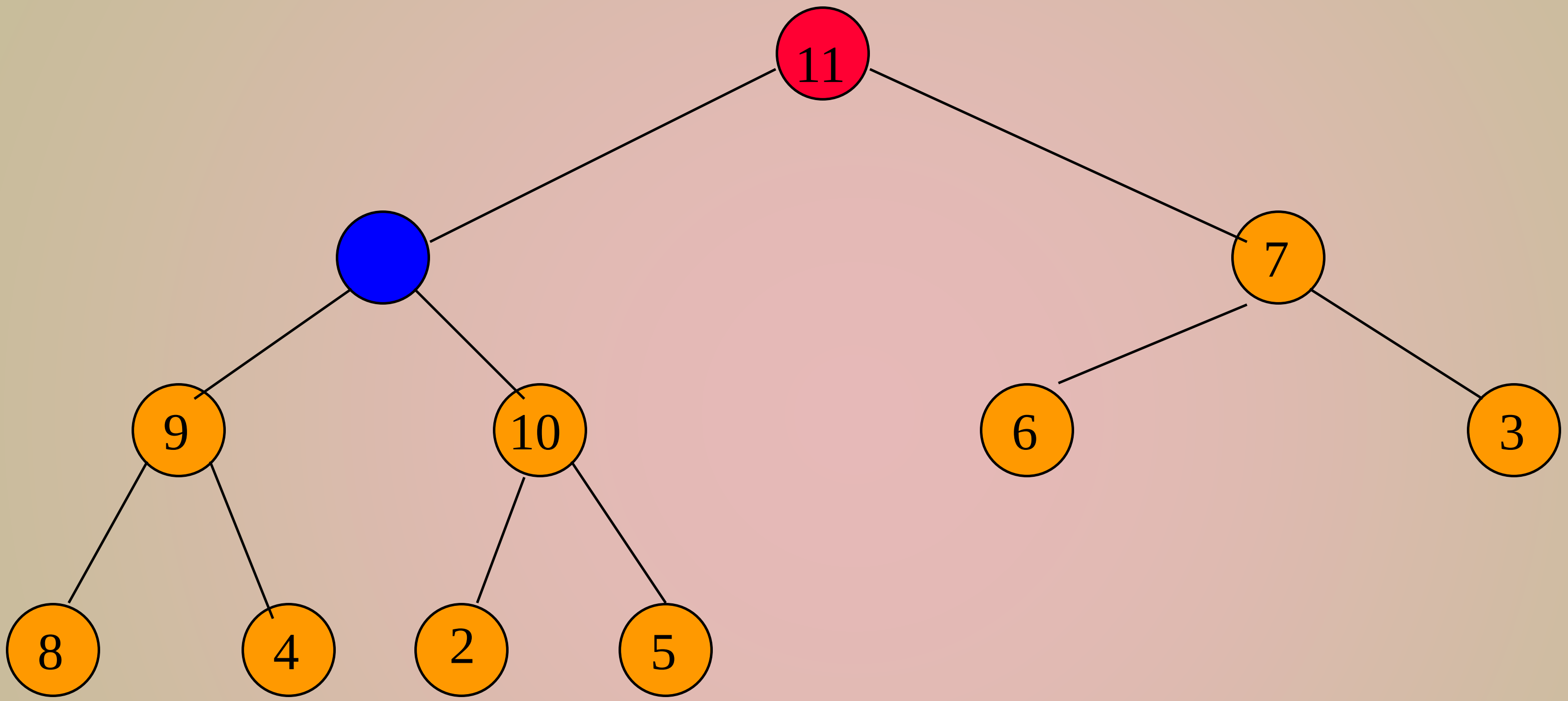
Done, move to next lower array position.

