APA 254
Data Structures

Lecture 9.1
(Priority Queue and Heap)

Dept. of Information System
Hanyang University

Priority Queues

- A priority queue is a collection of zero or more elements → each element has a **priority** or value
- Unlike the FIFO queues, the order of deletion from a priority queue (e.g., who gets served next) is determined by the element priority
- Elements are deleted by increasing or decreasing order of priority rather than by the order in which they arrived in the queue

Priority Queues

- Operations performed on priority queues

  1) Find an element, 2) insert a new element,
  3) delete an element, etc.
- Two kinds of (Min, Max) priority queues exist
- In a Min priority queue, find/delete operation finds/deletes the element with minimum priority
- In a Max priority queue, find/delete operation finds/deletes the element with maximum priority
- Two or more elements can have the same priority

Priority Queues

- See ADT 12.1 & Program 12.1 for max priority queue specification
- What would be different for min priority queue specification?
- Read Examples 12.1, 12.2
- What are other examples in our daily lives that utilize the priority queue concept?

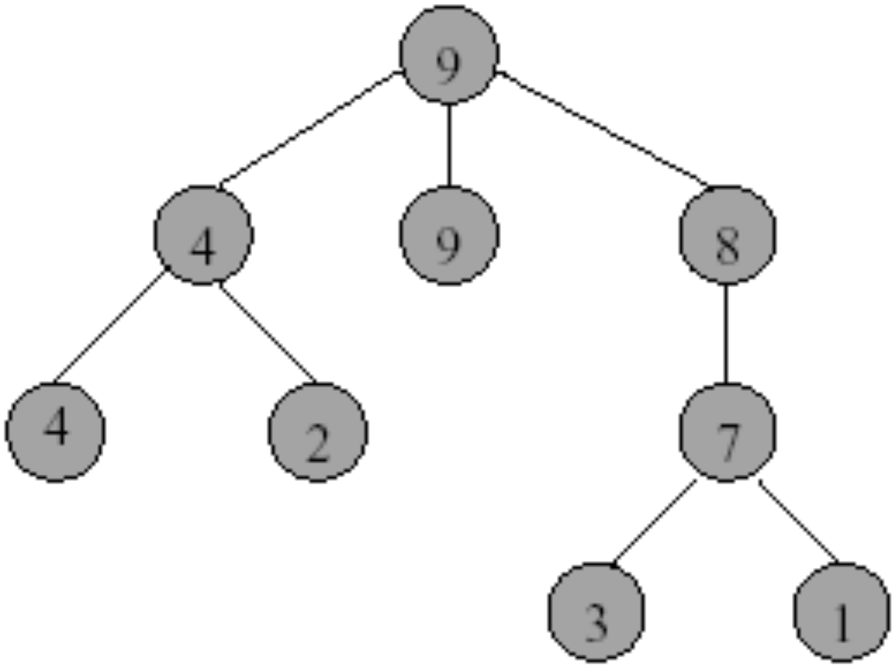
Implementation of Priority Queues

- Implemented using **heaps** and **leftist trees**
- **Heap** is a complete binary tree that is efficiently stored using the array-based representation
- **Leftist tree** is a linked data structure suitable for the implementation of a priority queue

Max (Min) Tree

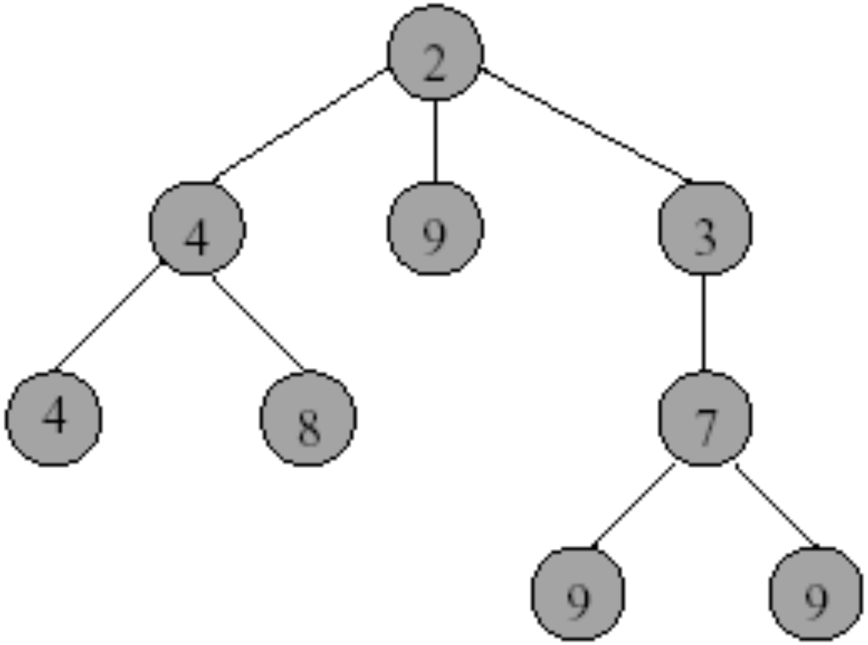
- A max tree (min tree) is a tree in which the value in each node is greater (less) than or equal to those in its children (if any)
  - See Figure 12.1, 12.2 for examples
  - Nodes of a max or min tree may have more than two children (i.e., may not be binary tree)

Max Tree Example



Root has maximum element.

