

Chapter Five

Priority Queues and Heaps

Priority Queues

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**)

Complexity Of Operations

Use a heap or a leftist tree (both are defined later).

empty, size, and top $\Rightarrow O(1)$ time

insert (push) and remove (pop) $\Rightarrow O(\log n)$
time where n is the size of the priority
queue

Applications

Sorting

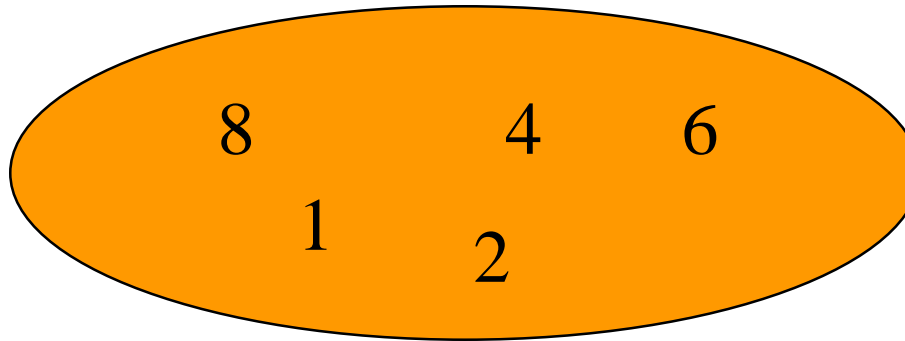
- use element key as priority
- insert elements to be sorted into a priority queue
- remove/pop elements in priority order
 - if a min priority queue is used, elements are extracted in ascending order of priority (or key)
 - if a max priority queue is used, elements are extracted in descending order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

***Sorting Example

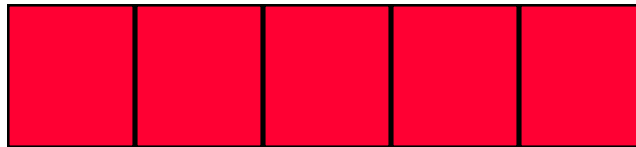
Sort five elements whose keys are 6, 8, 2, 4, 1 using a max priority queue.

- Insert the five elements into a max priority queue.
- Do five remove max operations placing removed elements into the sorted array from right to left.

After Inserting Into Max Priority Queue

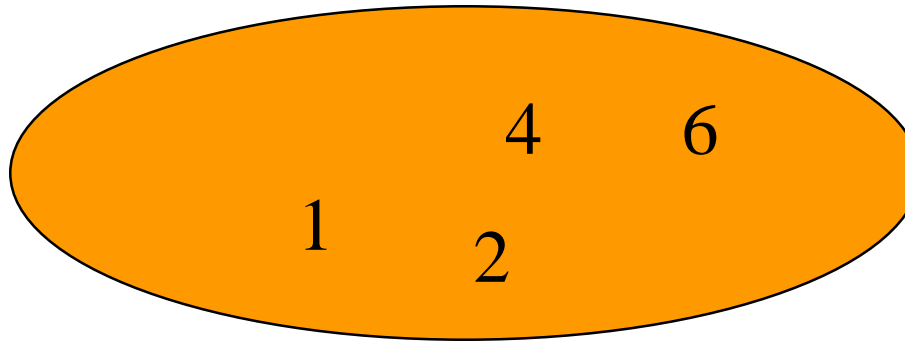


Max Priority
Queue



Sorted Array

After First Remove Max Operation

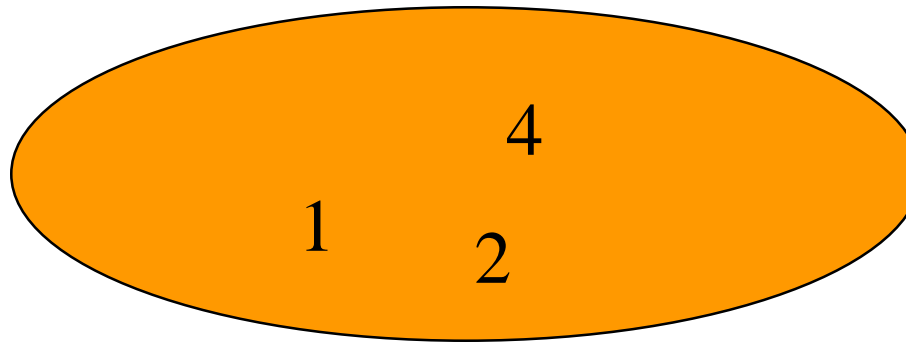


Max Priority
Queue



Sorted Array

After Second Remove Max Operation

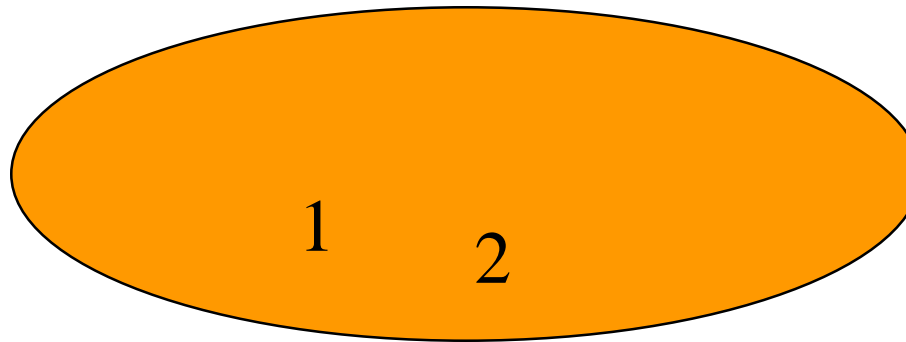


Max Priority
Queue

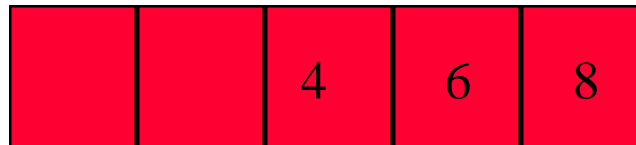


Sorted Array

After Third Remove Max Operation

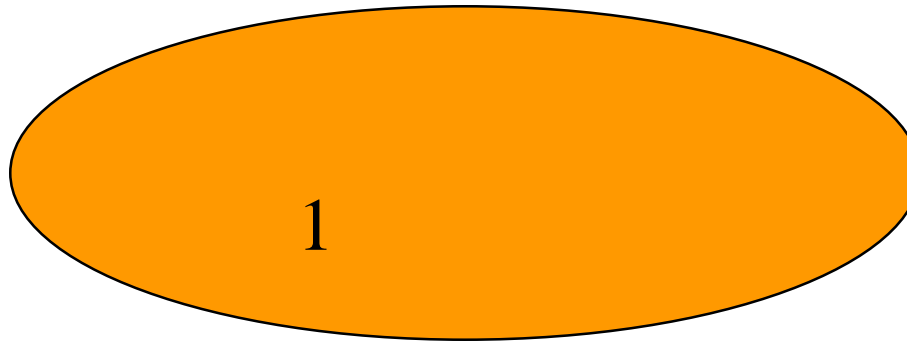


Max Priority
Queue



Sorted Array

After Fourth Remove Max Operation

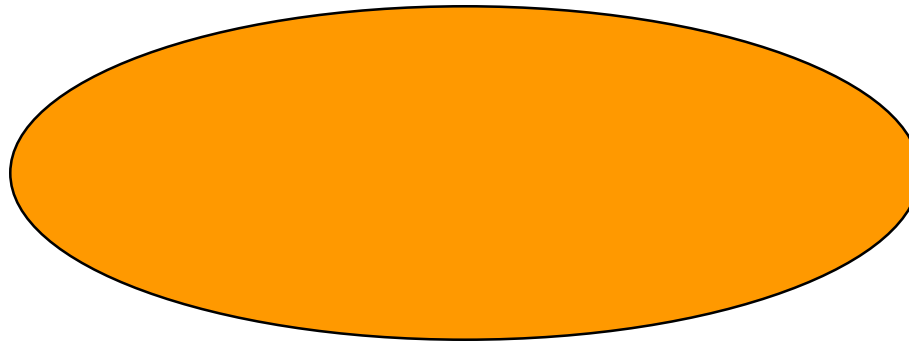


Max Priority
Queue



Sorted Array

After Fifth Remove Max Operation



Max Priority
Queue



Sorted Array

Complexity Of Sorting

Sort n elements.

- n insert operations $\Rightarrow O(n \log n)$ time.
- n remove max operations $\Rightarrow O(n \log n)$ time.
- total time is $O(n \log n)$.
- compare with $O(n^2)$ for insertion sort of Lecture 1.

Heap Sort

Uses a max priority queue that is implemented as a heap.

Initial insert operations are replaced by a heap initialization step that takes $O(n)$ time.

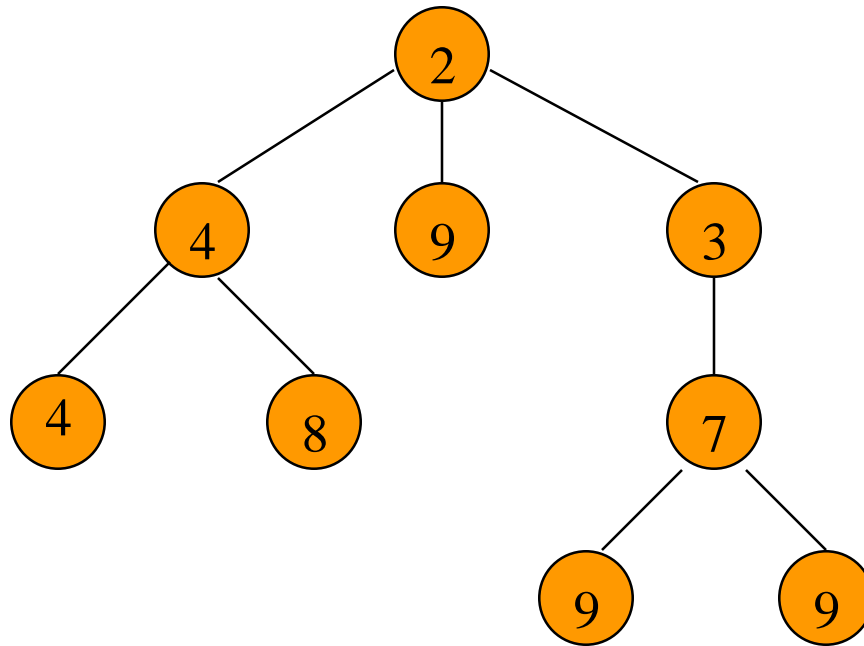
Min Tree Definition

Each tree node has a value.

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

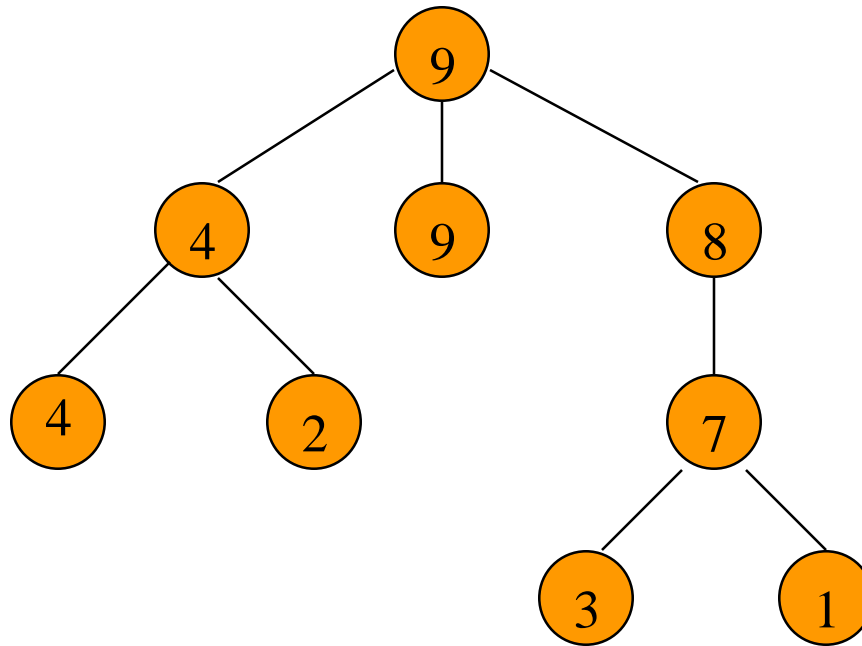
Equivalently, no descendent has a smaller value.

Min Tree Example



Root has minimum element.

Max Tree Example

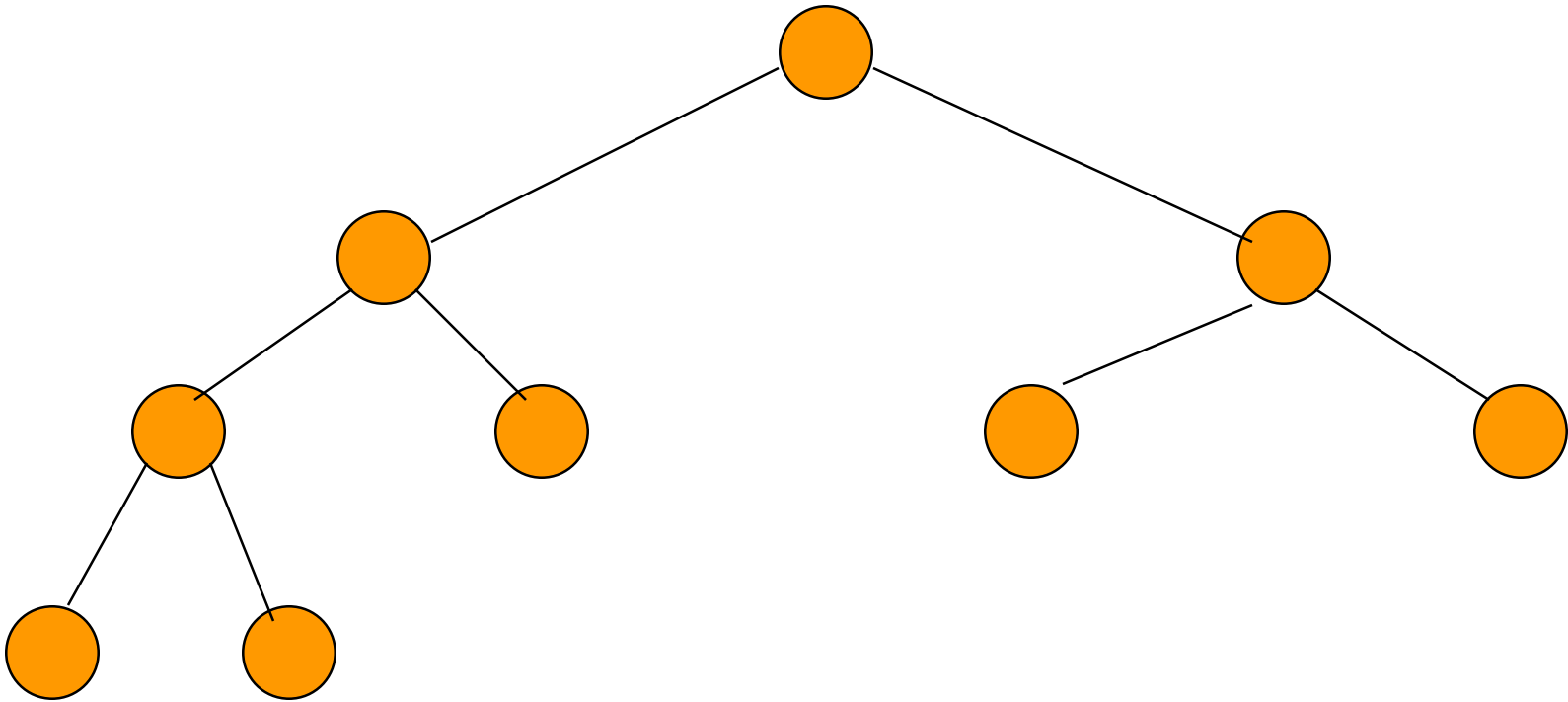


Root has maximum element.