Initializing A Max Heap



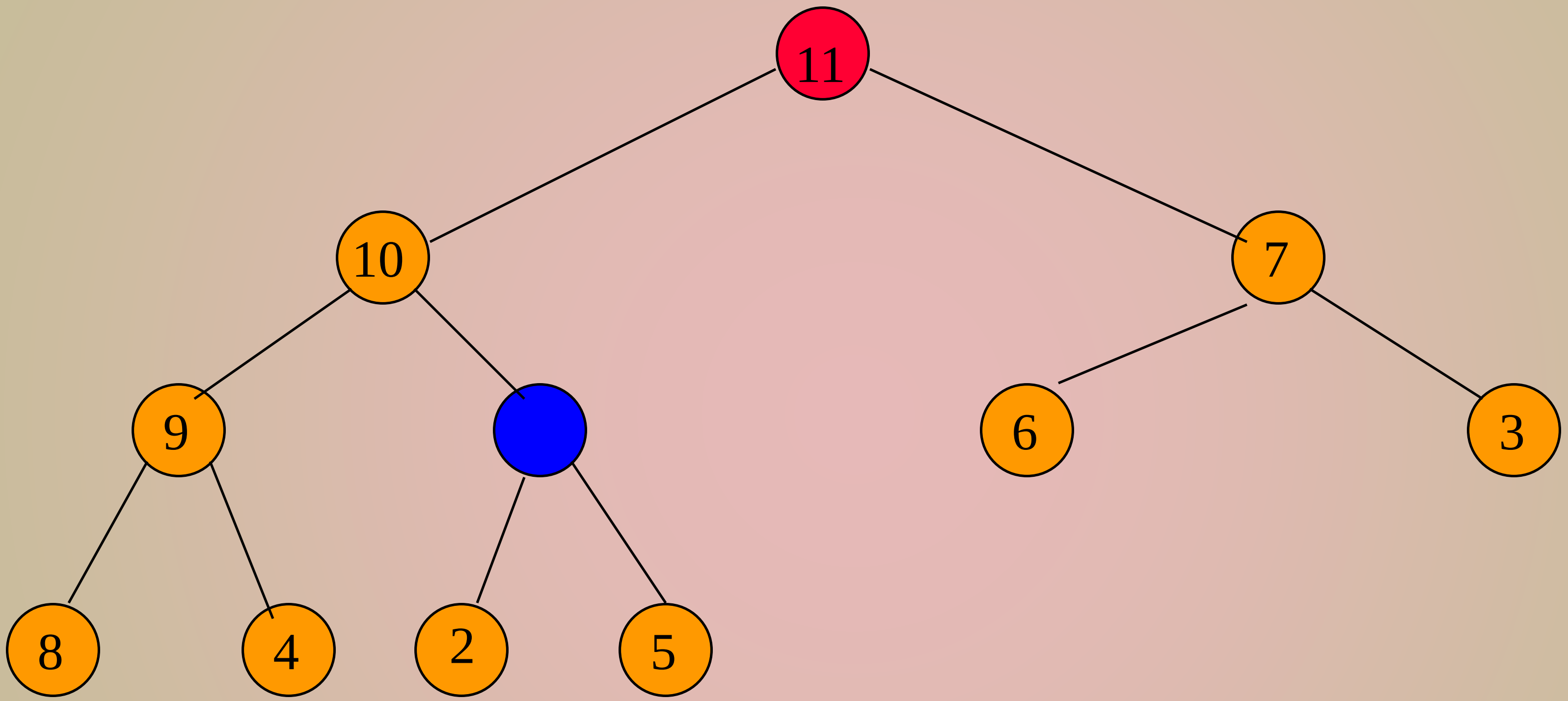
Find home for 1.