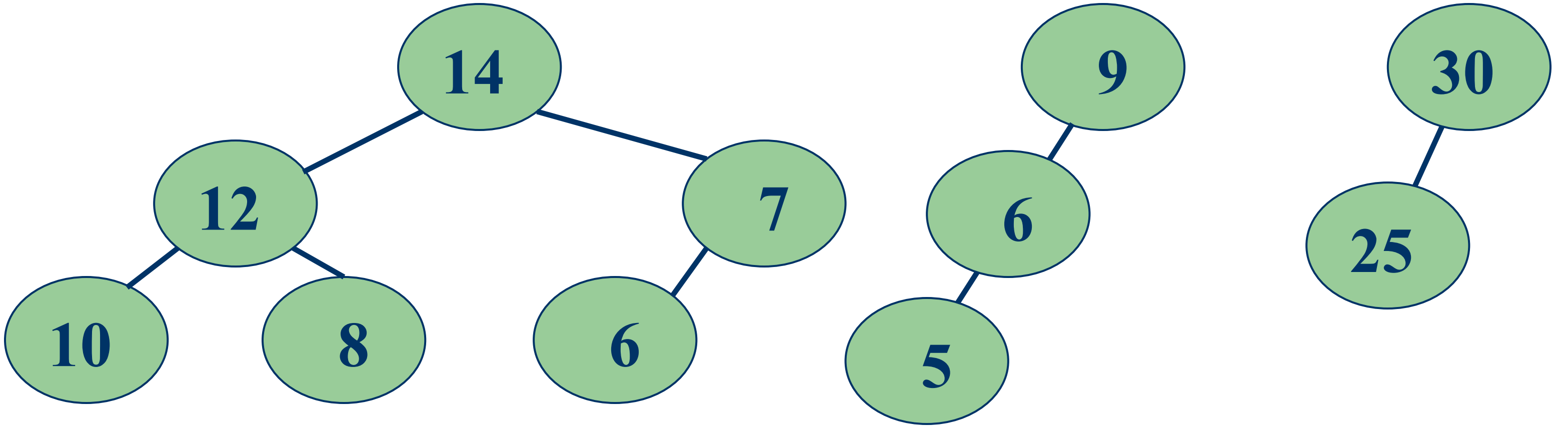
Min Tree Example



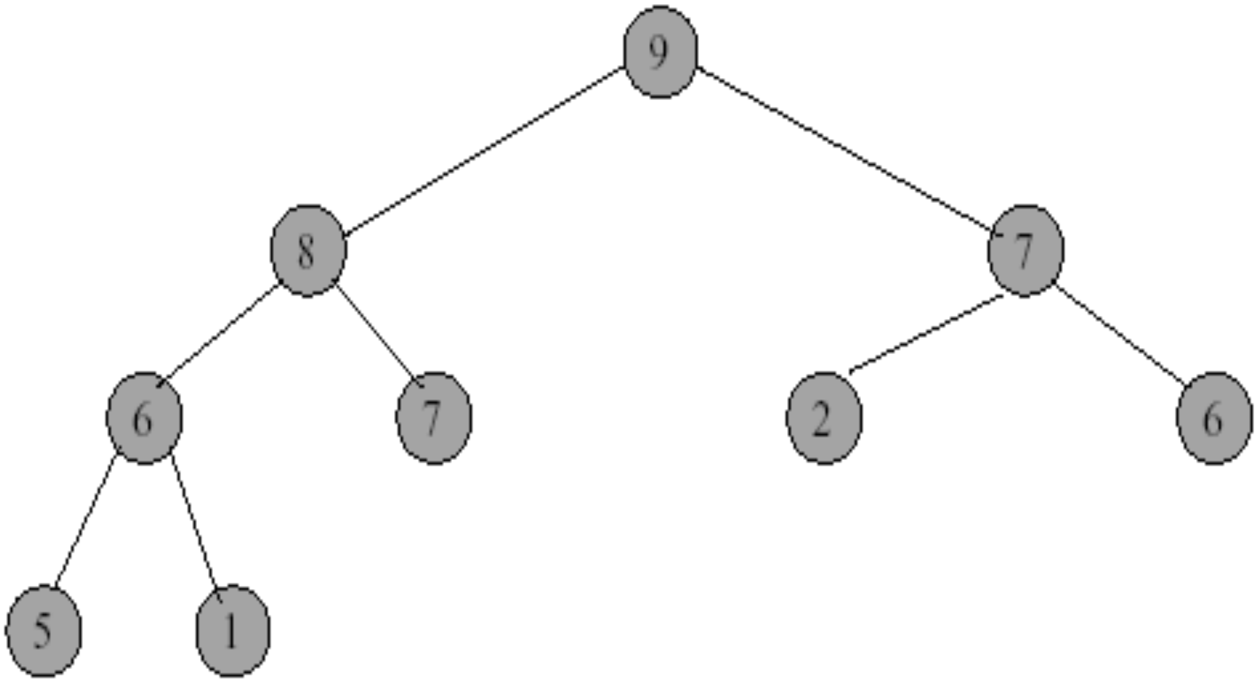
Root has minimum element.

Heaps - Definitions

- A max heap (min heap) is a max (min) tree that is also a complete binary tree
  - Figure 12.1 (a) & (c) are max heap
  - Figure 12.2 (a) & (c) are min heap
  - Pg. 427: definition of a complete binary tree

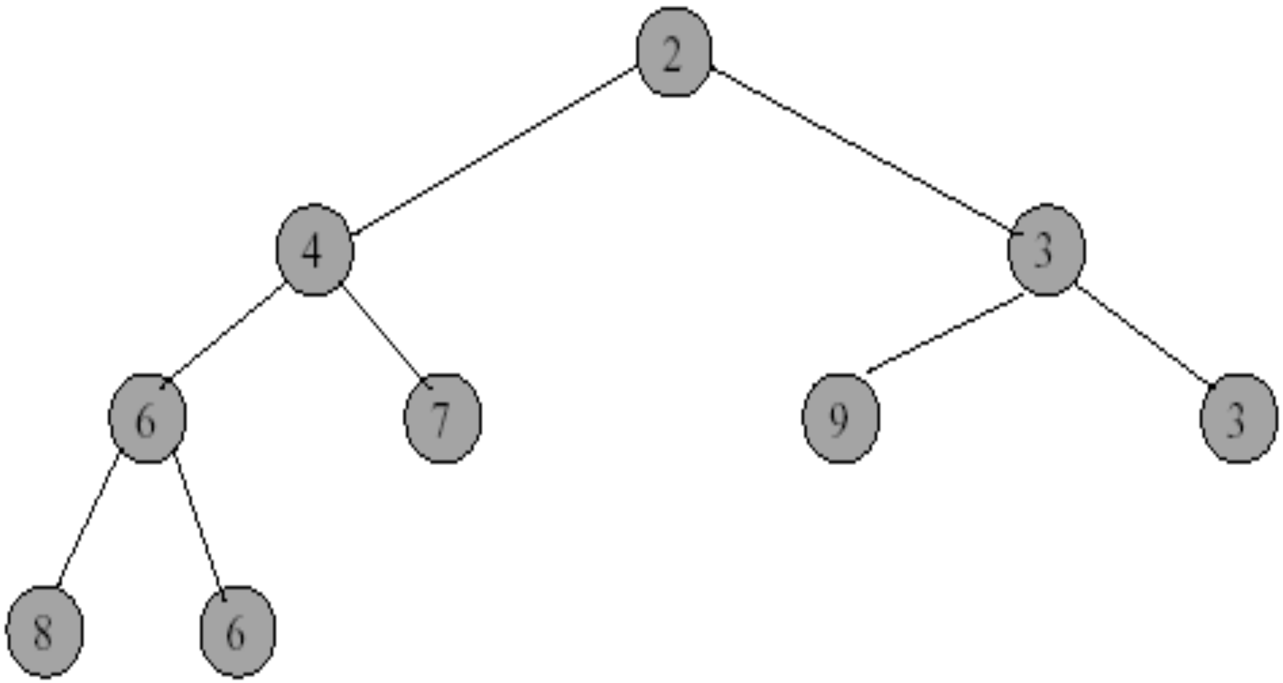


Max Heap with 9 Nodes



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

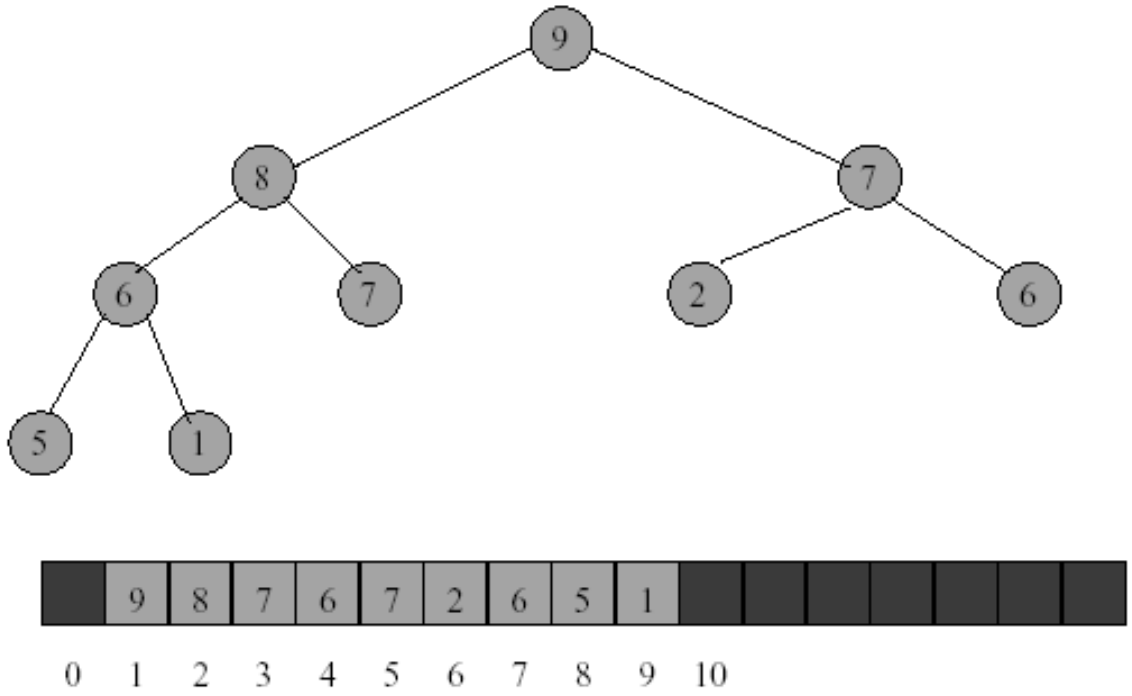
Min Heap with 9 Nodes



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

Array Representation of Heap

- A heap is efficiently represented as an array.