Min Heap Definition

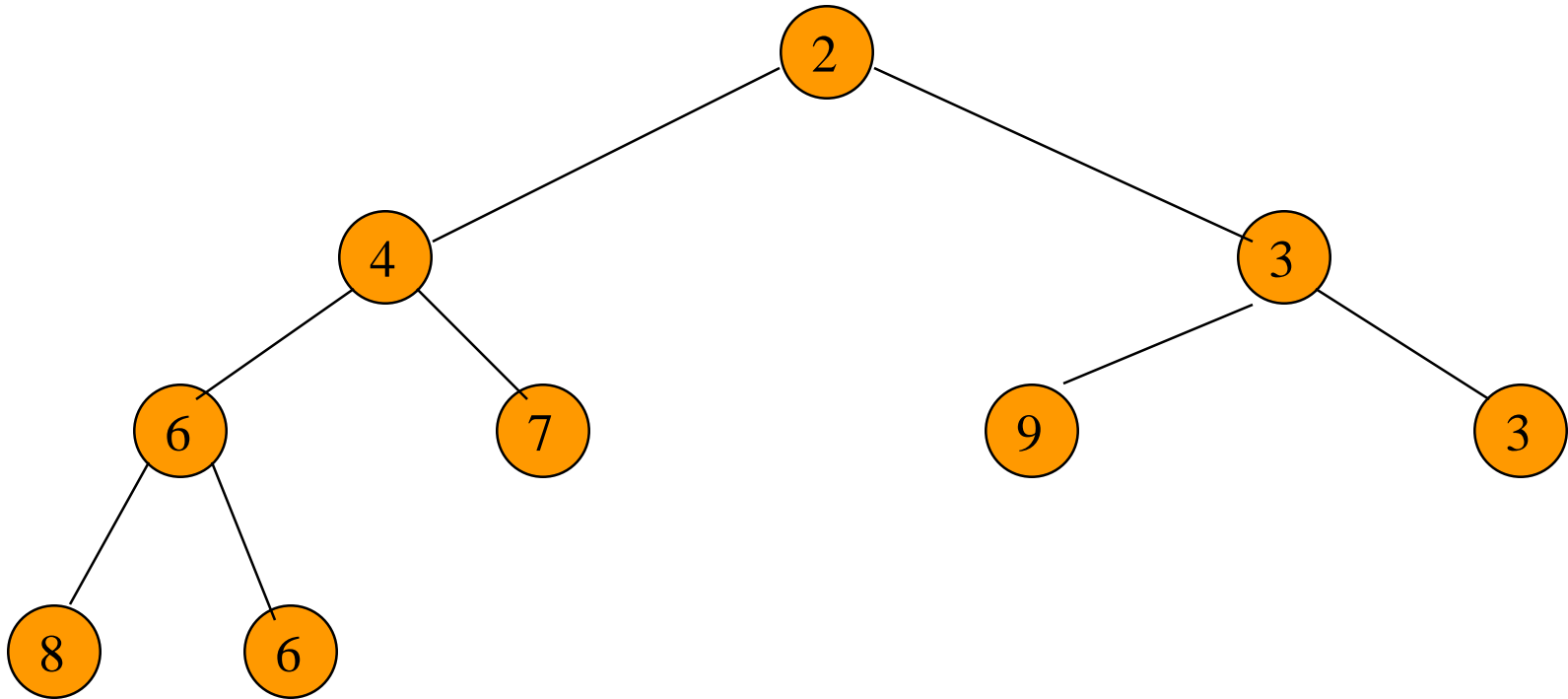
- complete binary tree
- min tree

Min Heap With 9 Nodes



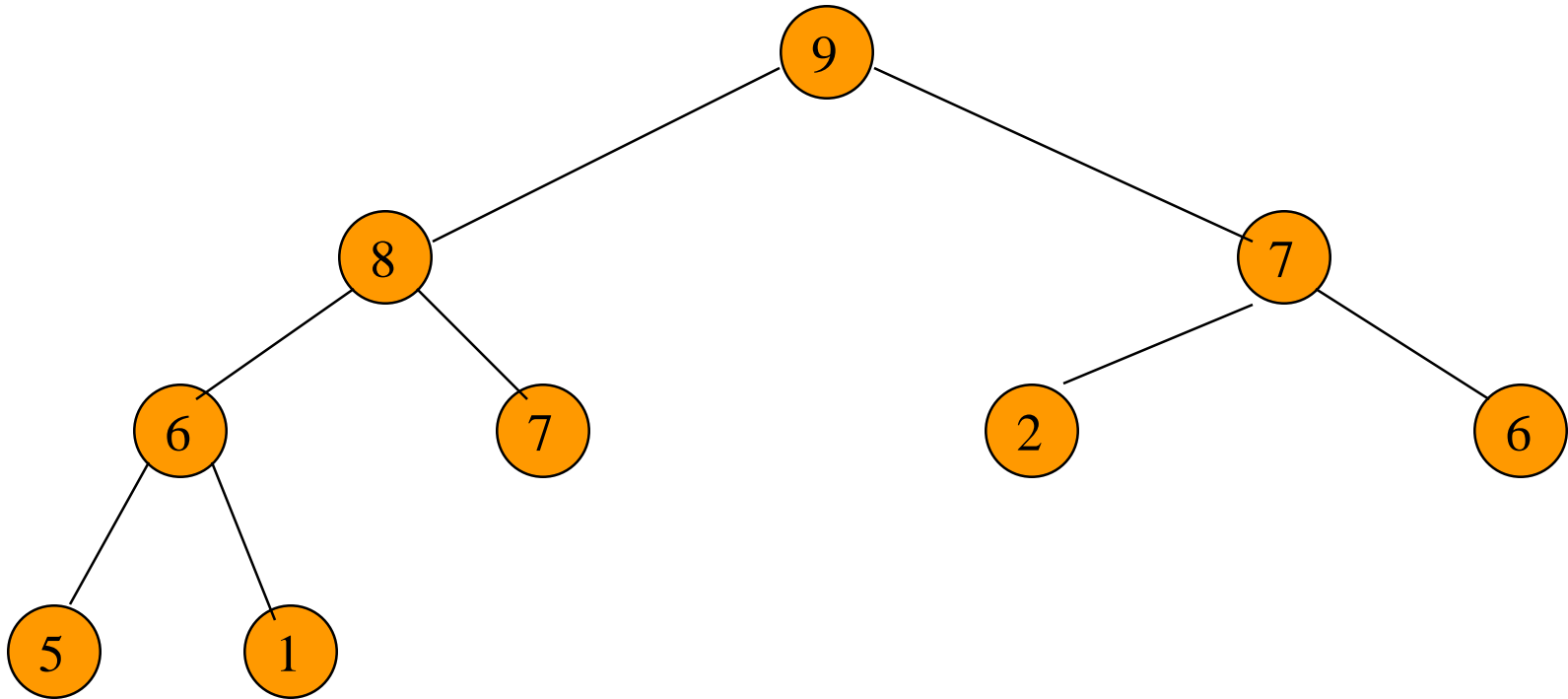
Complete binary tree with 9 nodes.

Min Heap With 9 Nodes



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

Max Heap With 9 Nodes

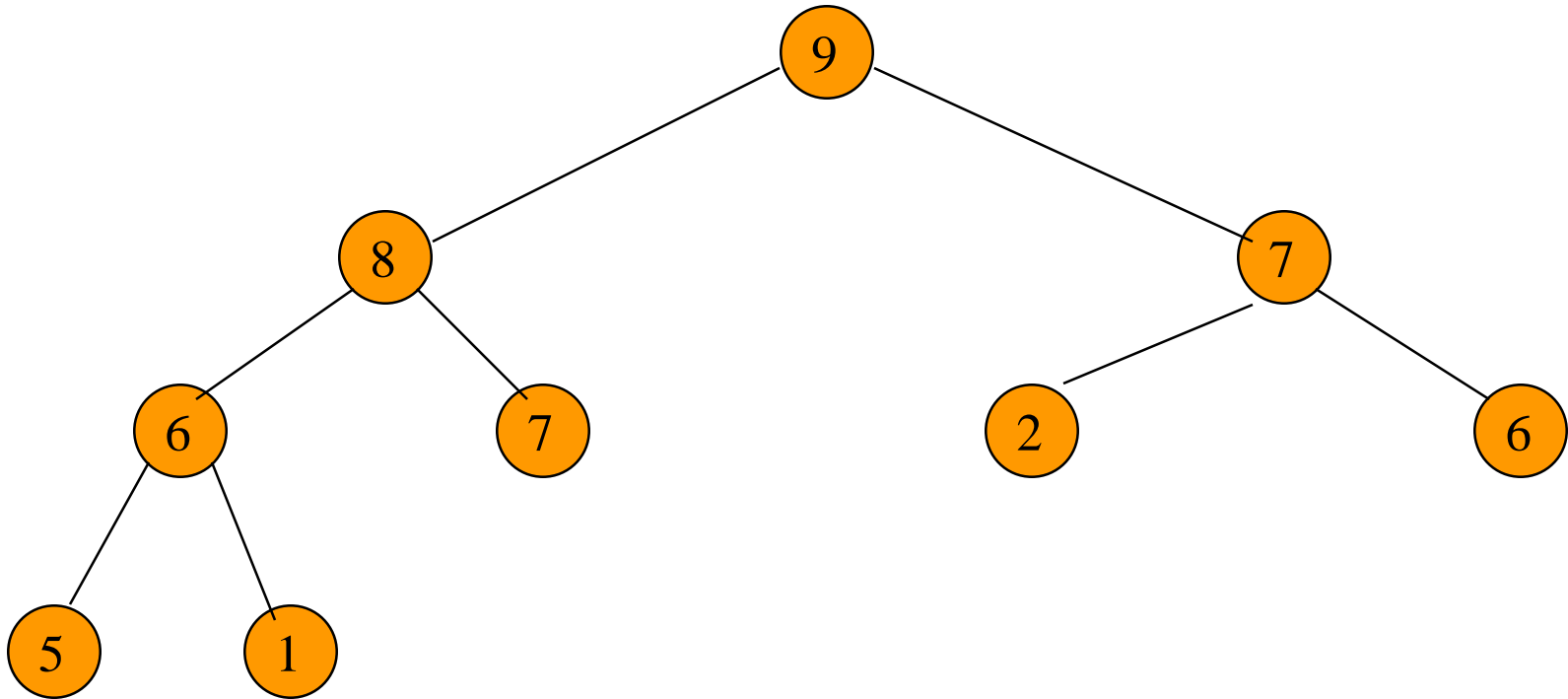


Complete binary tree with 9 nodes
that is also a max tree.

Heap Height

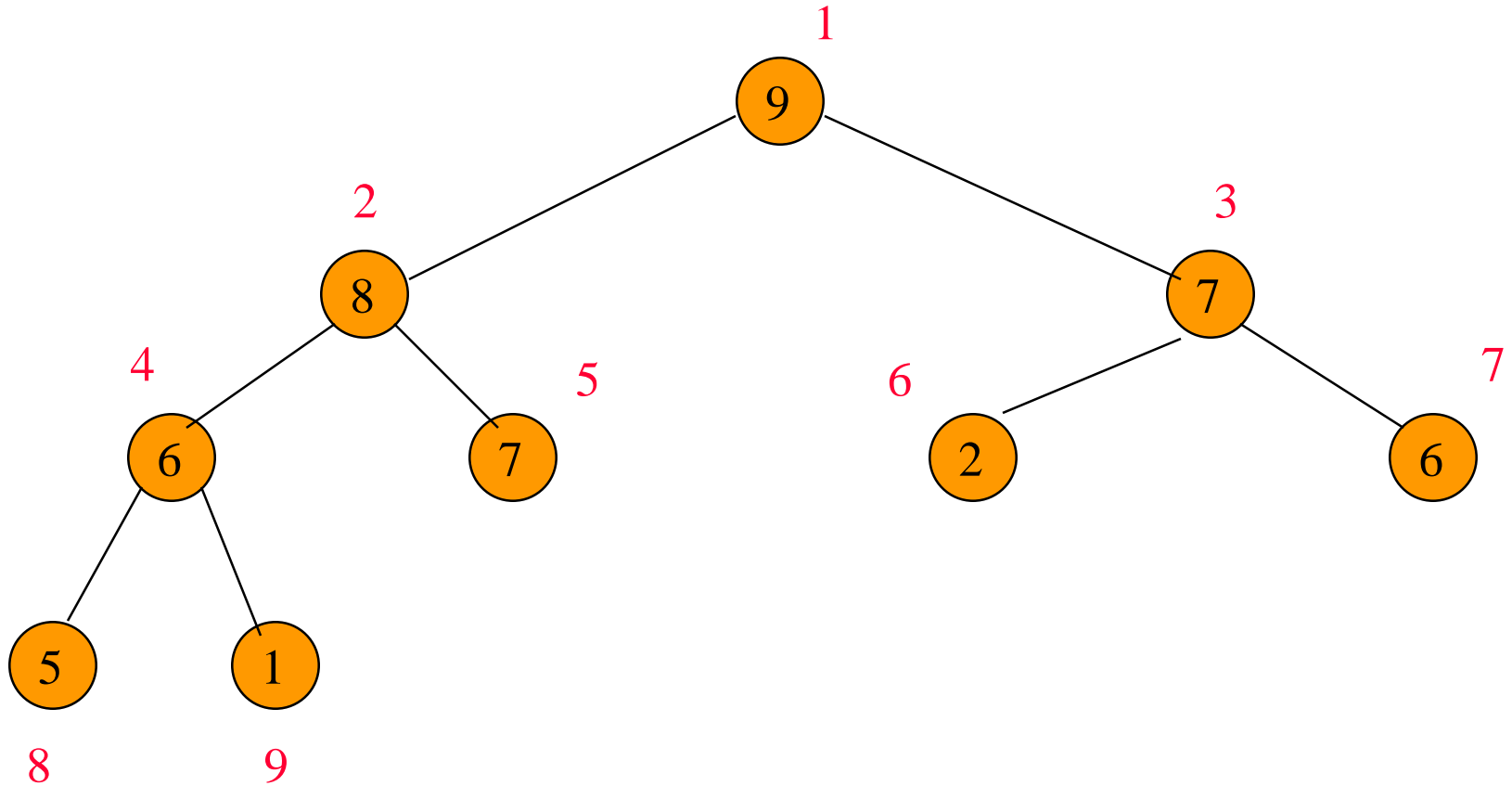
Since a heap is a complete binary tree, the height of an **n** node heap is $\log_2 (n+1)$.