Initializing A Max Heap



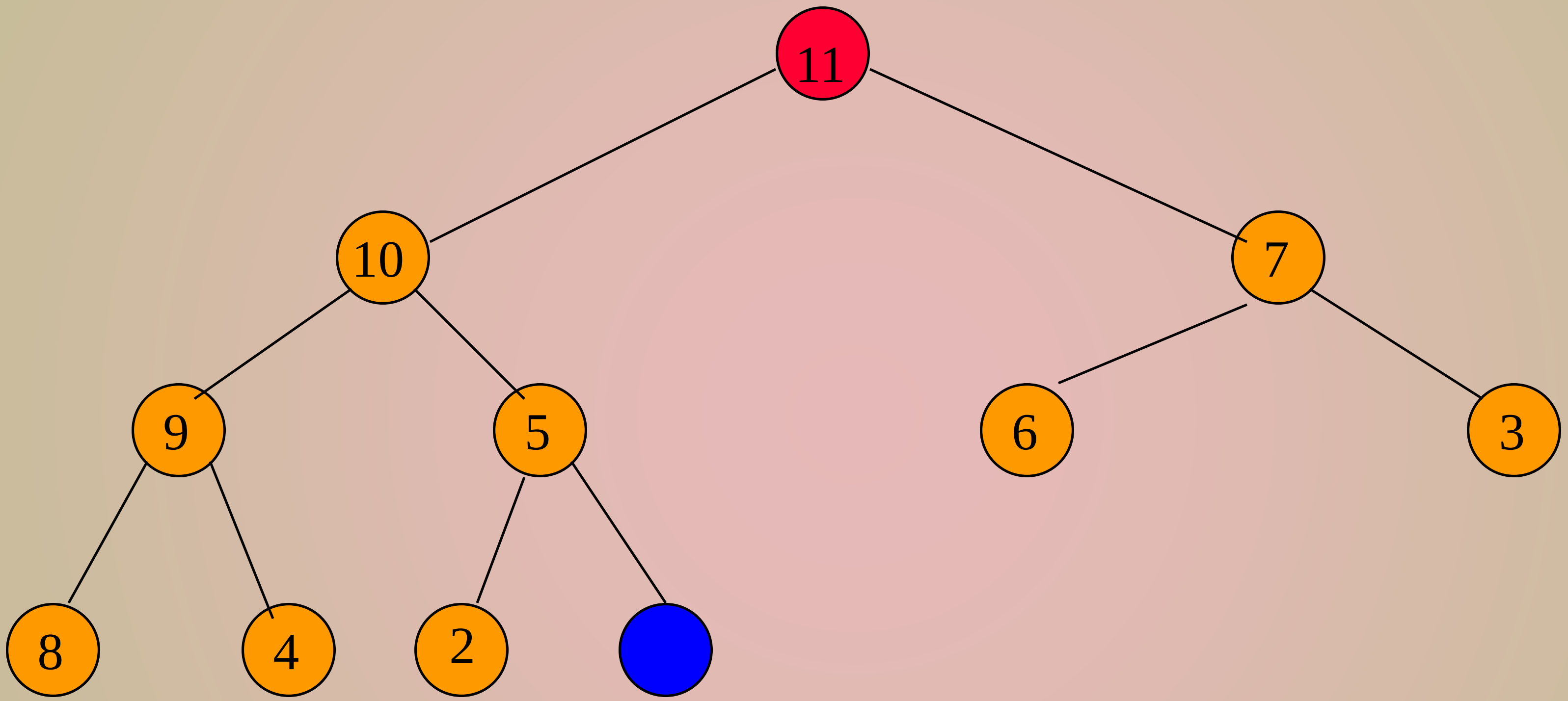
Find home for 1.