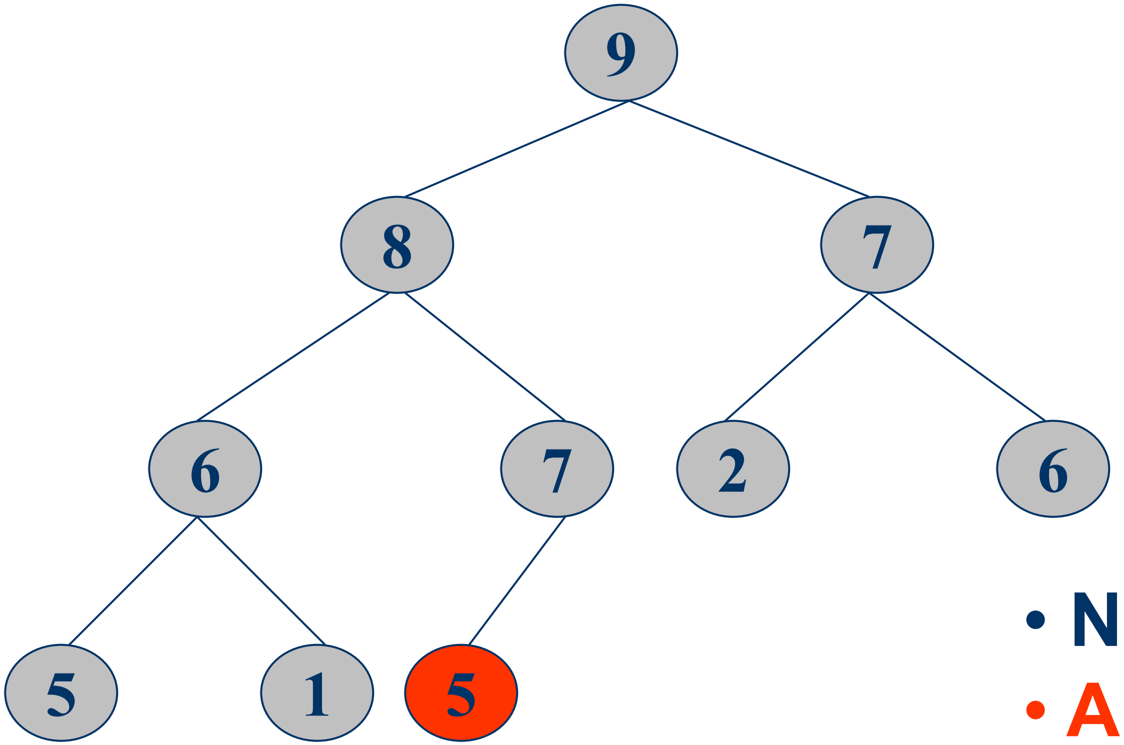


Heap Operations

When *n* is the number of elements (heap size),

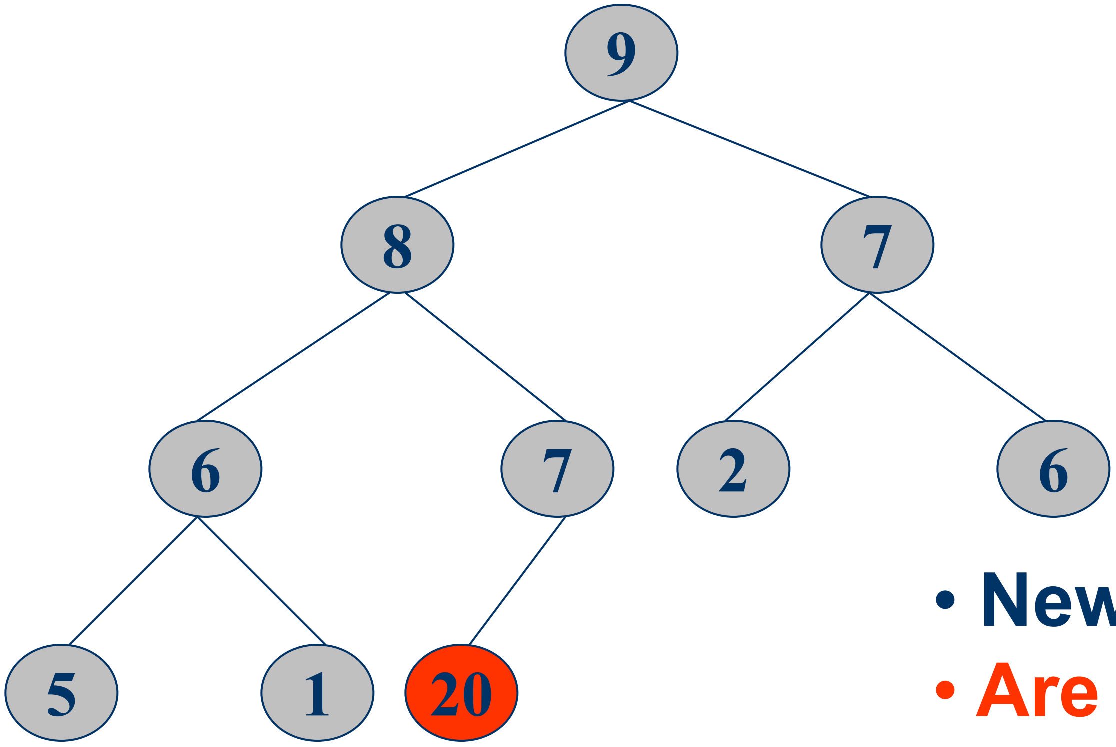
- Insertion → $O(\log_2 n)$
- Deletion → $O(\log_2 n)$
- Initialization → $O(n)$

Insertion into a Max Heap



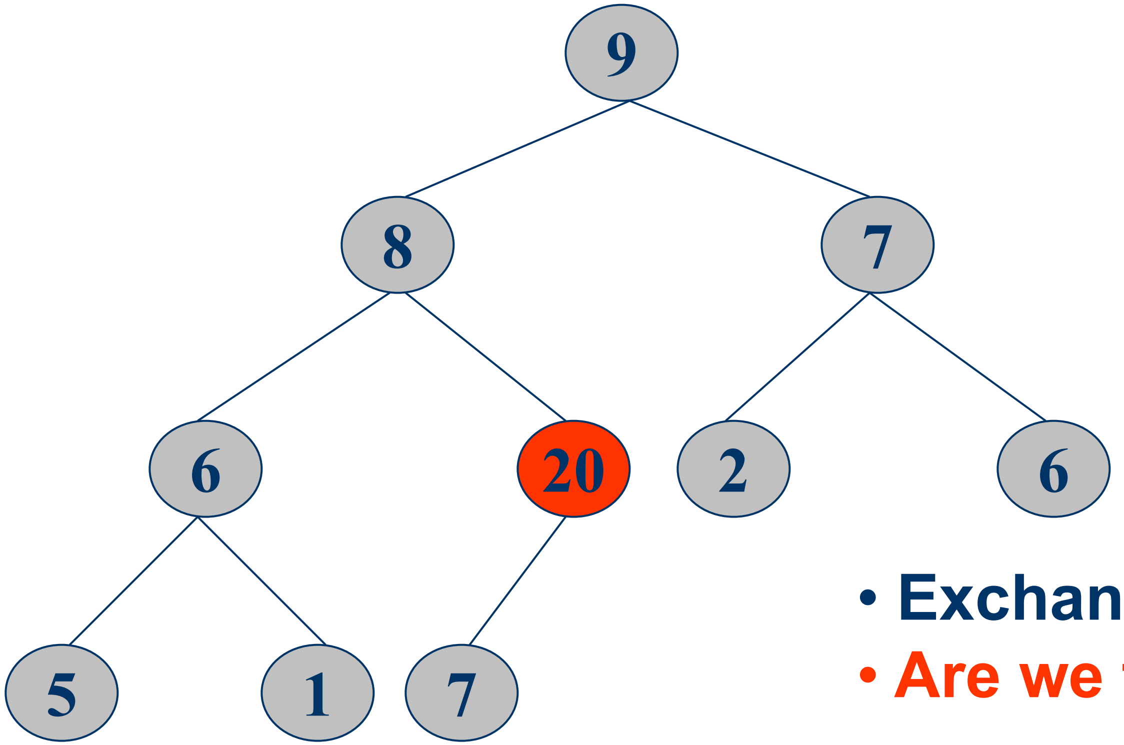
- New element is 5
- Are we finished?