A Heap Is Efficiently Represented As An Array

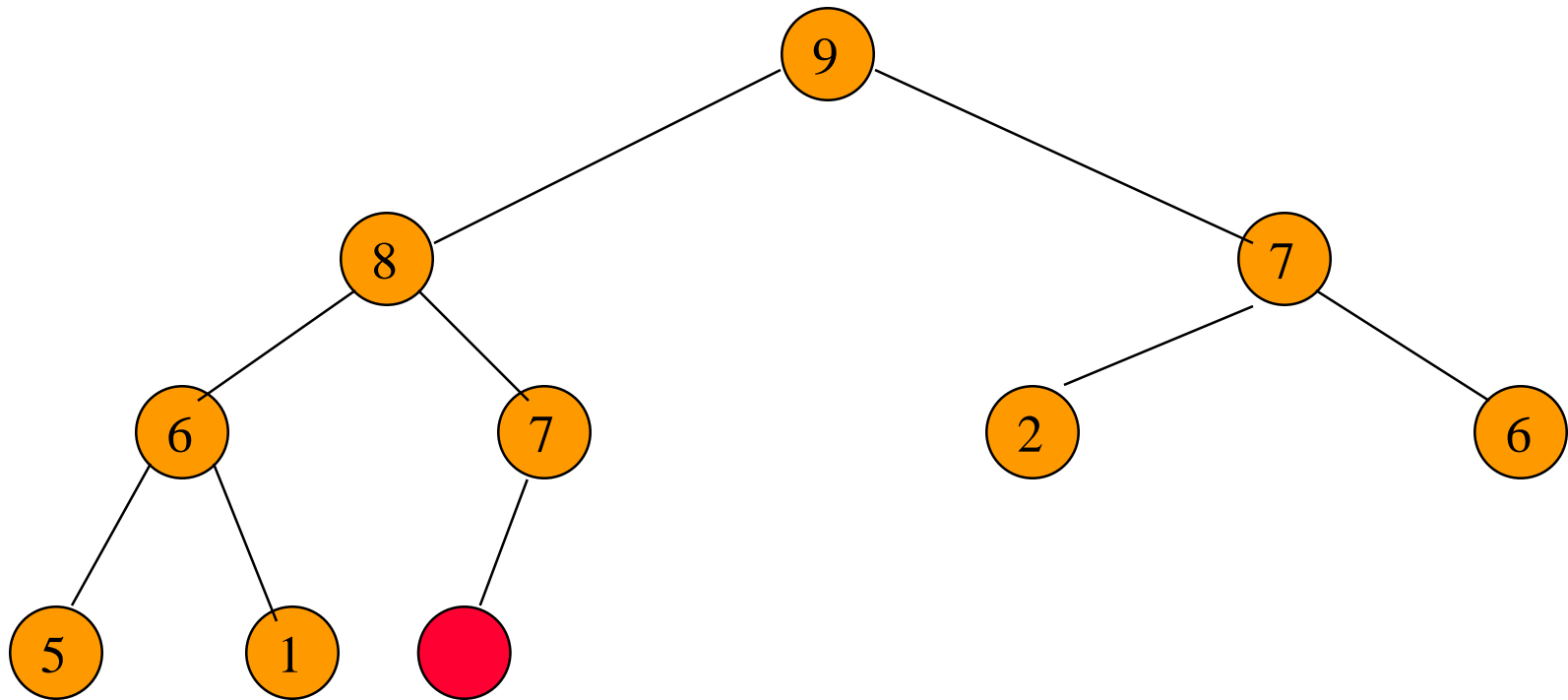


0 1 2 3 4 5 6 7 8 9 10

Moving Up And Down A Heap

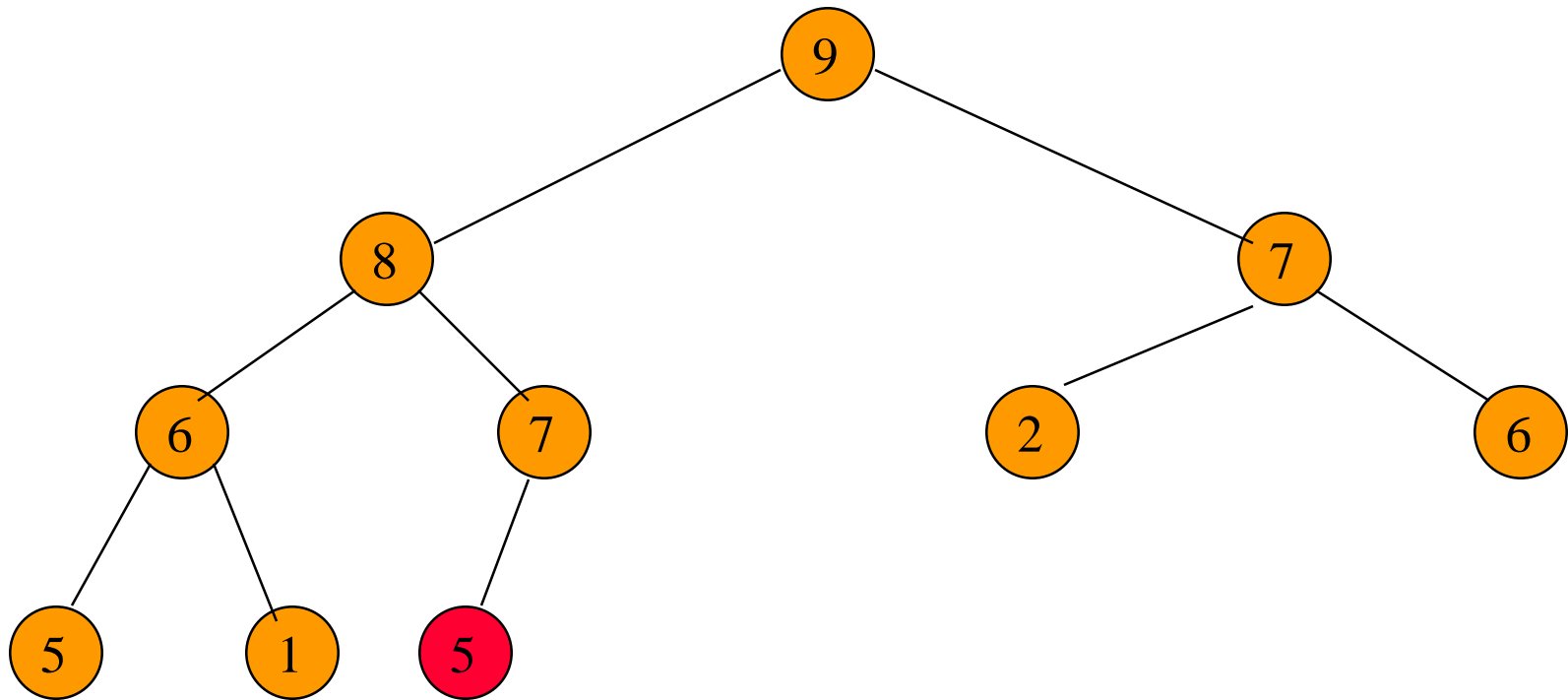


Inserting An Element Into A Max Heap



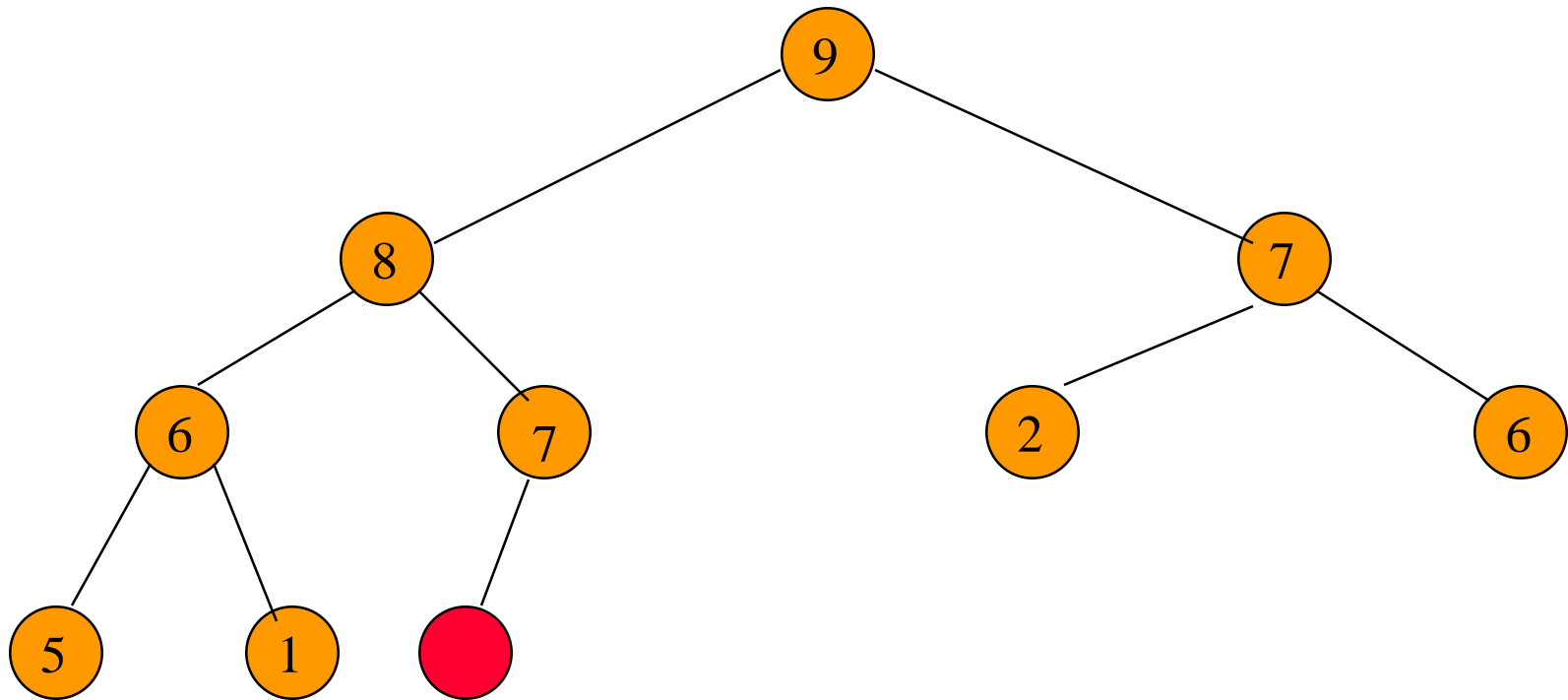
Complete binary tree with 10 nodes.

Inserting An Element Into A Max Heap



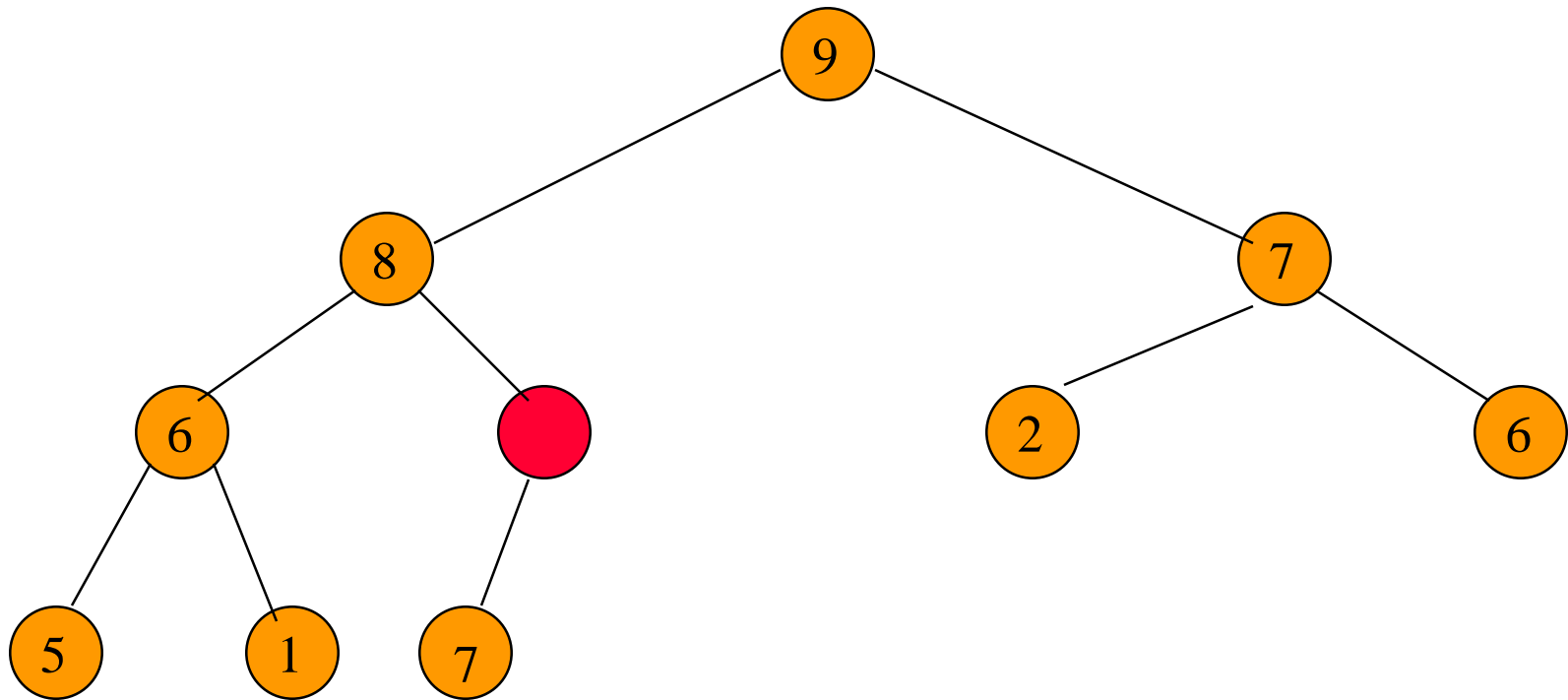
New element is 5.

Inserting An Element Into A Max Heap



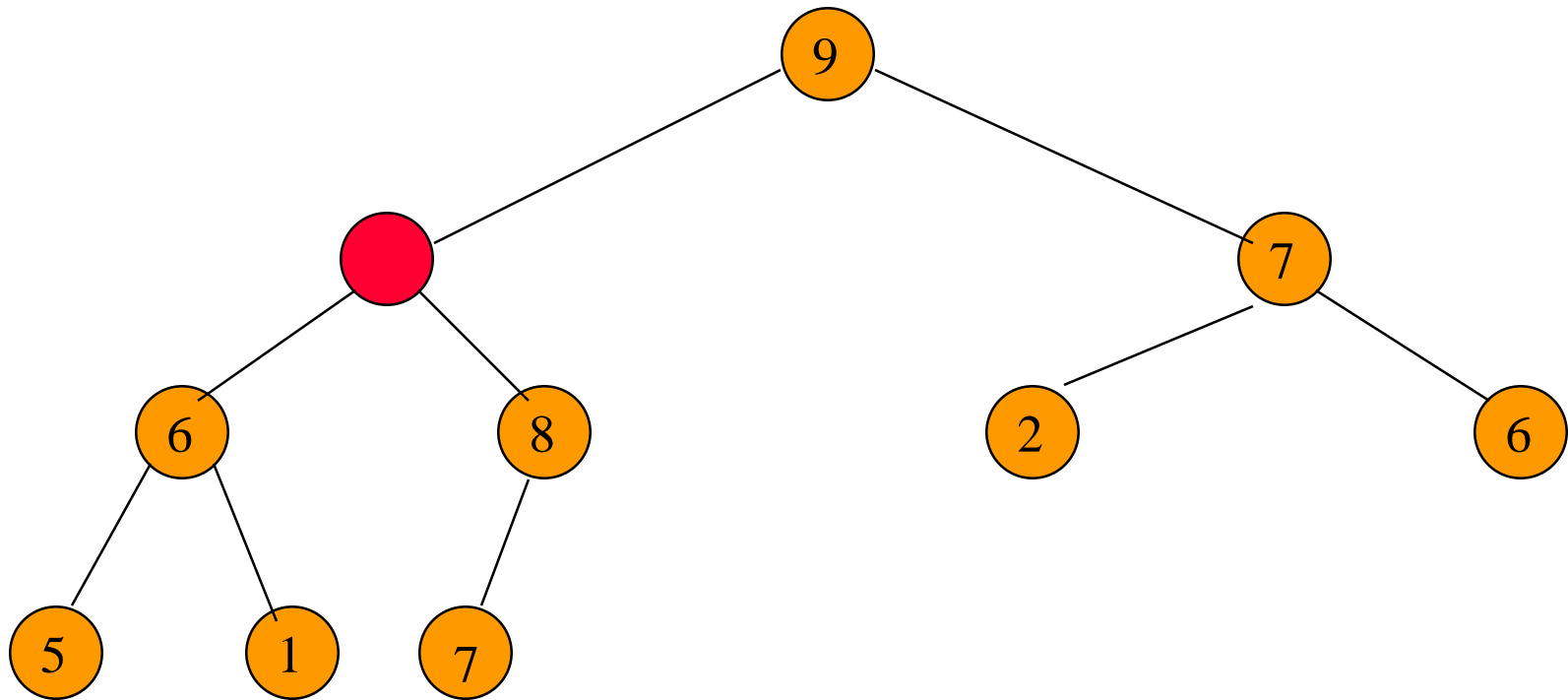
New element is 20.