Initializing A Max Heap



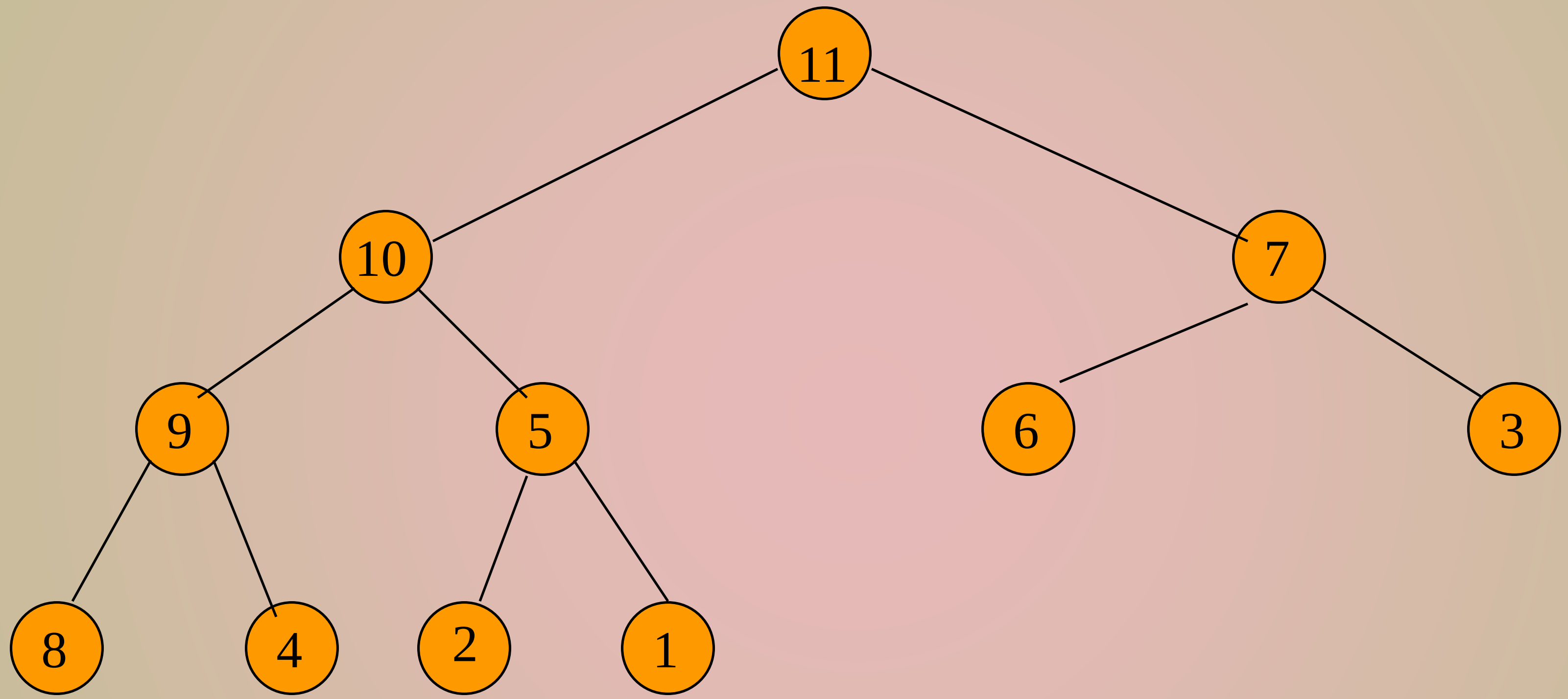
Find home for 1.

Initializing A Max Heap



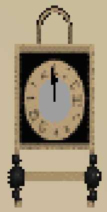
Find home for 1.