Insertion into a Max Heap



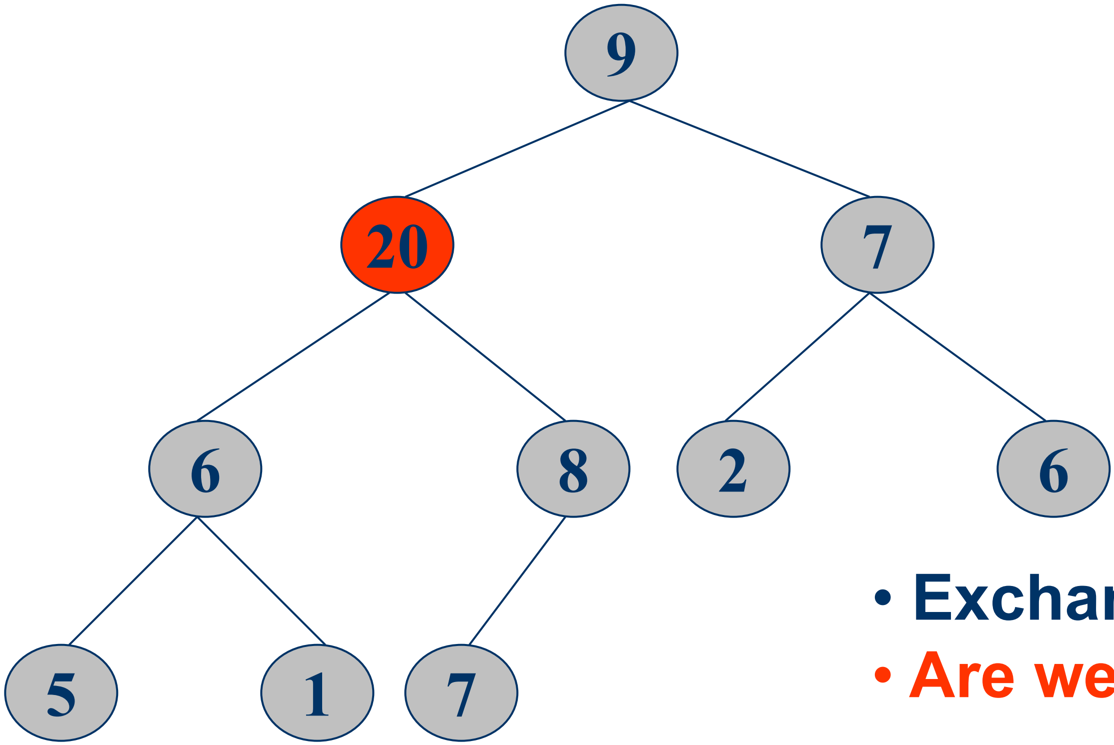
- New element is 20
- Are we finished?

Insertion into a Max Heap



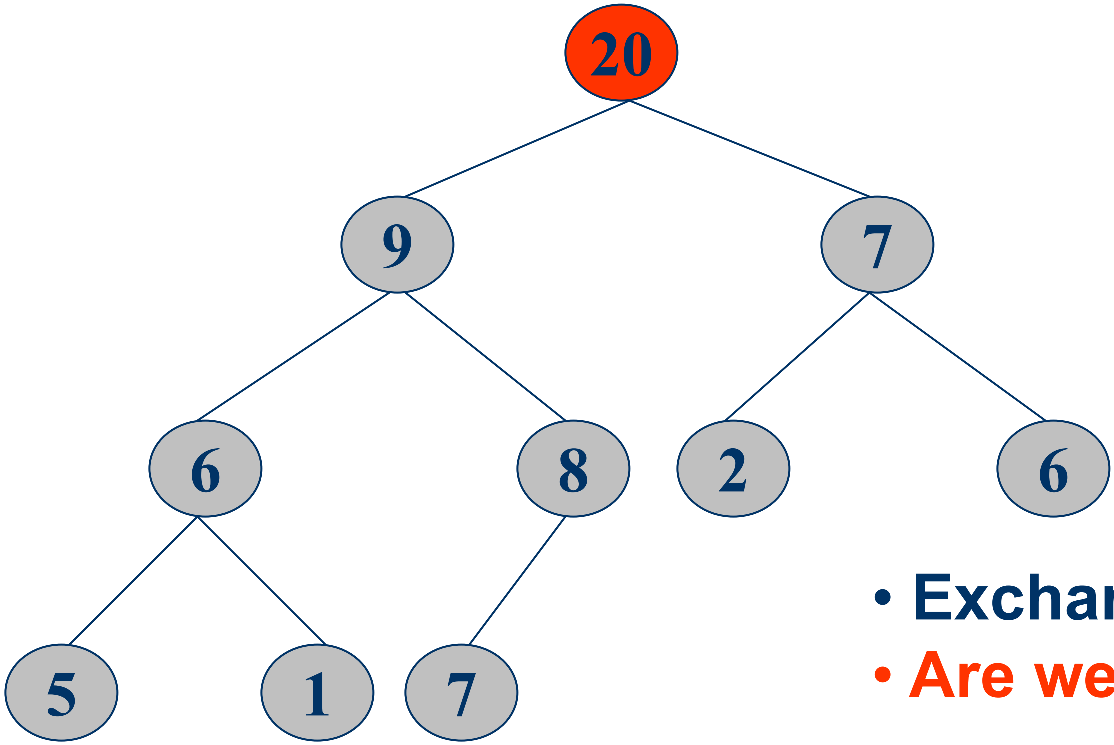
- Exchange the positions with 7
- Are we finished?

Insertion into a Max Heap



- Exchange the positions with 8
- Are we finished?

Insertion into a Max Heap



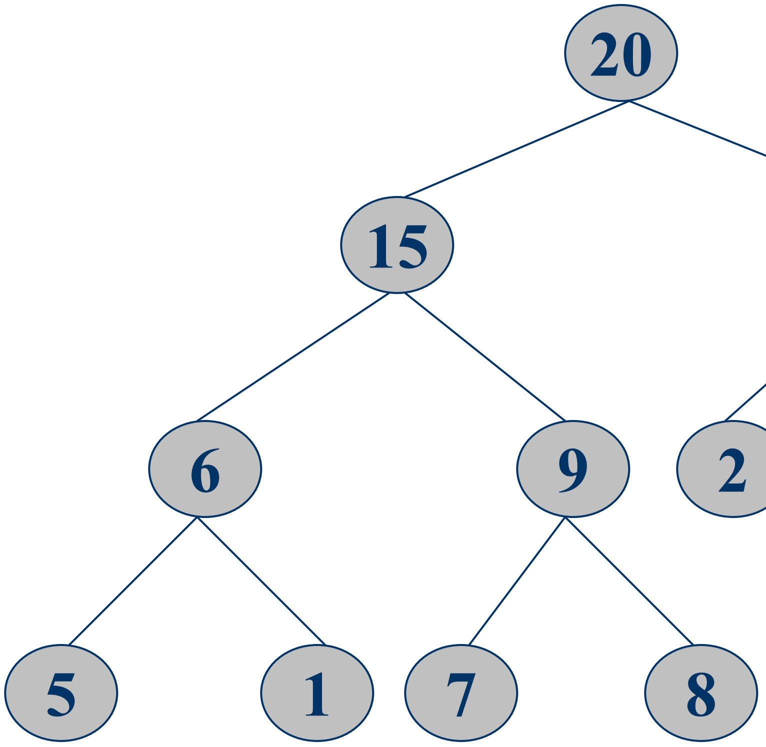
- Exchange the positions with 9
- Are we finished?

Complexity of Insertion

- See also Figure 12.3 for another insertion example

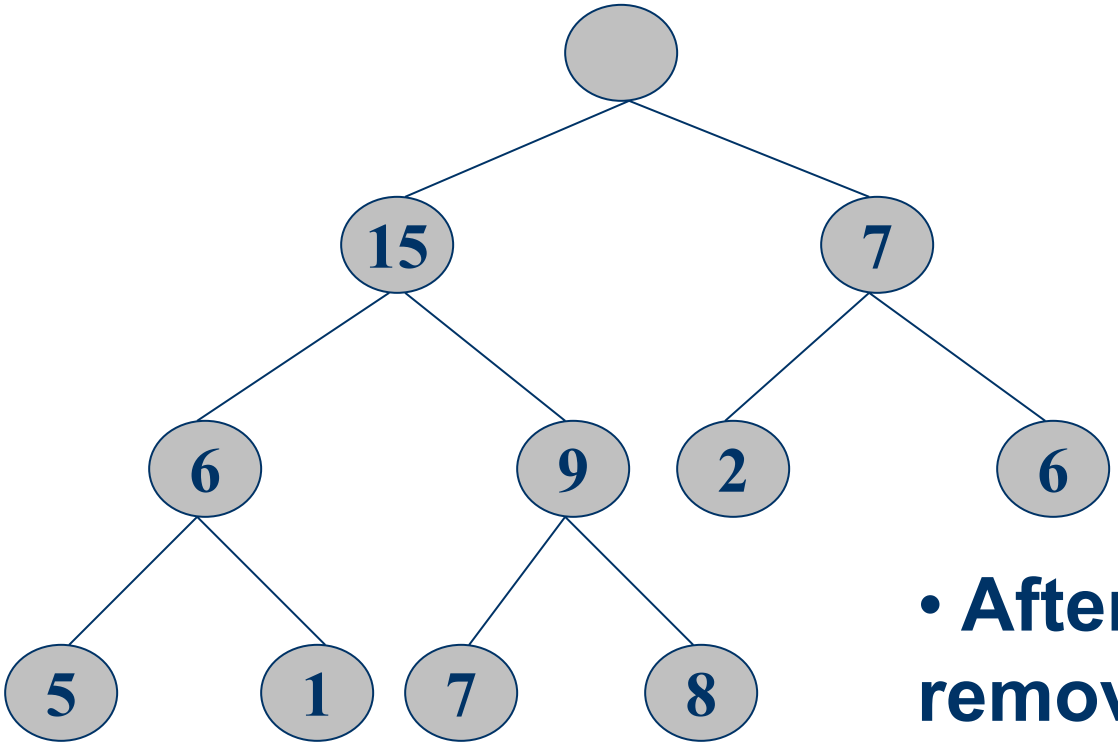
- At each level, we do $\Theta(1)$ work
- Thus the time complexity is **O(height) = O($\boldsymbol{log_2 n}$)**, where *n* is the heap size