Inserting An Element Into A Max Heap



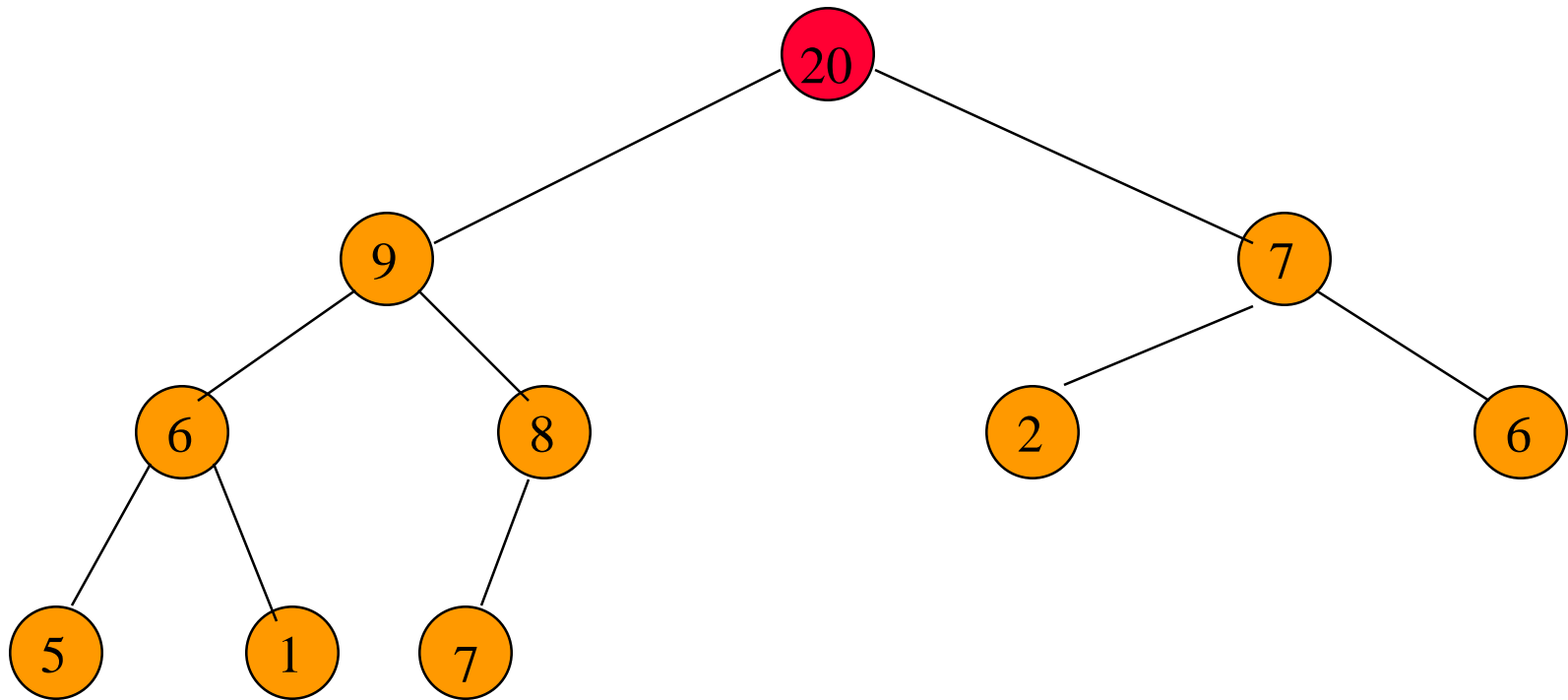
New element is 20.

Inserting An Element Into A Max Heap



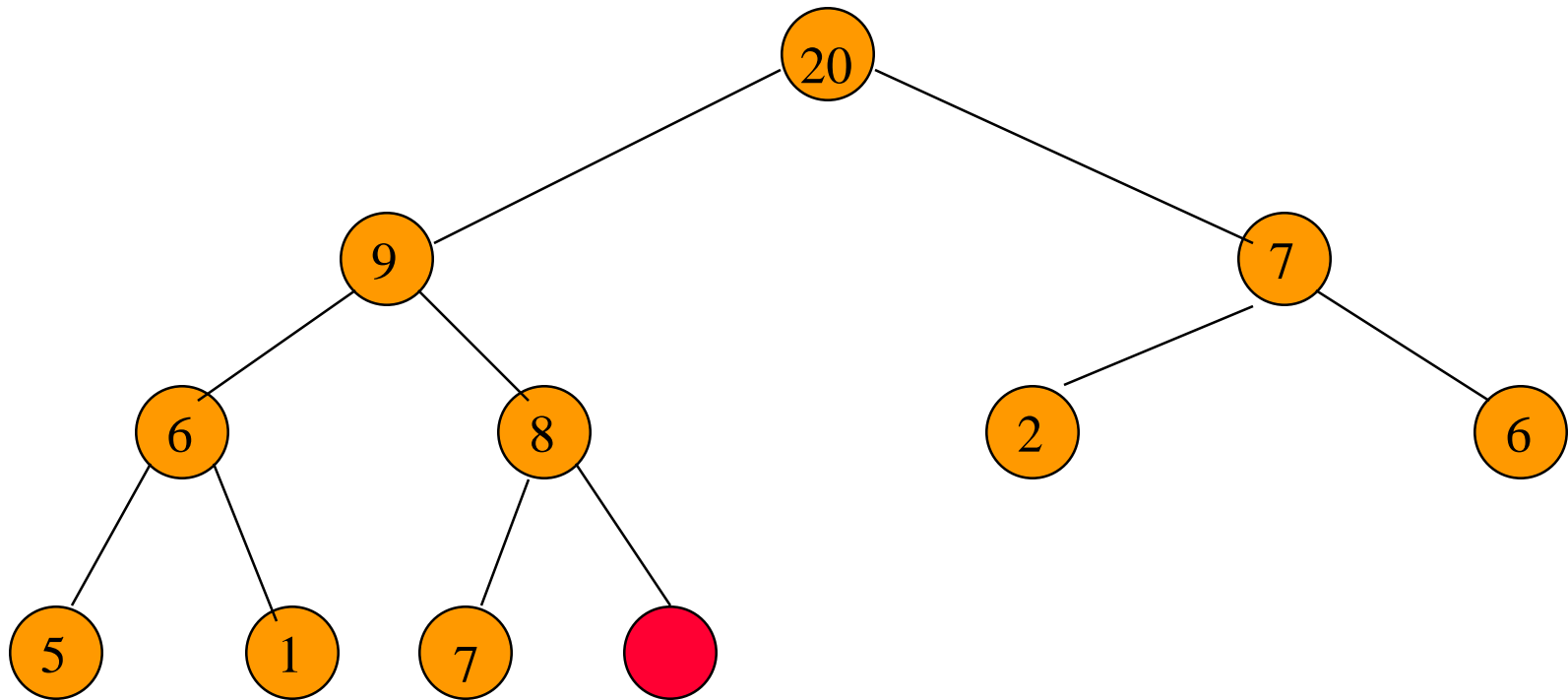
New element is 20.

Inserting An Element Into A Max Heap



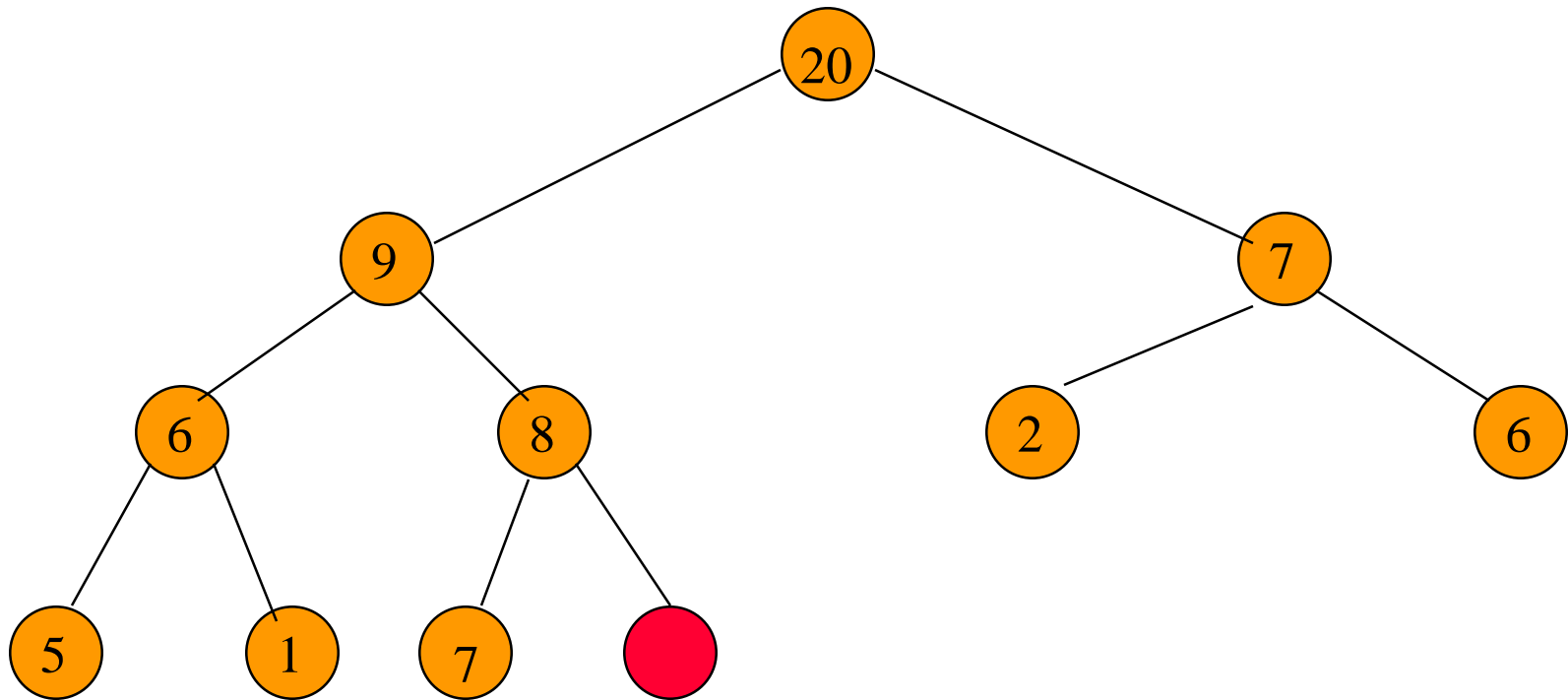
New element is 20.

Inserting An Element Into A Max Heap



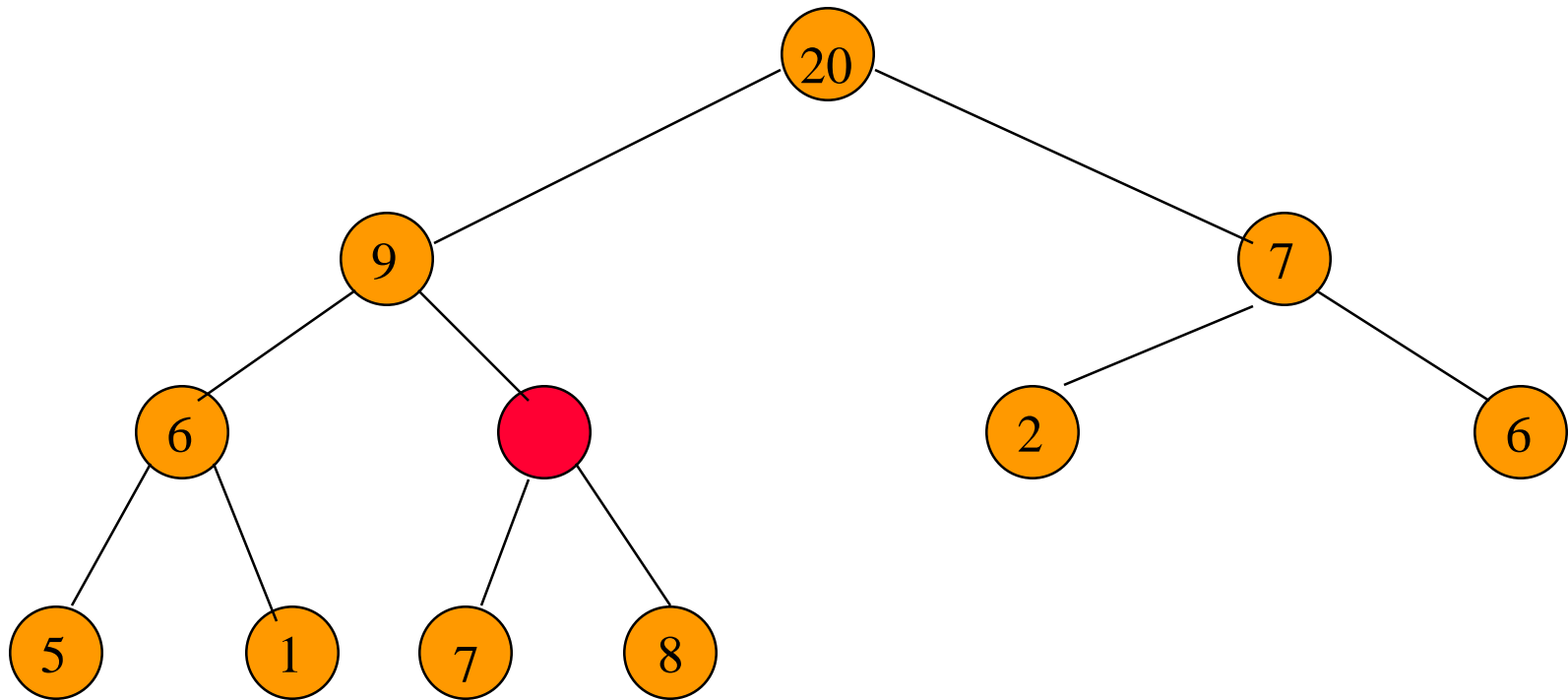
Complete binary tree with **11** nodes.