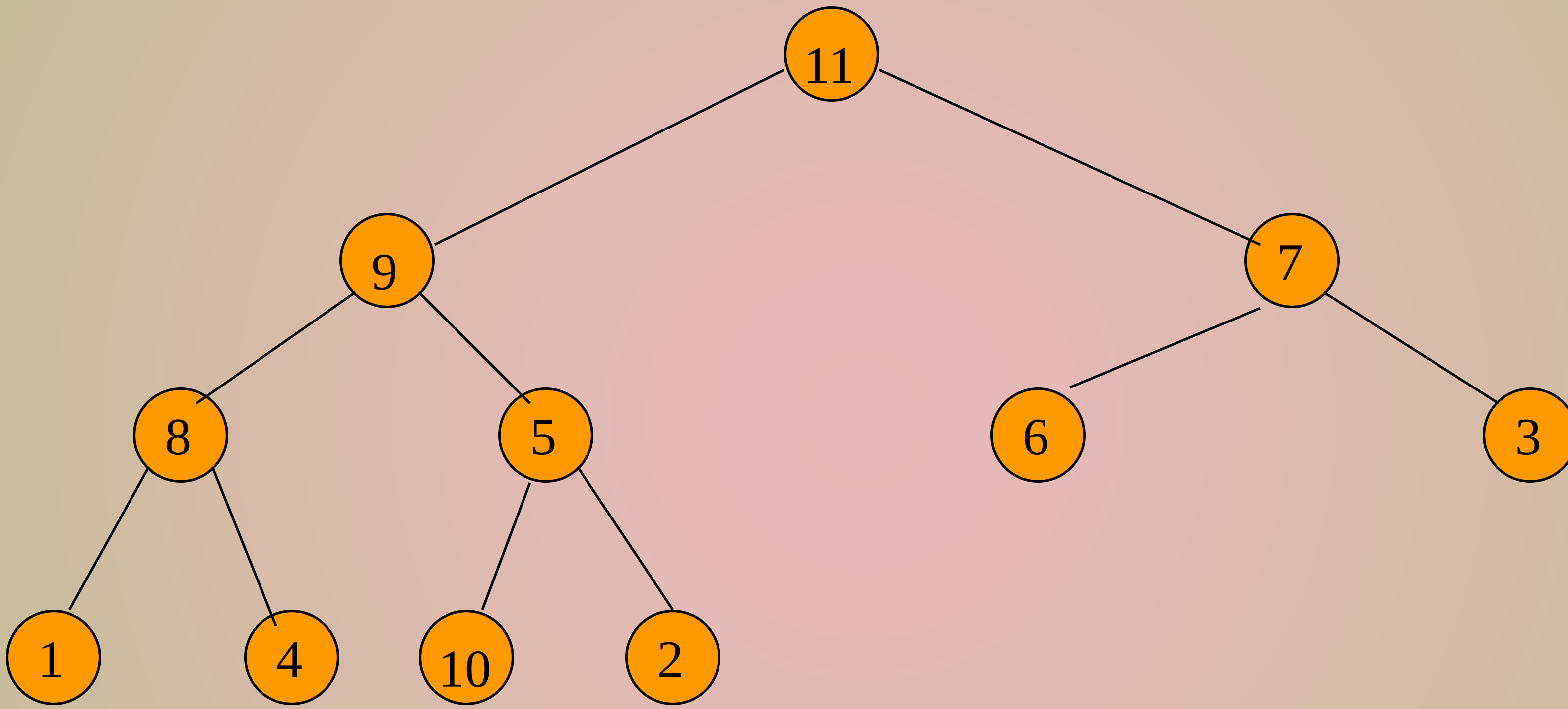
Initializing A Max Heap



Done.

Time Complexity



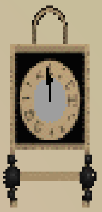


Height of heap $= h$.

Number of subtrees with root at level j is $\leq 2^{j-1}$.

Time for each subtree is $O(h-j+1)$.

Complexity



Time for level j subtrees is $\leq 2^{j-1}(h-j+1) = t(j)$.

Total time is $T = t(1) + t(2) + \ldots + t(h-1) = O(n)$.

$$T = \sum_{j=1}^{h} (h-j+1)2^{j-1} = \sum_{j=0}^{h-1} (h-j)2^{j}$$

$$= \sum_{j=0}^{h-1} \sum_{k=1}^{h-j} 2^{j} = \sum_{k=1}^{h} \sum_{j=0}^{h-k} 2^{j} = \sum_{k=1}^{h} \left(2^{h-k+1} - 1\right)$$

$$= \sum_{k=1}^{h} \left(2^{k} - 1\right) = 2^{h+1} - 1 - 1 - h \leq 2\left(2^{h}\right) \in O(n)$$

$$\sum_{j=0}^{h-1} (h-j)2^j$$

2^0	2^1	2^2	$\cdots$	2^{h-1}
2^0	2^1	2^2	$\cdots$	
2^0	2^1	2^2		
$\vdots$	$\vdots$			
2^0	2^1			
2^0				

Heap Sort

- ابتدا heap را با داده هایی که باید مرتب شوند پر می کنیم. (O(n))
- در هر مرحله کوچک ترین عنصر را حذف می کنیم و به ترتیب در لیست خروجی قرار می دهیم. O(logn)
- با n بار تکرار عمل حذف عنصر کمینه، تمام عناصر به ترتیب در لیست خروجی قرار می گیرند.
- زمان کل: O(nlogn)