Deletion from a Max Heap



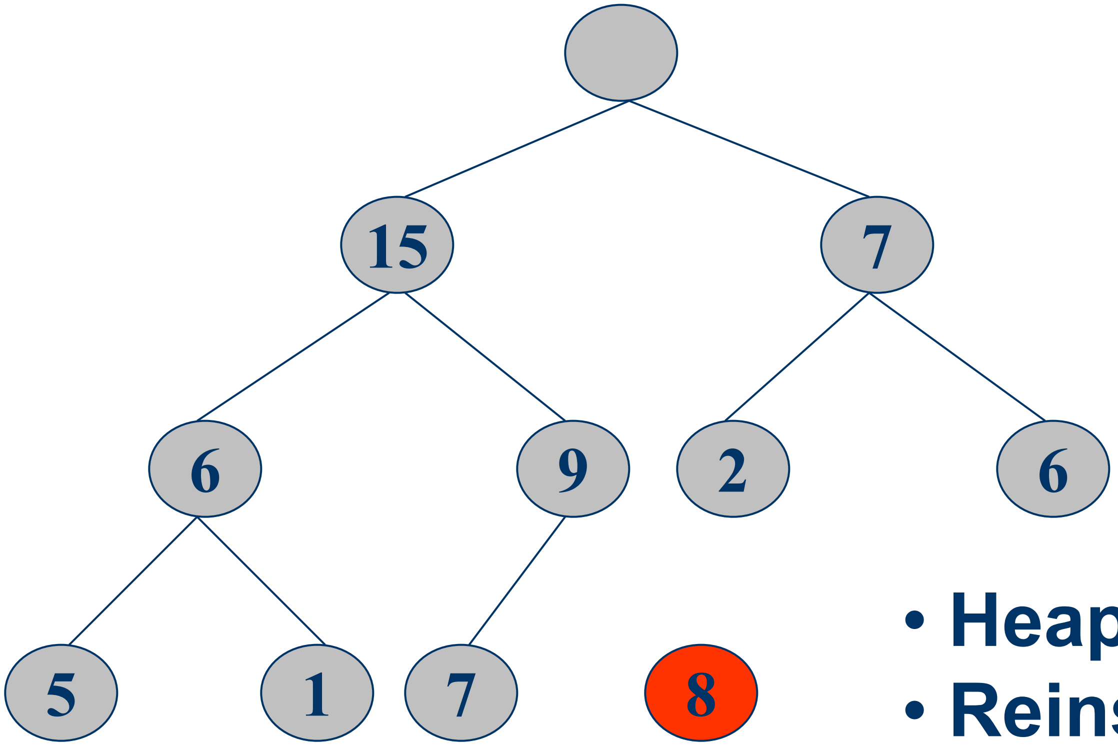
- Max element is in the root
- What happens when we delete an element?

Deletion from a Max Heap



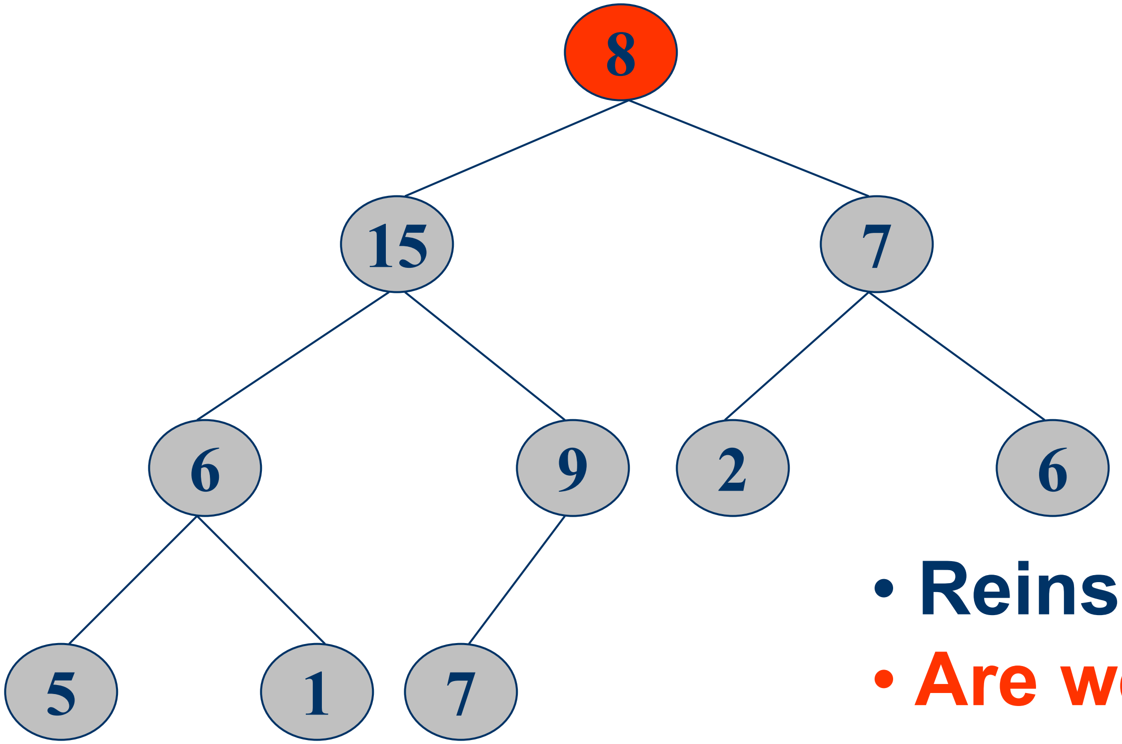
- After the max element is removed.
- Are we finished?

Deletion from a Max Heap



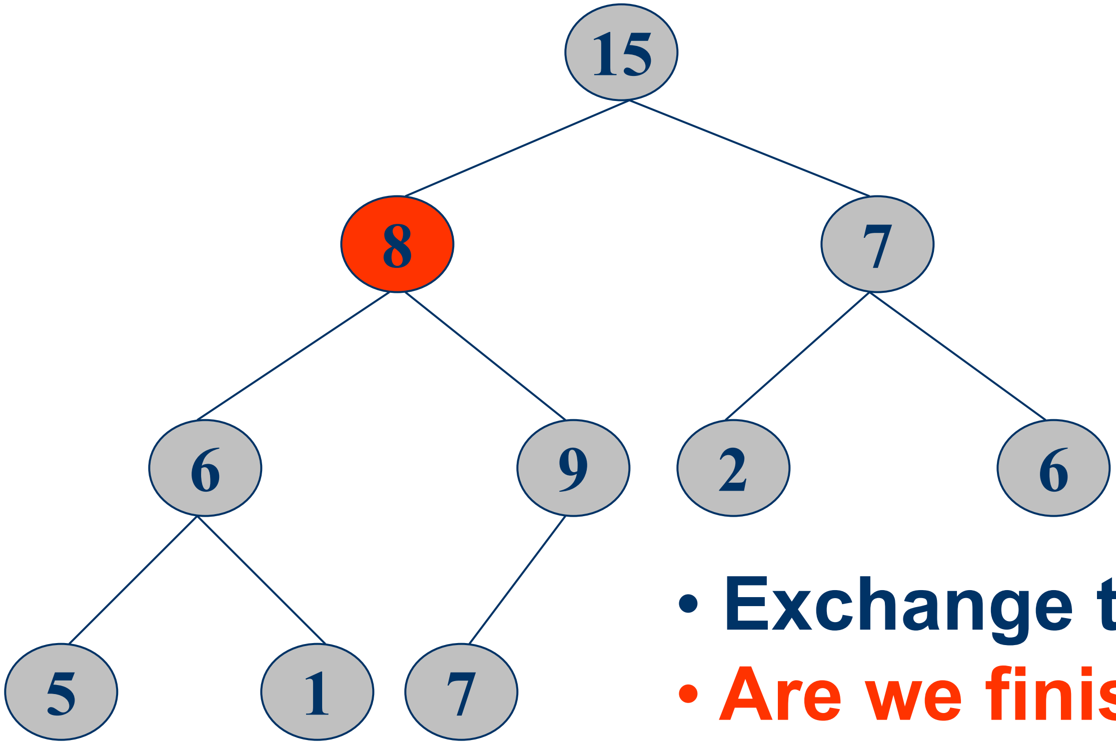
- Heap with 10 nodes.
- Reinsert 8 into the heap.