Inserting An Element Into A Max Heap



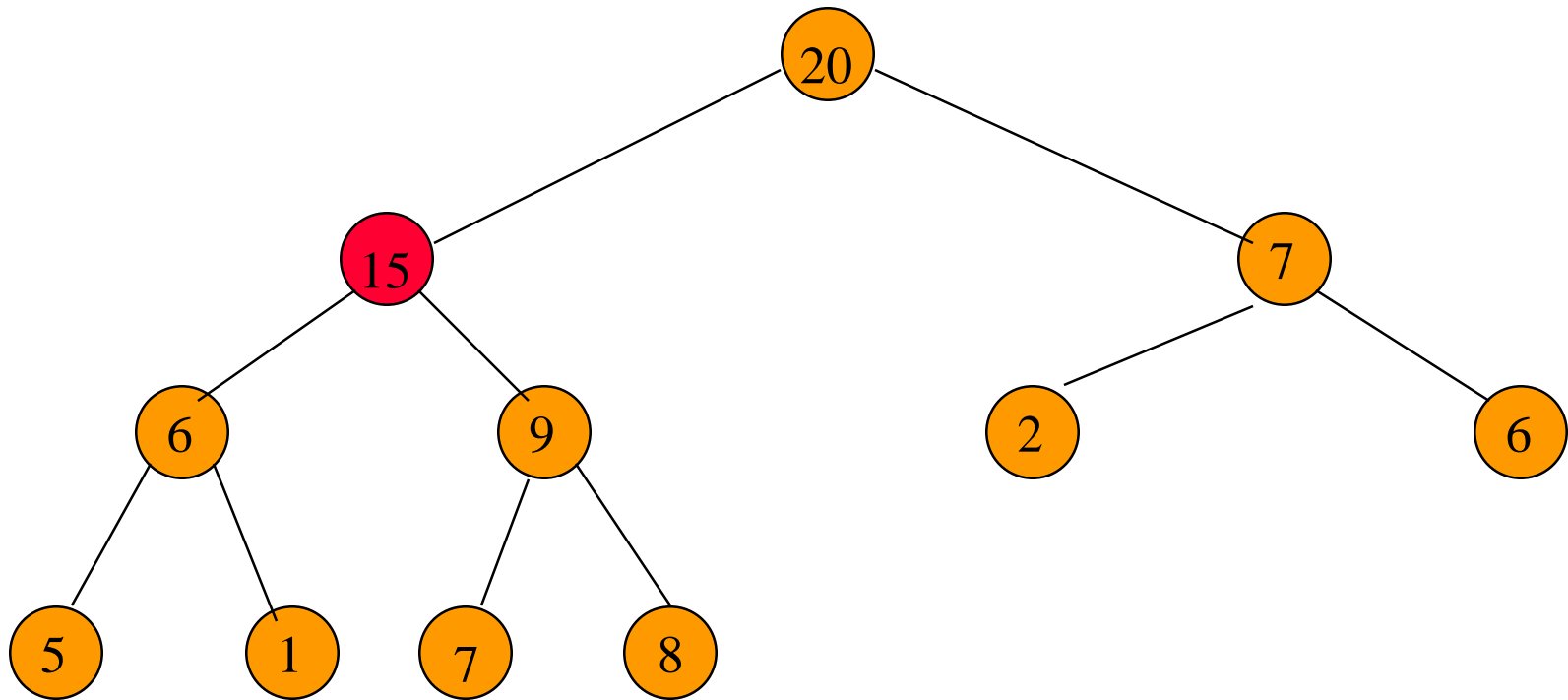
New element is 15.

Inserting An Element Into A Max Heap



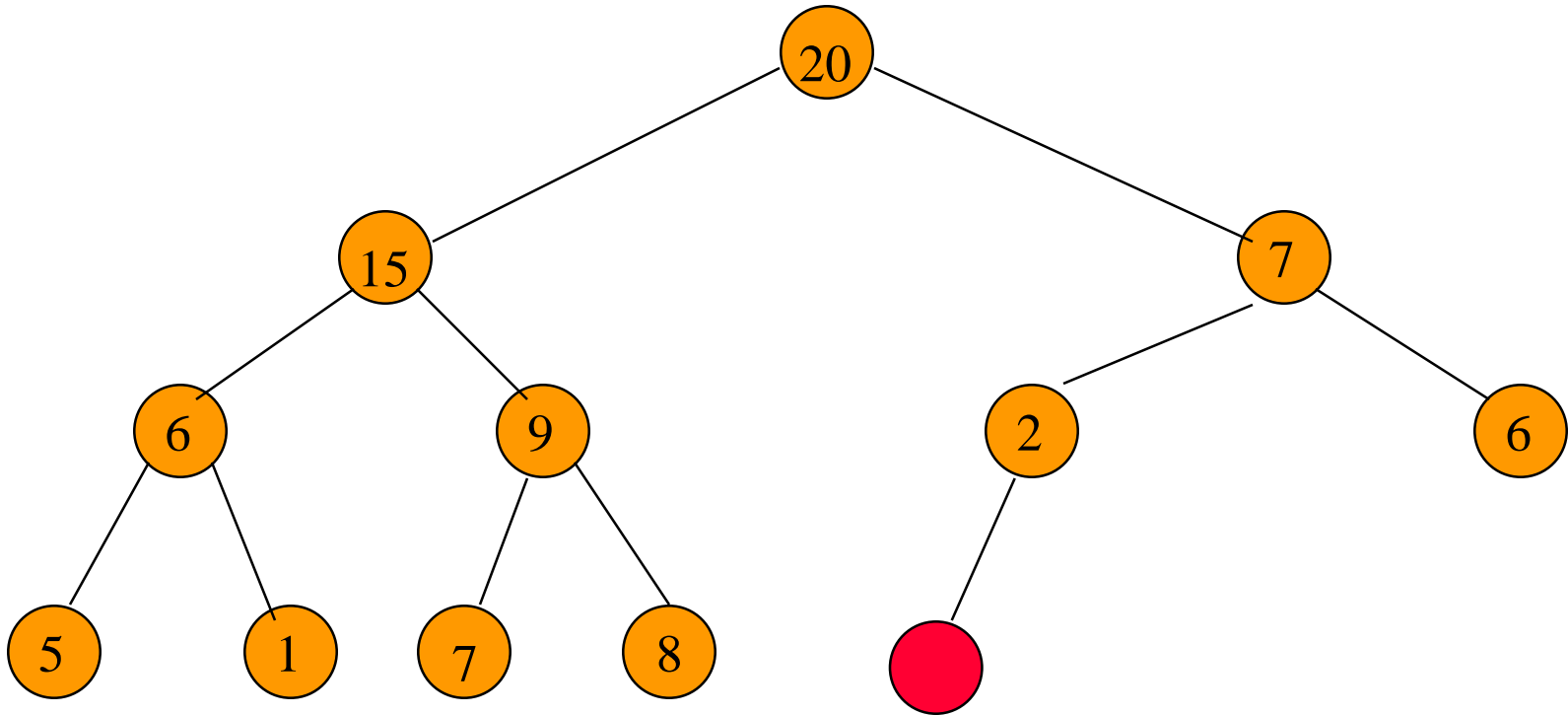
New element is 15.

Inserting An Element Into A Max Heap



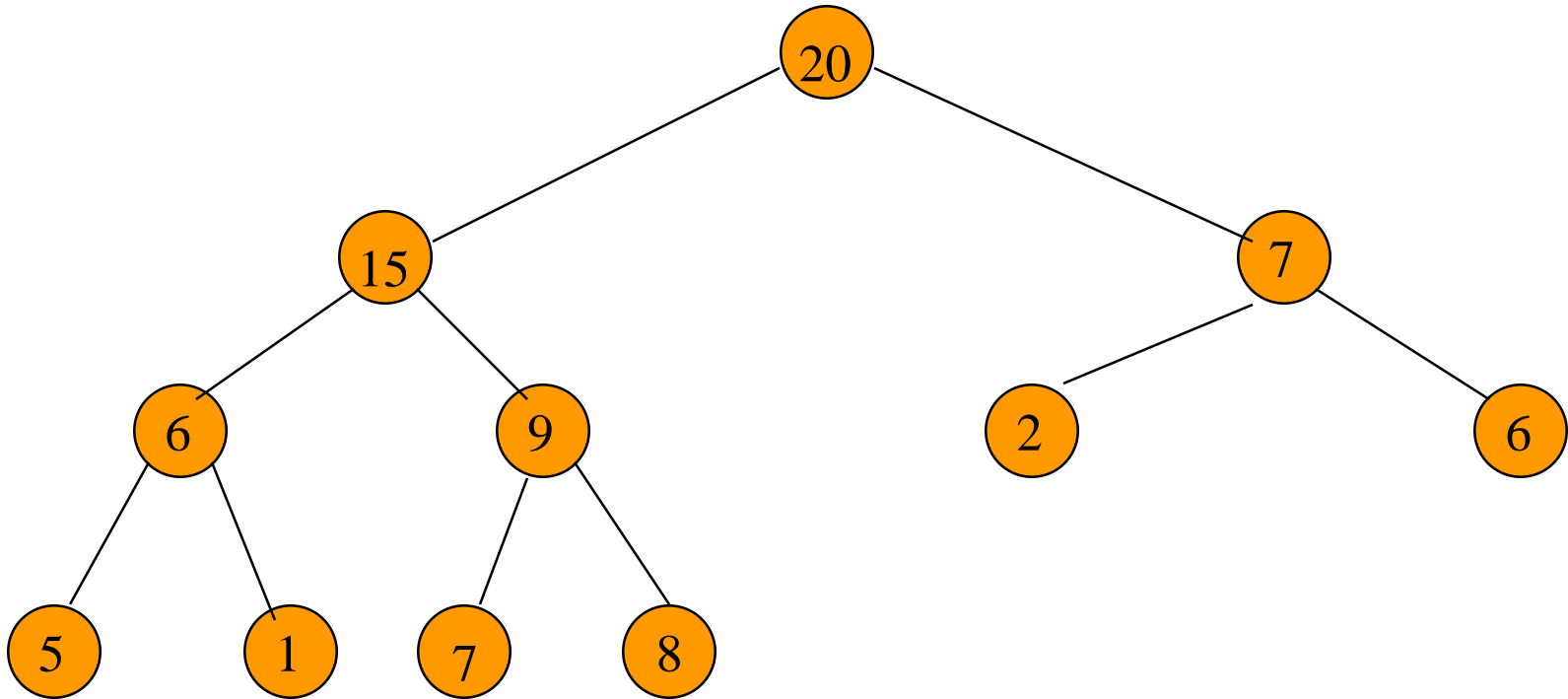
New element is 15.

Complexity Of Insert



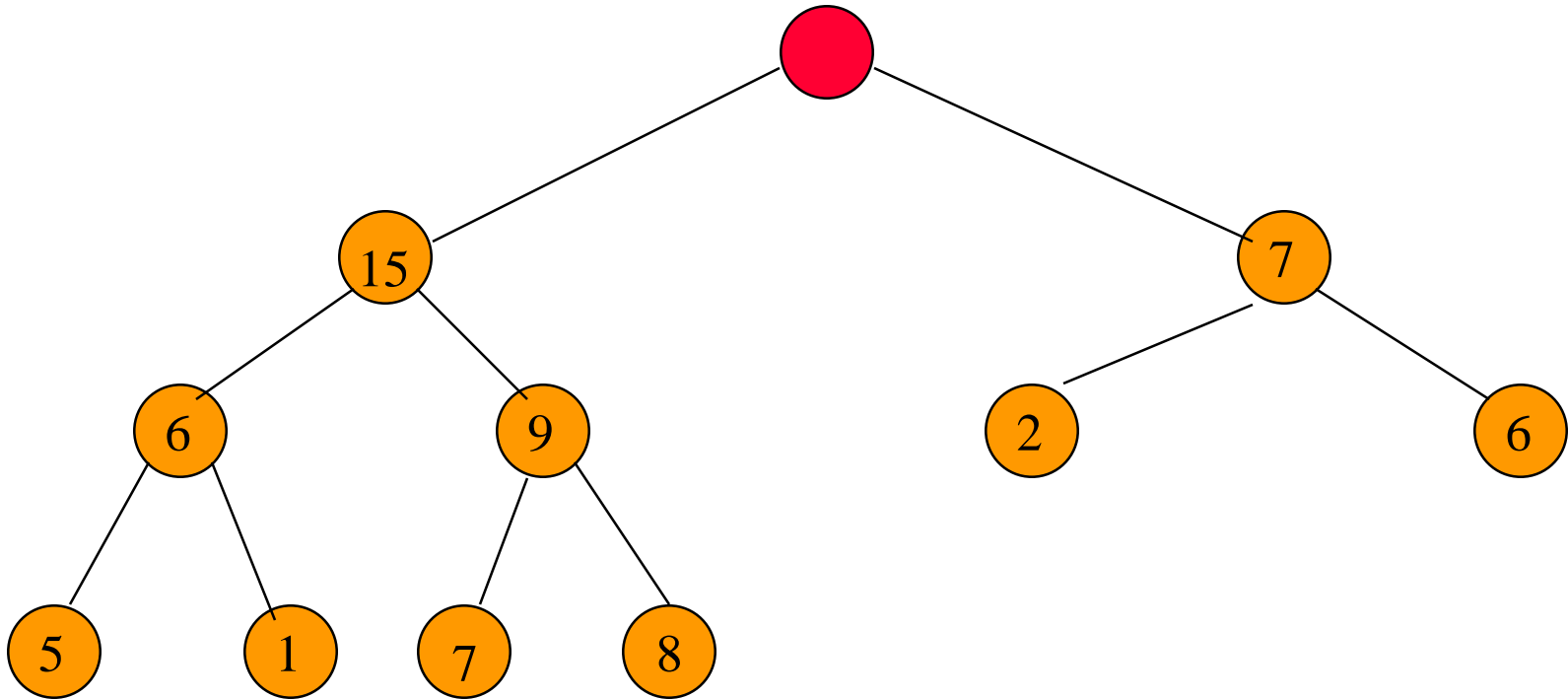
Complexity is $O(\log n)$, where n is heap size.

Removing The Max Element



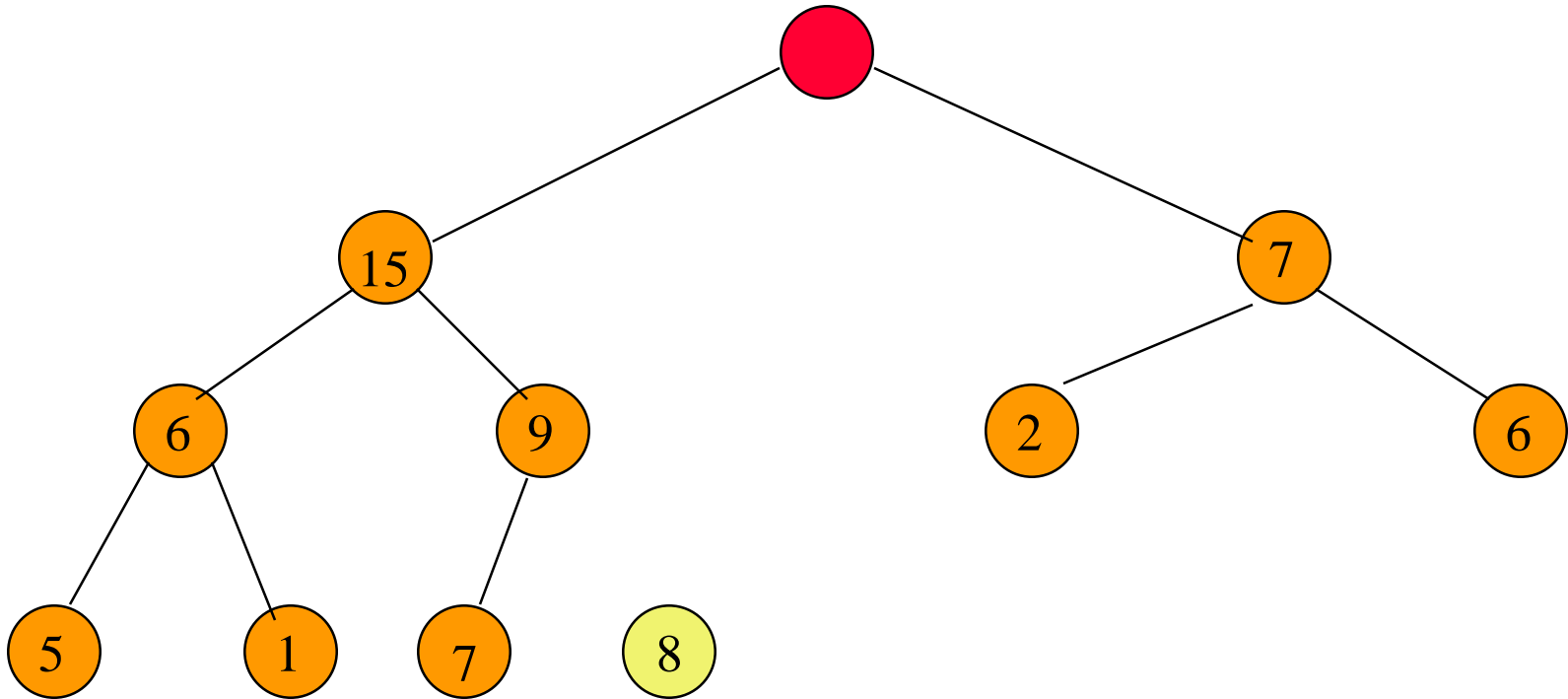
Max element is in the root.

Removing The Max Element



After max element is removed.

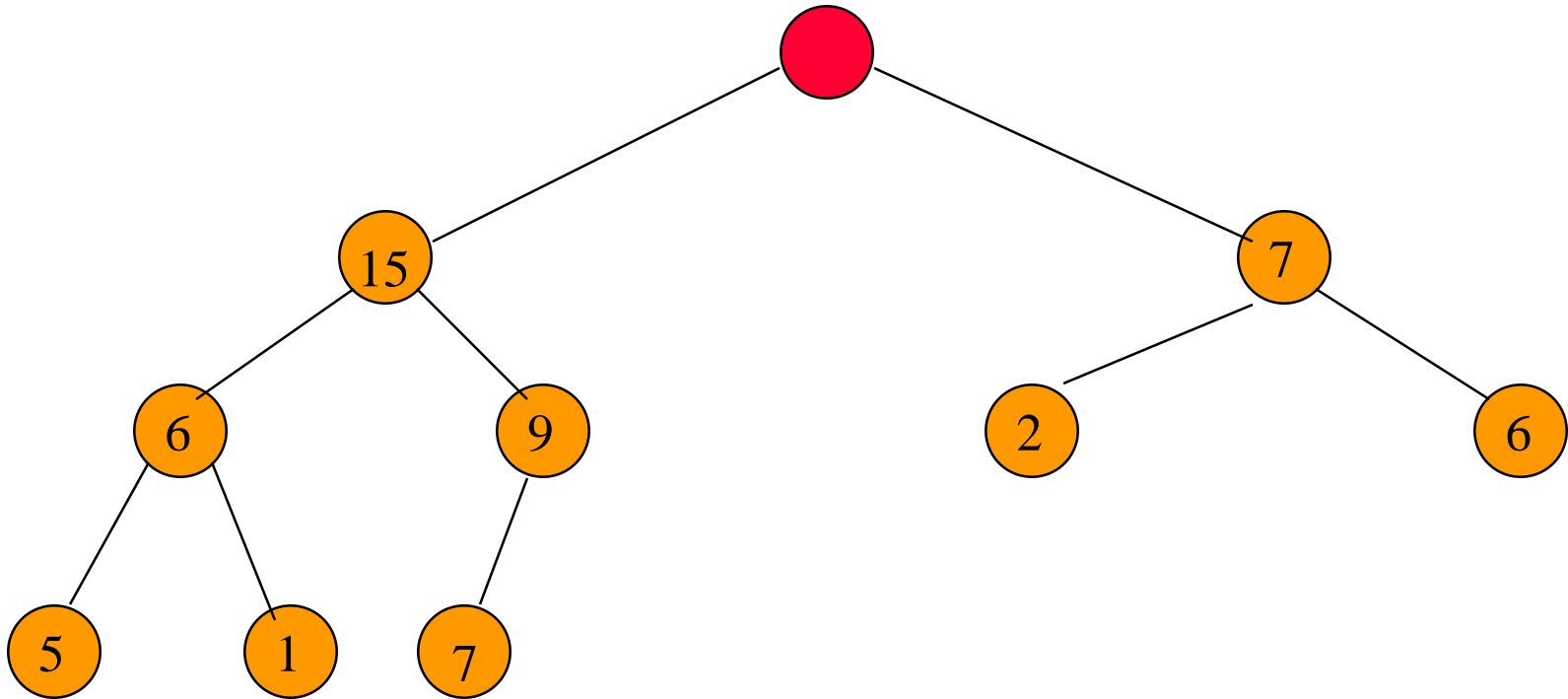
Removing The Max Element



Heap with 10 nodes.

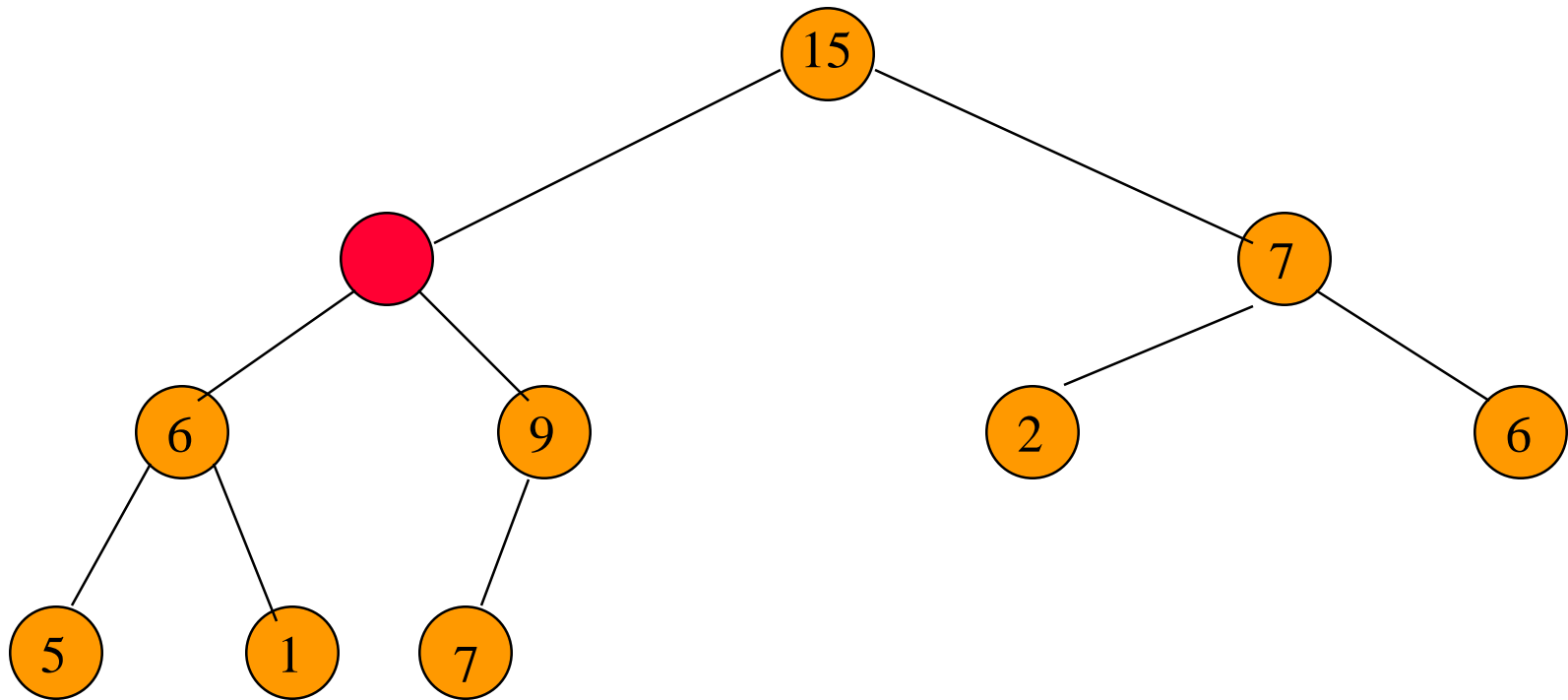
Reinsert 8 into the heap.

Removing The Max Element



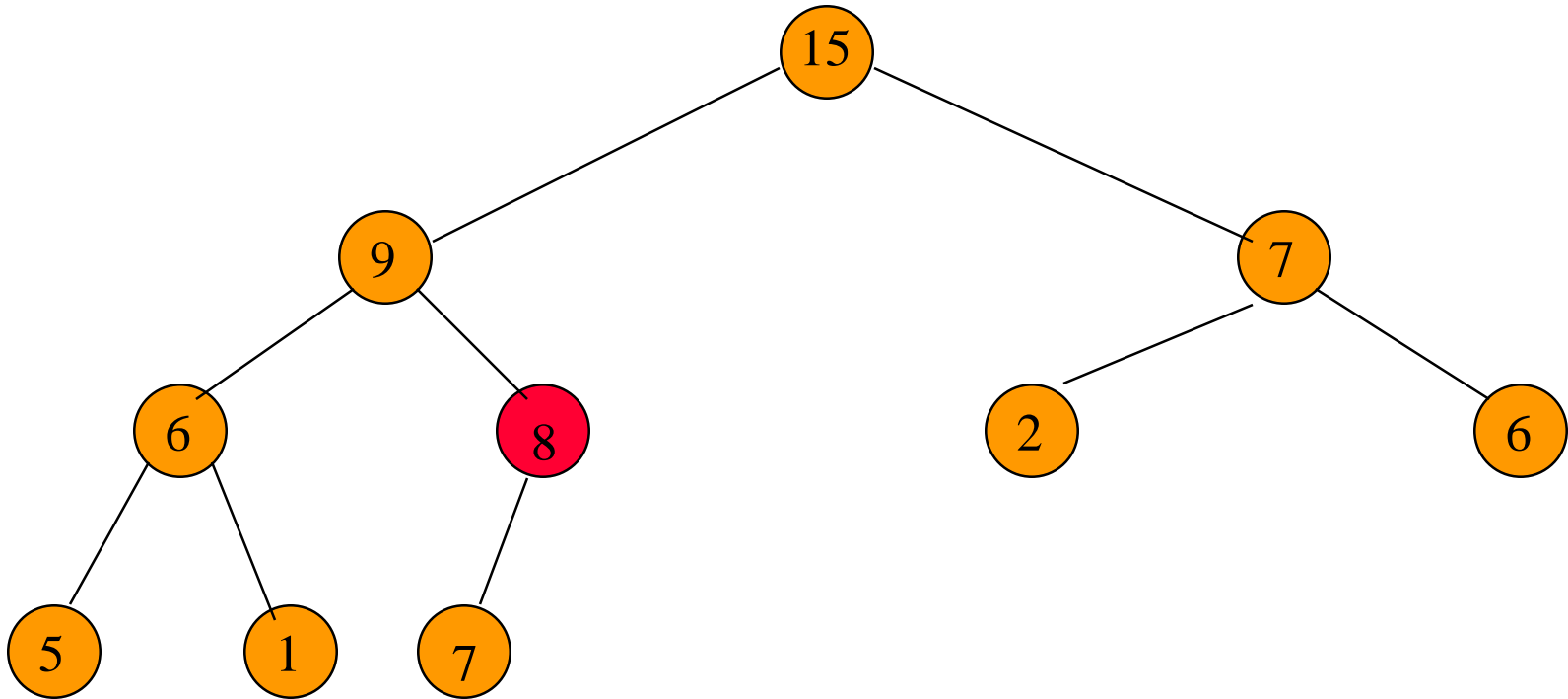
Reinsert **8** into the heap.

Removing The Max Element



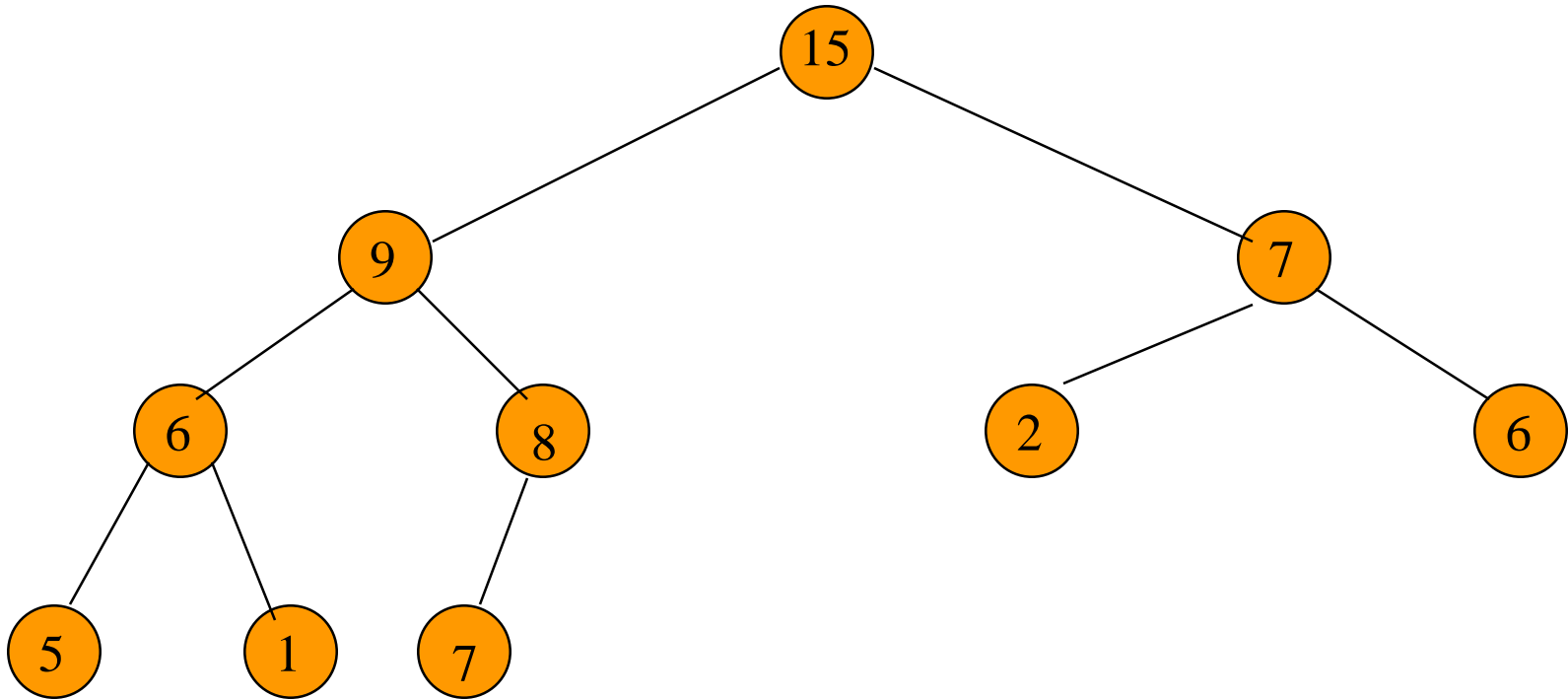
Reinsert **8** into the heap.