Deletion from a Max Heap



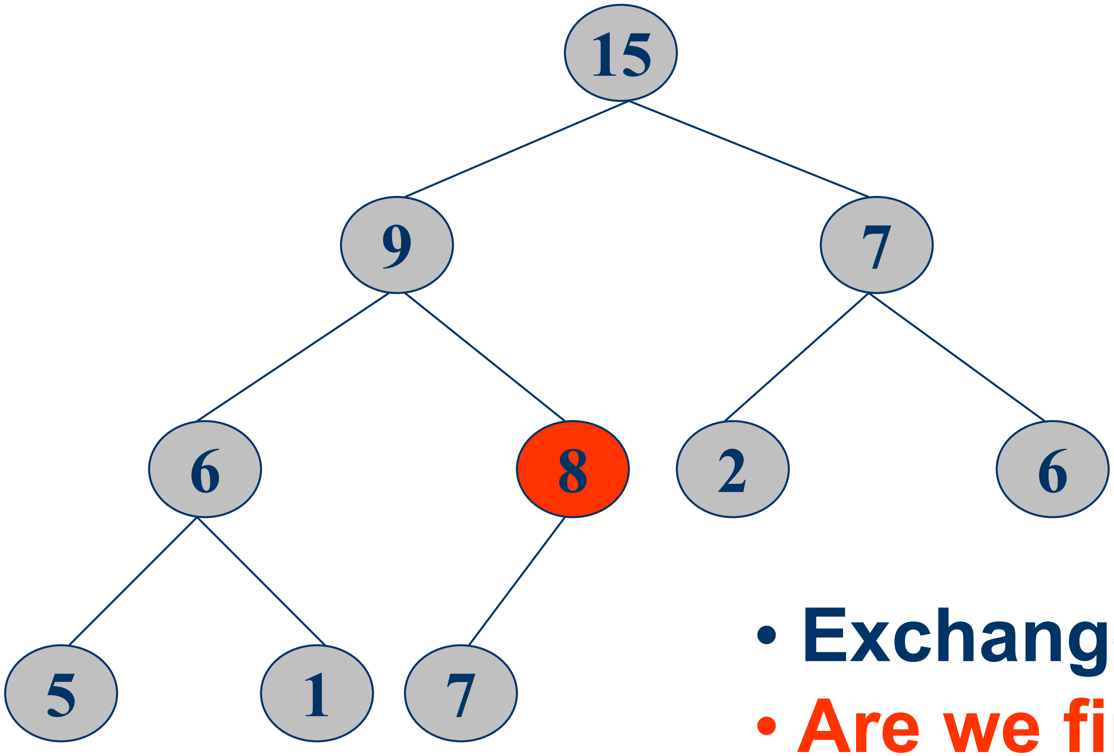
- Reinsert 8 into the heap.
- Are we finished?

Deletion from a Max Heap



- **Exchange the position with 15**
- **Are we finished?**

Deletion from a Max Heap



- **Exchange the position with 9**
- **Are we finished?**

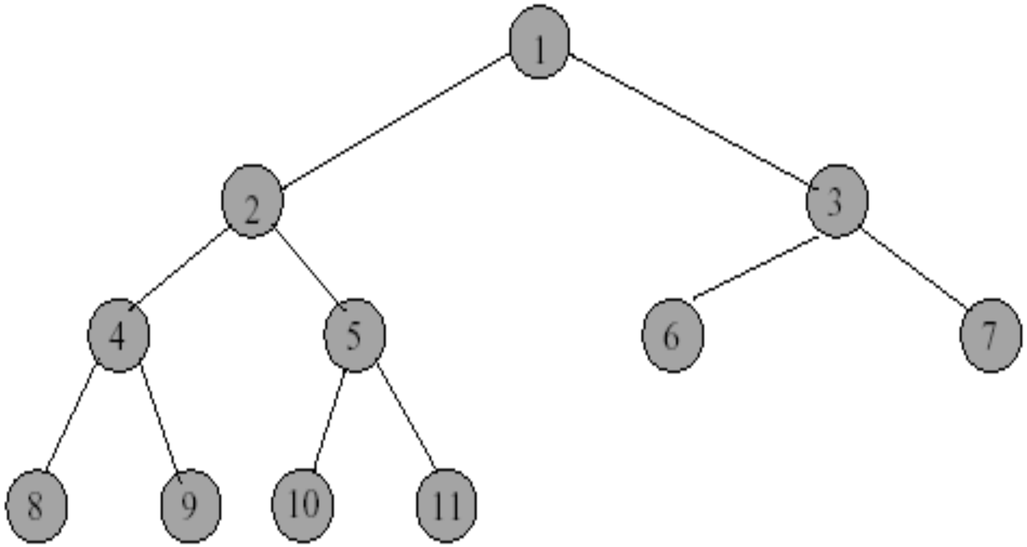
Complexity of Deletion

- See also Figure 12.4 for another deletion example

- The time complexity of deletion is the same as insertion
- At each level, we do $\Theta(1)$ work
- Thus the time complexity is **O(height) = O($log_2 n$)**, where n is the heap size

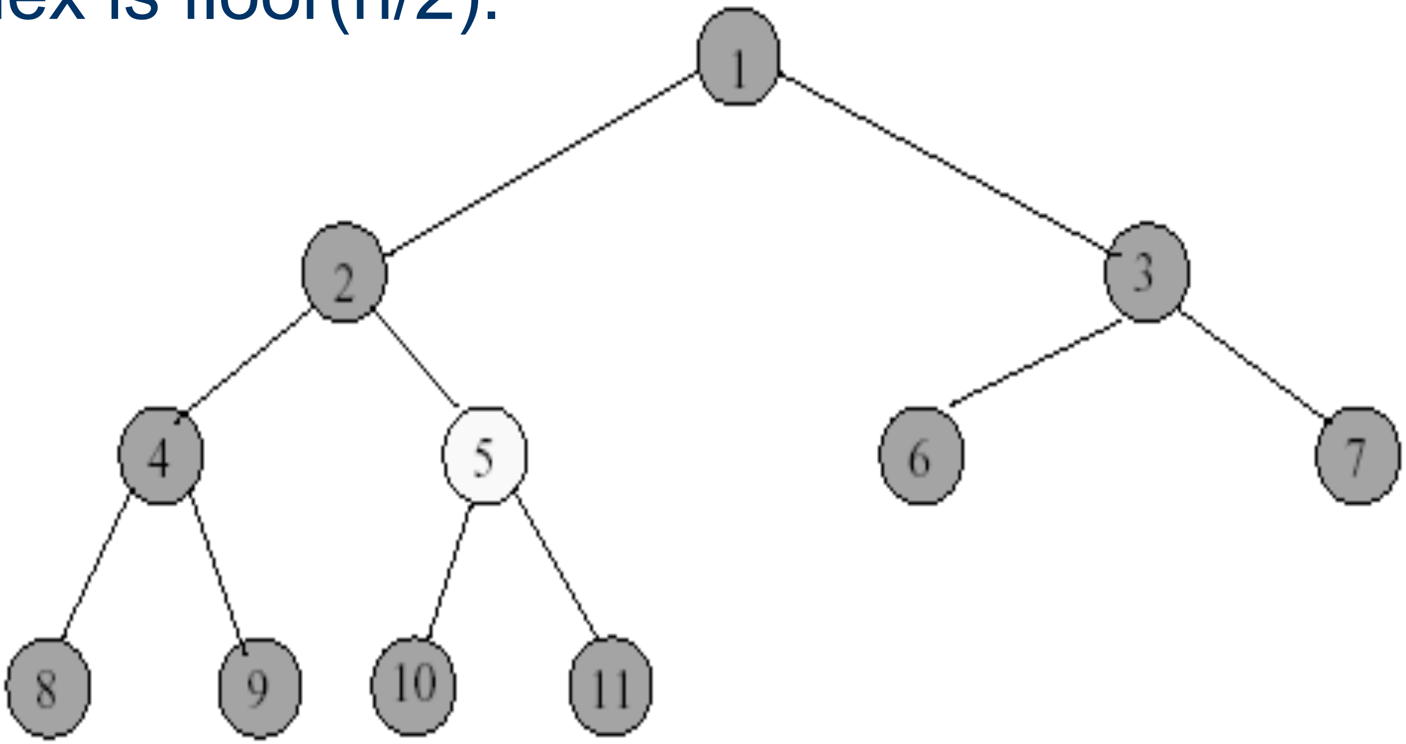
Max Heap Initialization

- Heap initialization means to construct a heap by adjusting the tree if necessary
- Example: input array = [-,1,2,3,4,5,6,7,8,9,10,11]

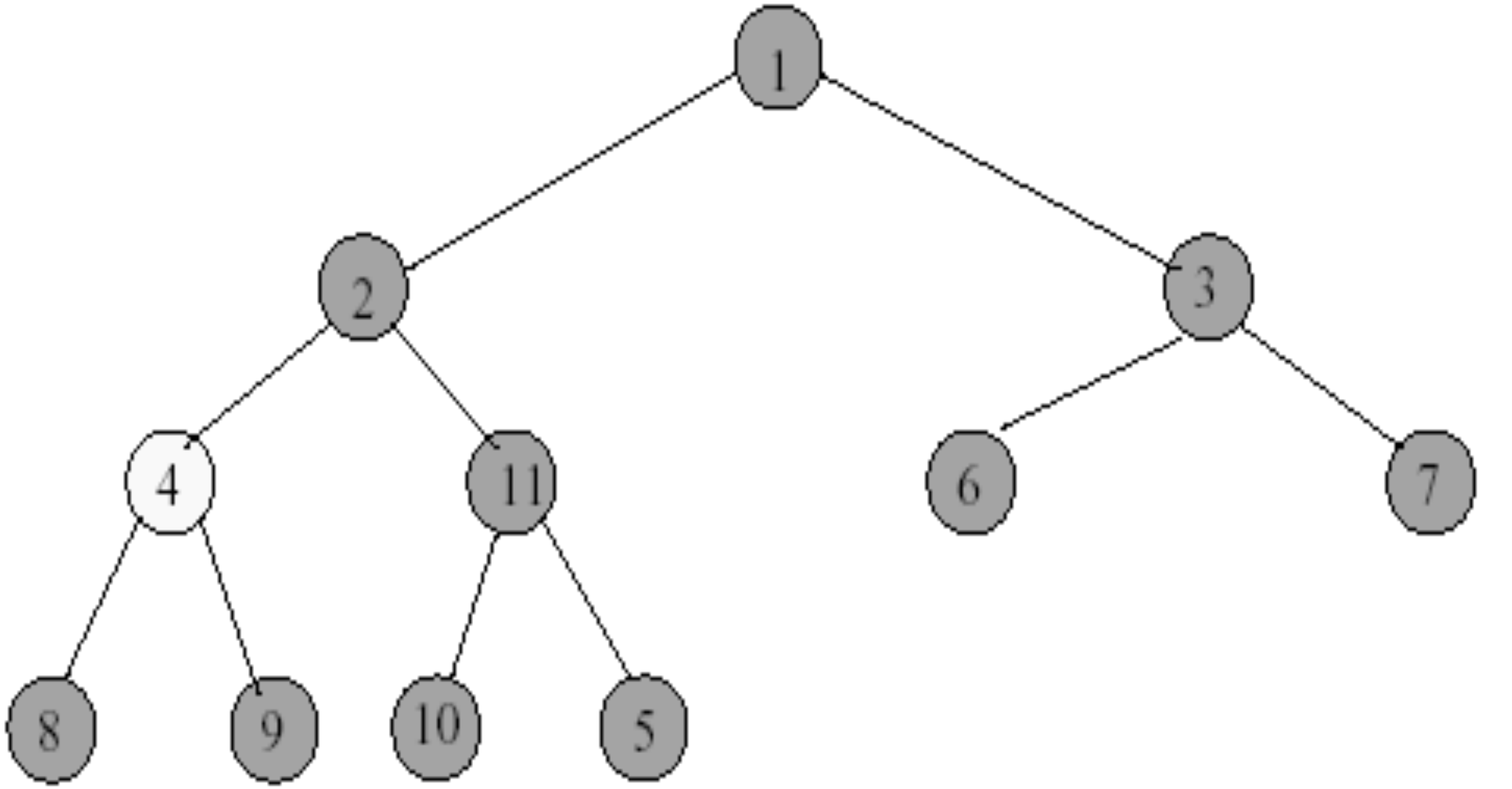


Max Heap Initialization

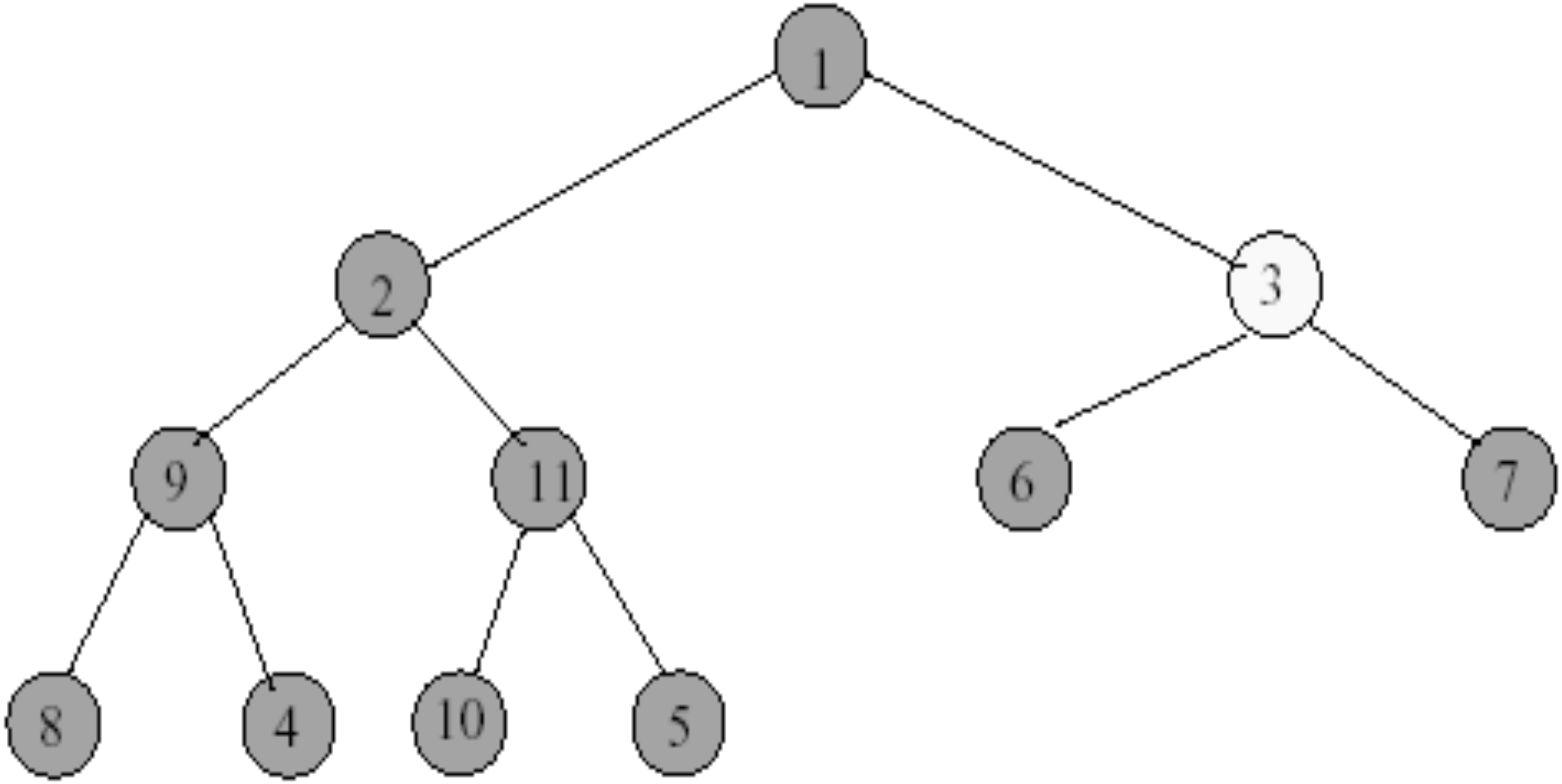
- Start at rightmost array position that has a child.
- Index is floor(n/2).