Removing The Max Element



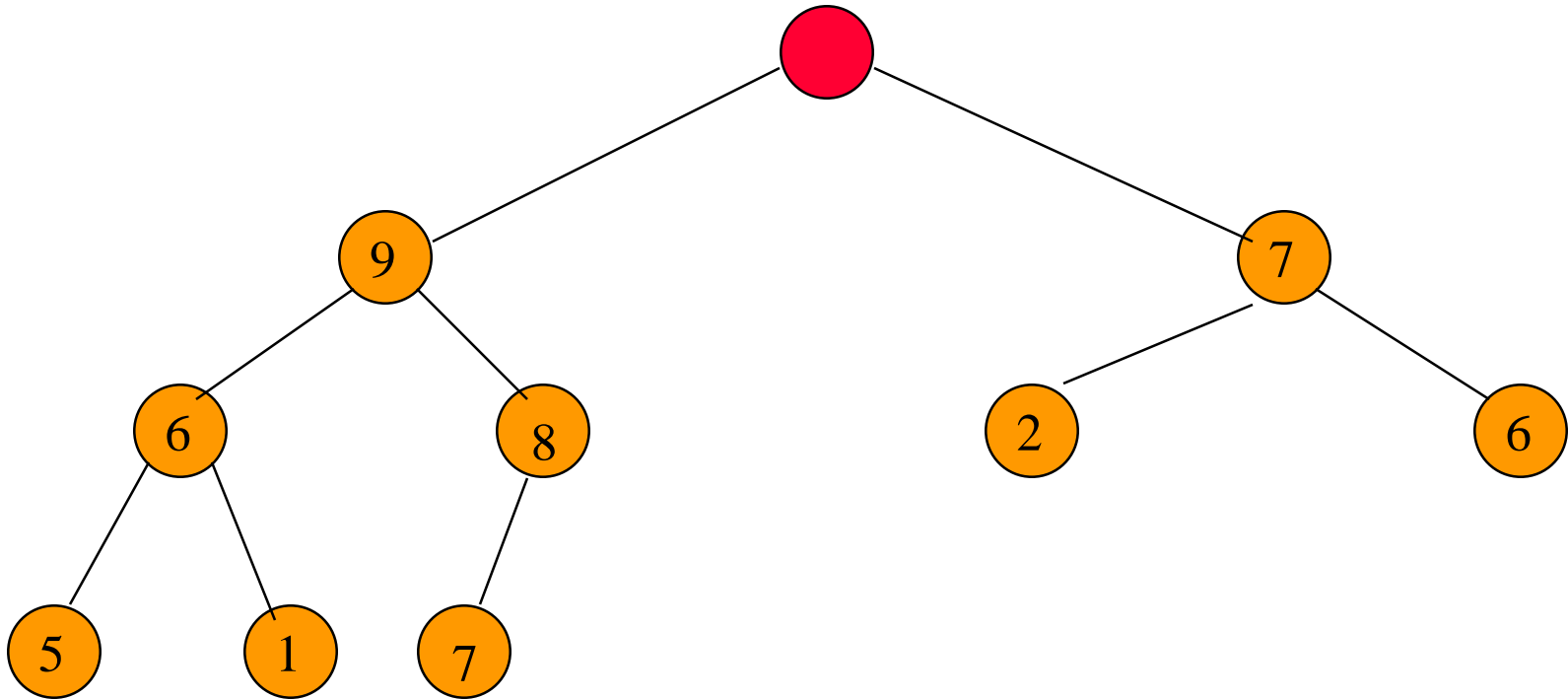
Reinsert **8** into the heap.

Removing The Max Element



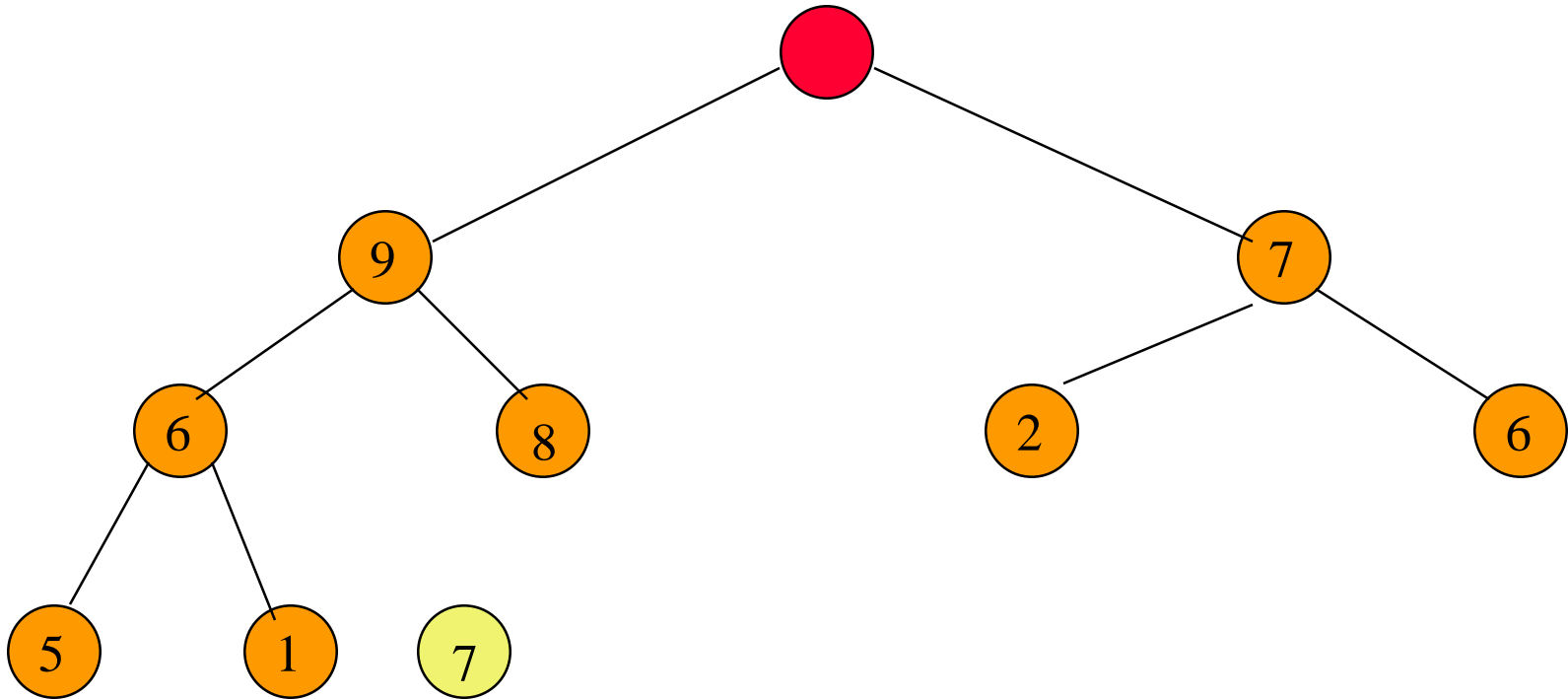
Max element is 15.

Removing The Max Element



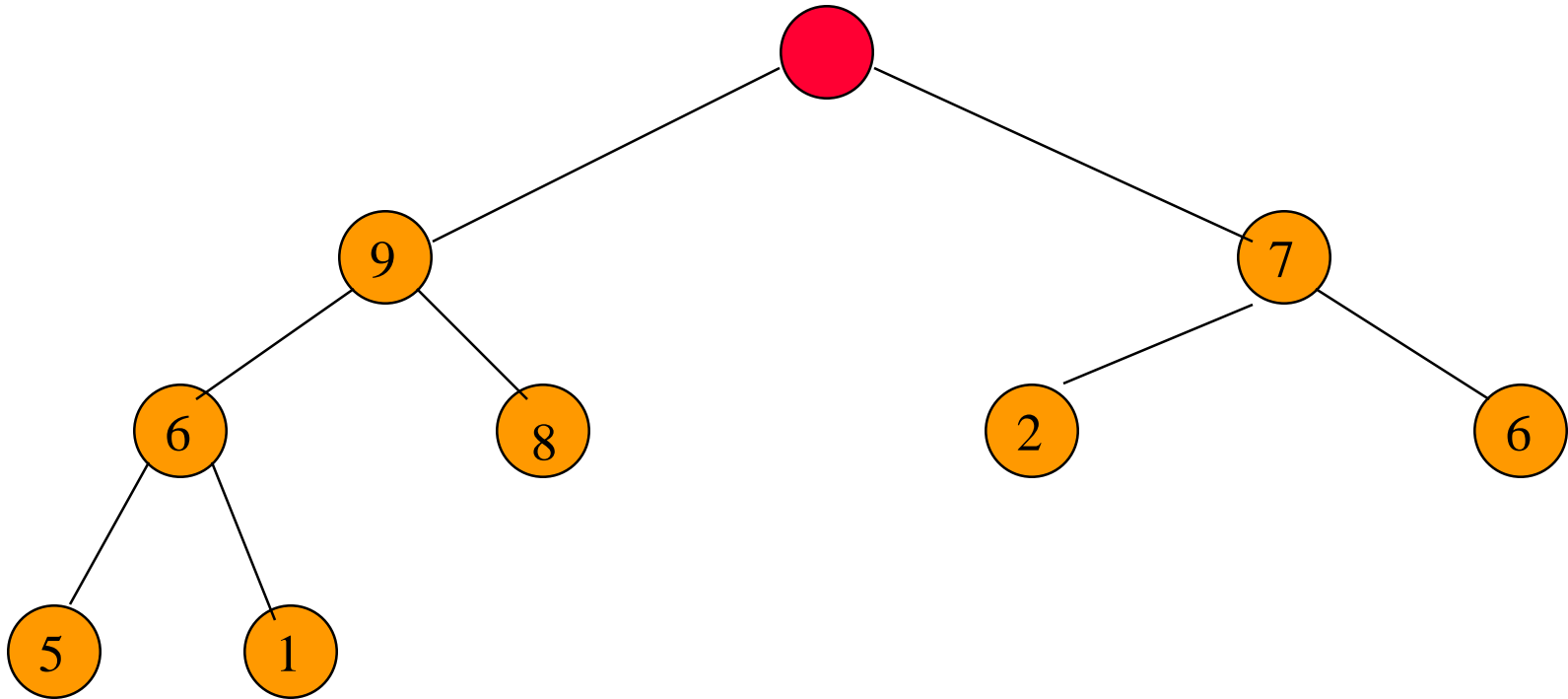
After max element is removed.

Removing The Max Element



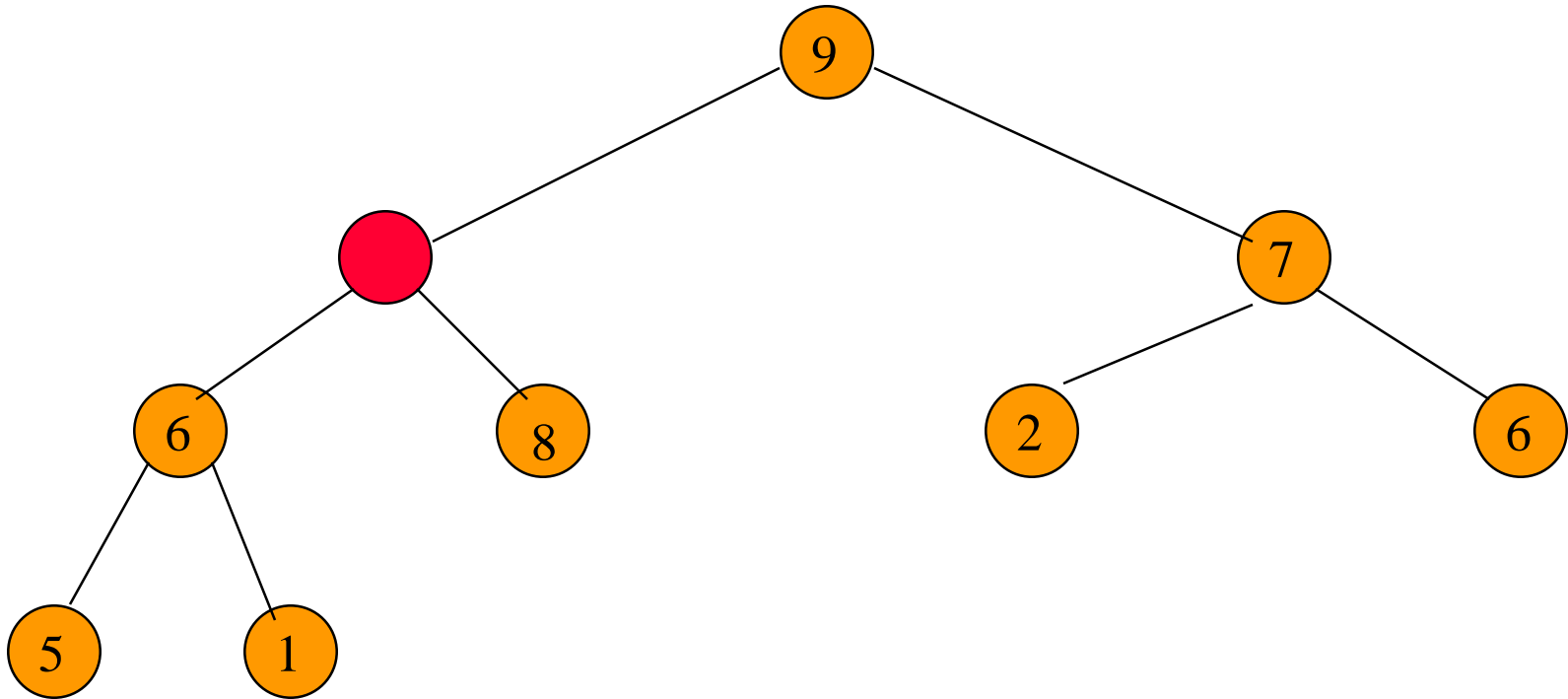
Heap with 9 nodes.

Removing The Max Element



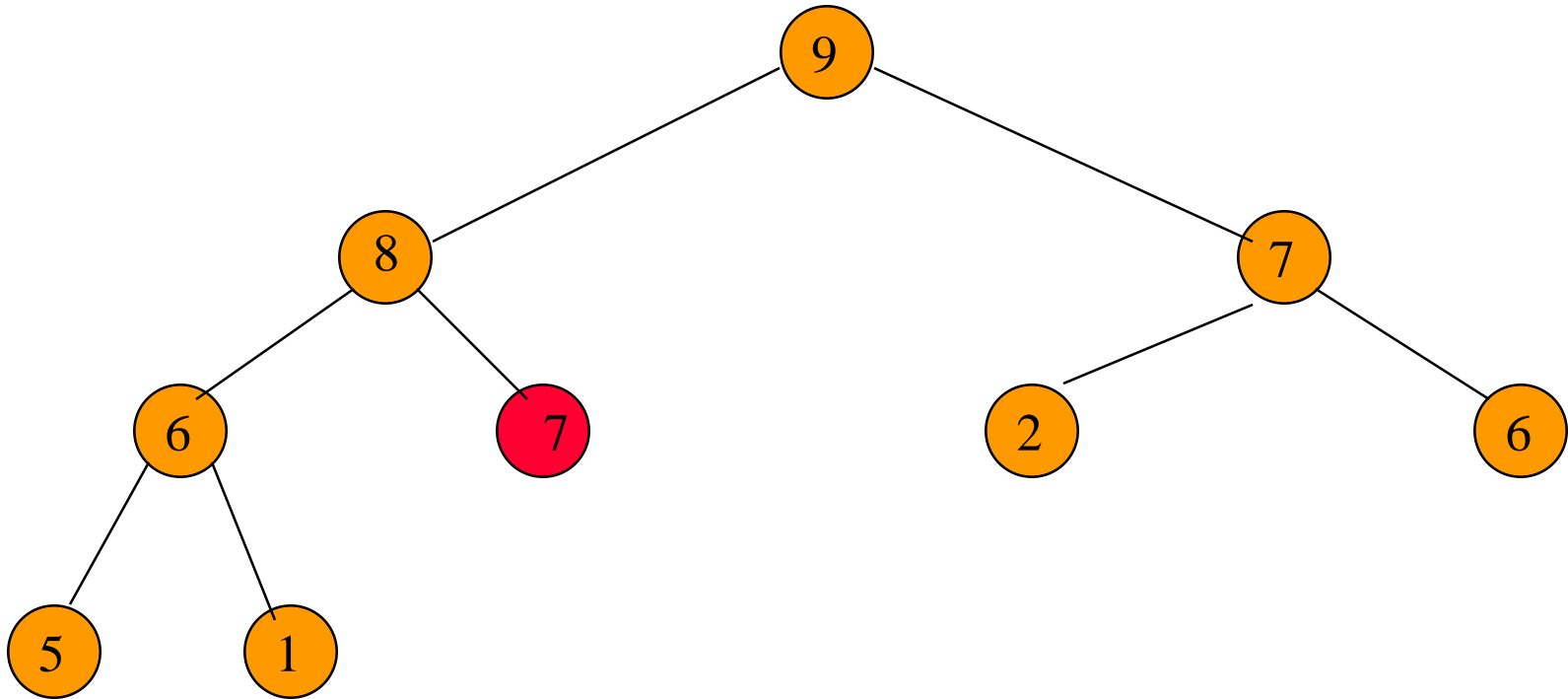
Reinsert **7**.

Removing The Max Element



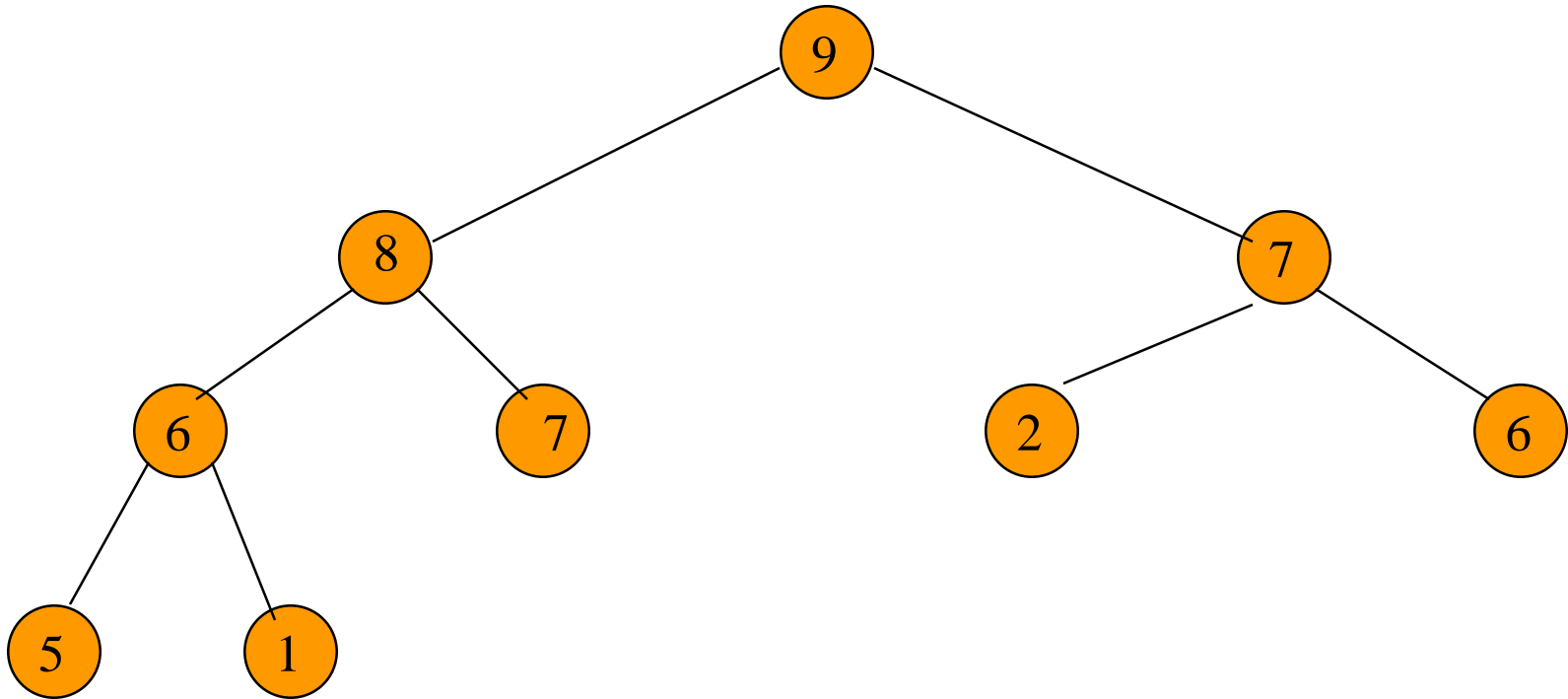
Reinsert **7**.

Removing The Max Element



Reinsert **7**.

Complexity Of Remove Max Element



Complexity is $O(\log n)$.

(Comparison again) Complexity Of Sorting

Sort n elements.

- n insert operations $\Rightarrow O(n \log n)$ time.
- n remove max operations $\Rightarrow O(n \log n)$ time.
- total time is $O(n \log n)$.
- ** compare with $O(n^2)$ for insertion sort of Lecture 1.

Machine Scheduling

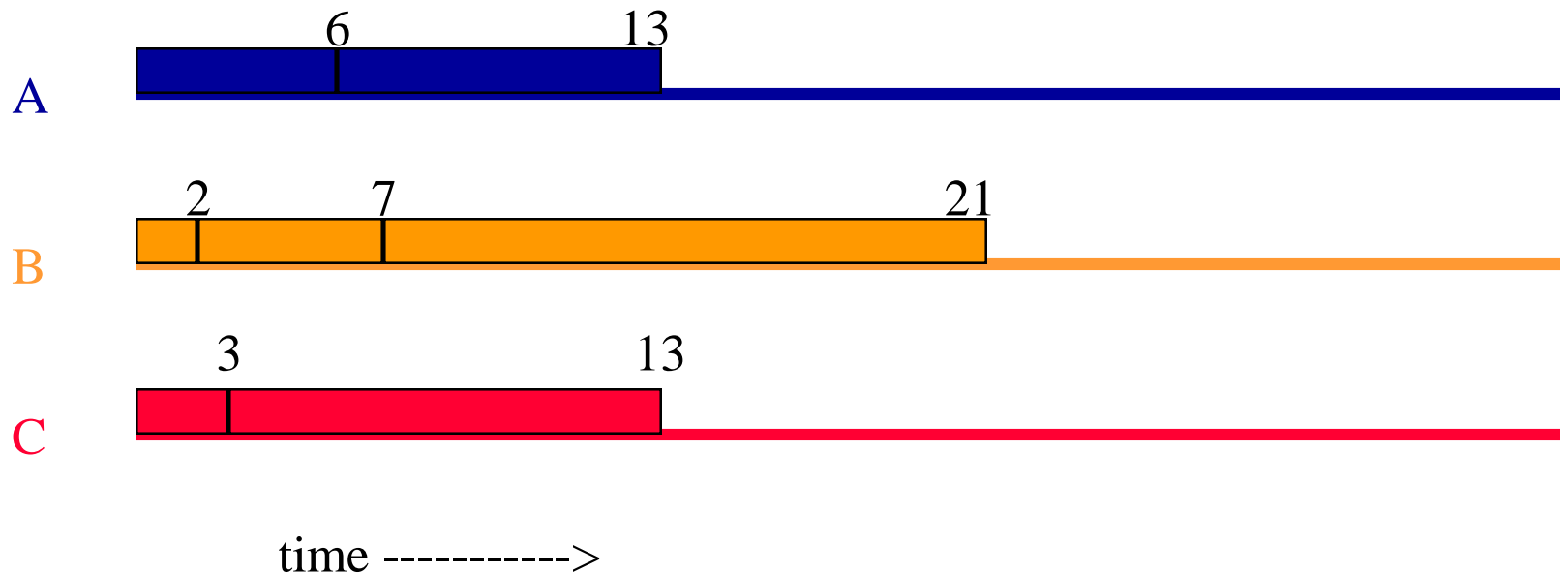
- m identical machines (drill press, cutter, sander, etc.)
- n jobs/tasks to be performed
- assign jobs to machines so that the time at which the last job completes is minimum

Machine Scheduling Example

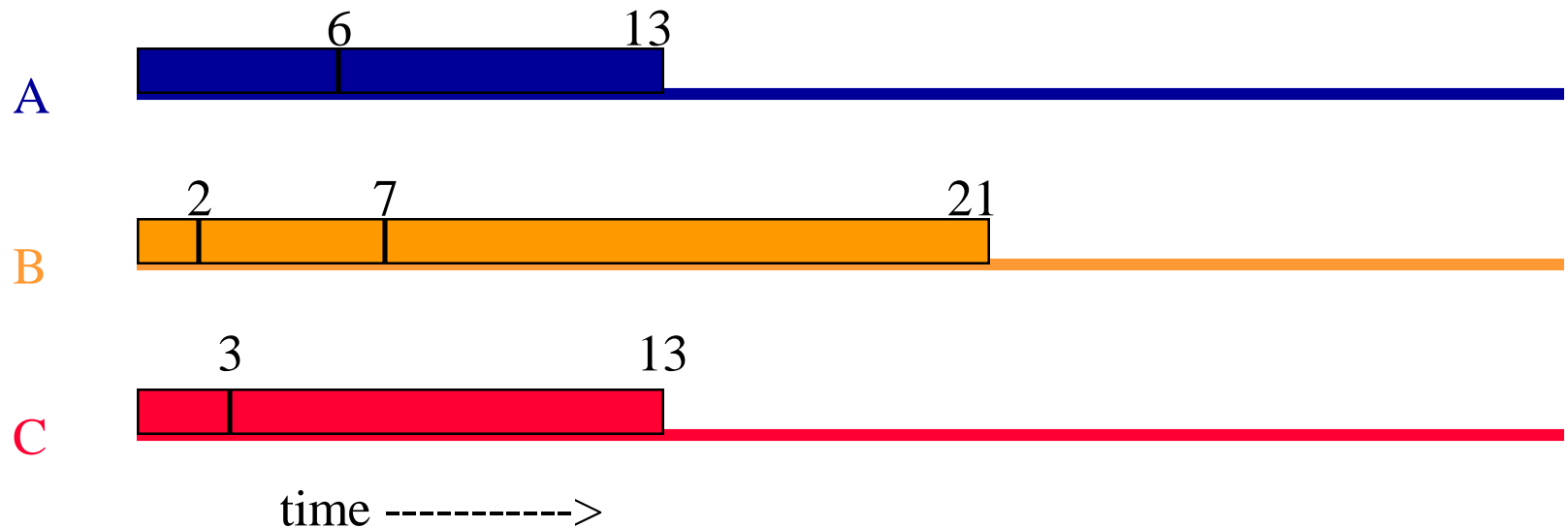
3 machines and 7 jobs

job times are [6, 2, 3, 5, 10, 7, 14]

possible schedule



Machine Scheduling Example



Finish time = 21

Objective: Find schedules with minimum finish time.

LPT Schedules

Longest Processing Time first.

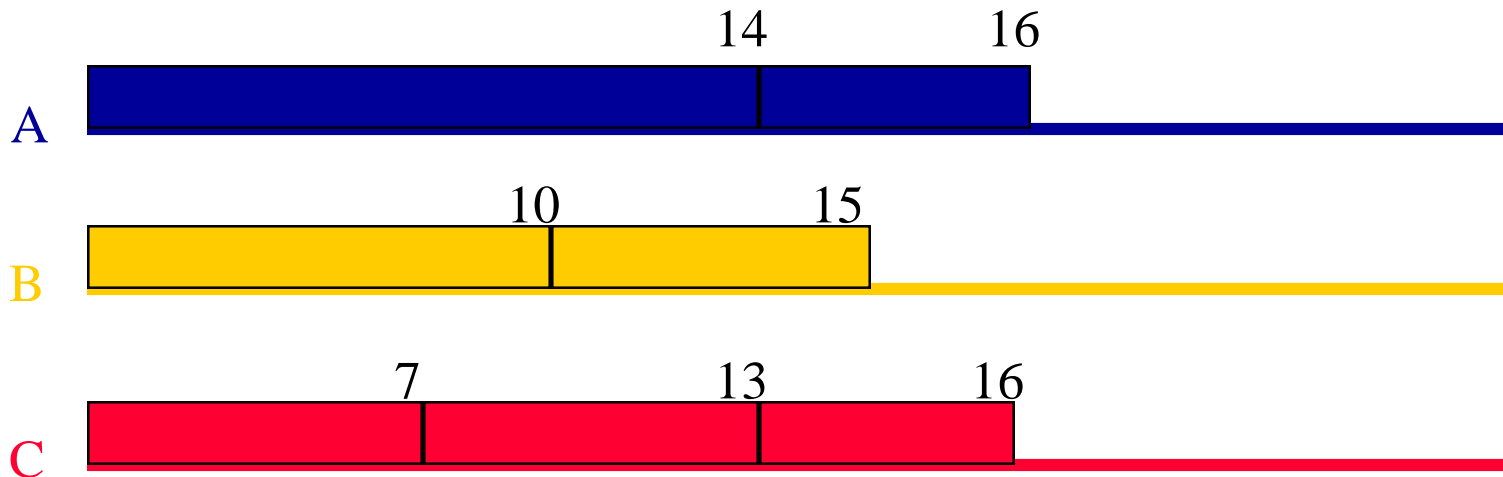
Jobs are scheduled in the order

14, 10, 7, 6, 5, 3, 2

Each job is scheduled on the machine on which it finishes earliest.

LPT Schedule

[14, 10, 7, 6, 5, 3, 2]



Finish time is 16!

LPT Schedule

- LPT rule does not guarantee minimum finish time schedules.
- $(\text{LPT Finish Time})/(\text{Minimum Finish Time}) \leq 4/3 - 1/(3m)$ where m is number of machines.
- Usually LPT finish time is much closer to minimum finish time.
- Minimum finish time scheduling is *NP-hard*.

NP-hard Problems

- Infamous class of problems for which no one has developed a polynomial time algorithm.
- That is, no algorithm whose complexity is $O(n^k)$ for any constant k is known for any NP-hard problem.
- The class includes thousands of real-world problems.
- Highly unlikely that any NP-hard problem can be solved by a polynomial time algorithm.

NP-hard Problems

- Since even polynomial time algorithms with degree $k > 3$ (say) are not practical for large n , we must change our expectations of the algorithm that is used.
- Usually develop *fast heuristics* for NP-hard problems.
 - Algorithm that gives a solution close to best.
 - Runs in acceptable amount of time.
- LPT rule is good heuristic for minimum finish time scheduling.

Complexity Of LPT Scheduling

- Sort jobs into decreasing order of task time.
 - $O(n \log n)$ time (n is number of jobs)
- Schedule jobs in this order.
 - assign job to machine that becomes available first
 - must find minimum of m (m is number of machines) finish times
 - takes $O(m)$ time using simple strategy
 - so need $O(mn)$ time to schedule all n jobs.

Using A Min Priority Queue

- Min priority queue has the finish times of the **m** machines.
- Initial finish times are all **0**.
- To schedule a job remove machine with minimum finish time from the priority queue.
- Update the finish time of the selected machine and insert the machine back into the priority queue.

Using A Min Priority Queue

- m put operations to initialize priority queue
- 1 remove min and 1 insert to schedule each job
- each insert and remove min operation takes $O(\log m)$ time
- time to schedule is $O(n \log m)$
- overall time is

$$O(n \log n + n \log m) = O(n \log (mn))$$

Huffman Codes

Discussed in Chapter 7 of text.