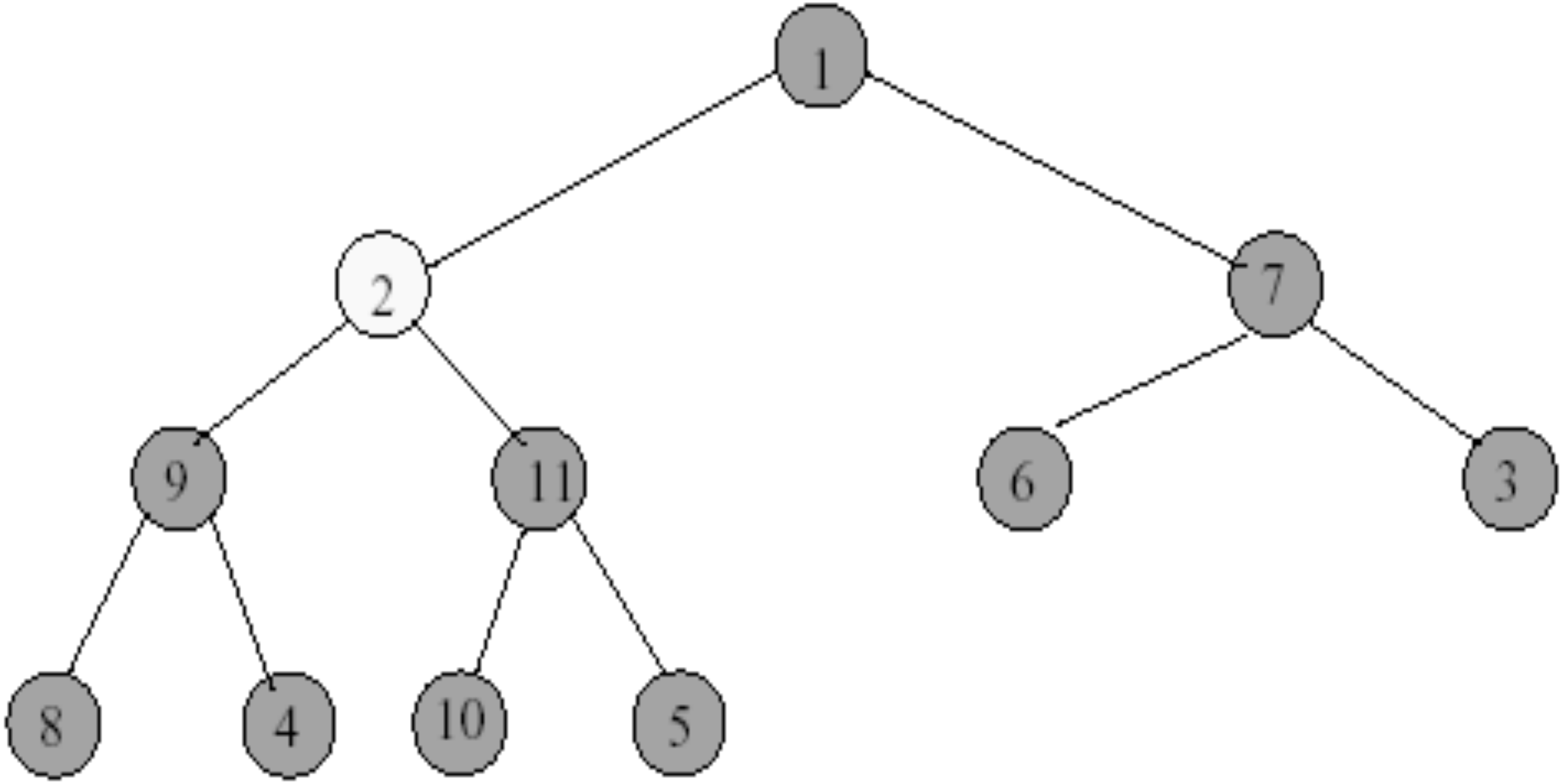
Max Heap Initialization



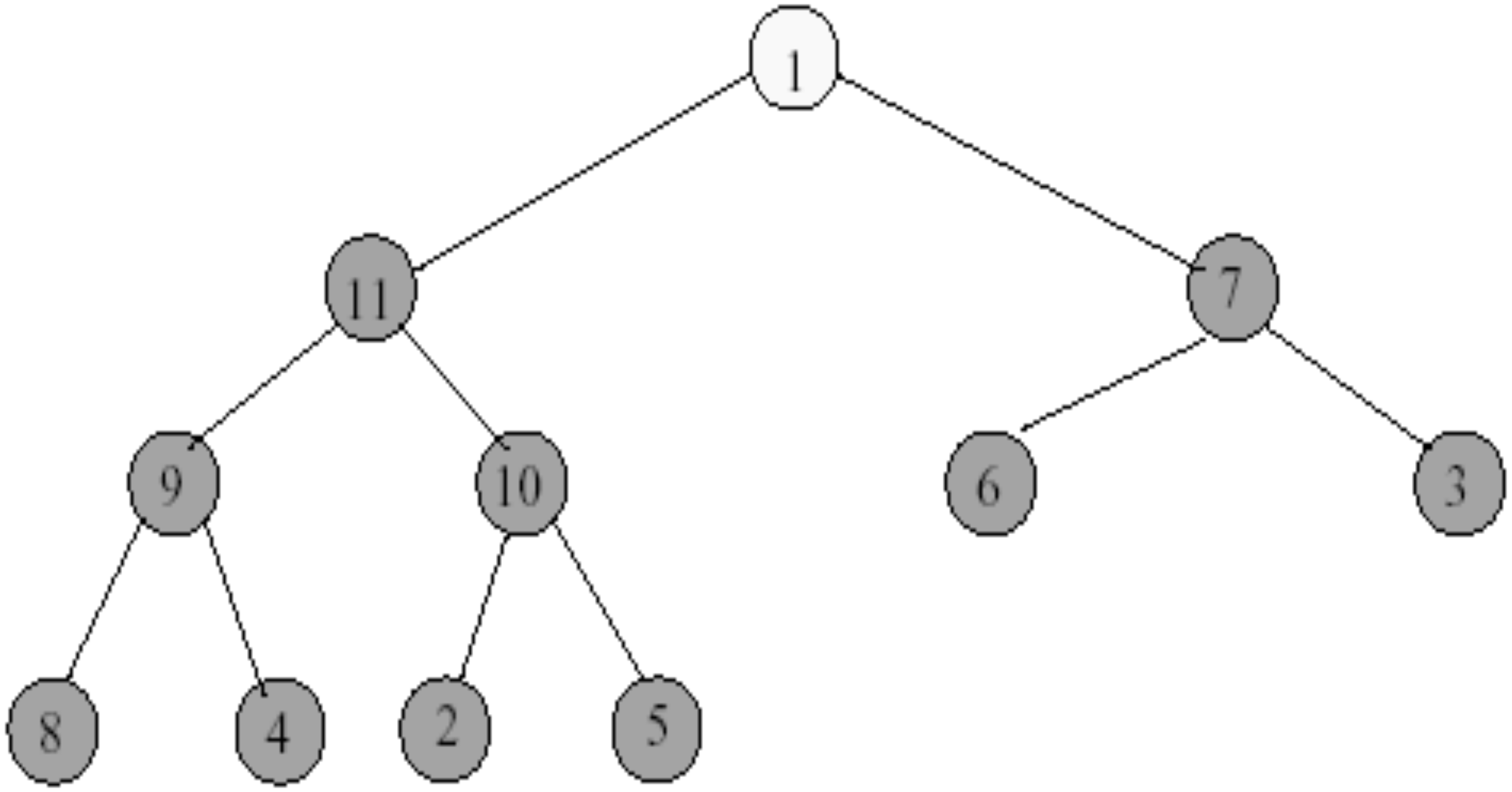
Max Heap Initialization



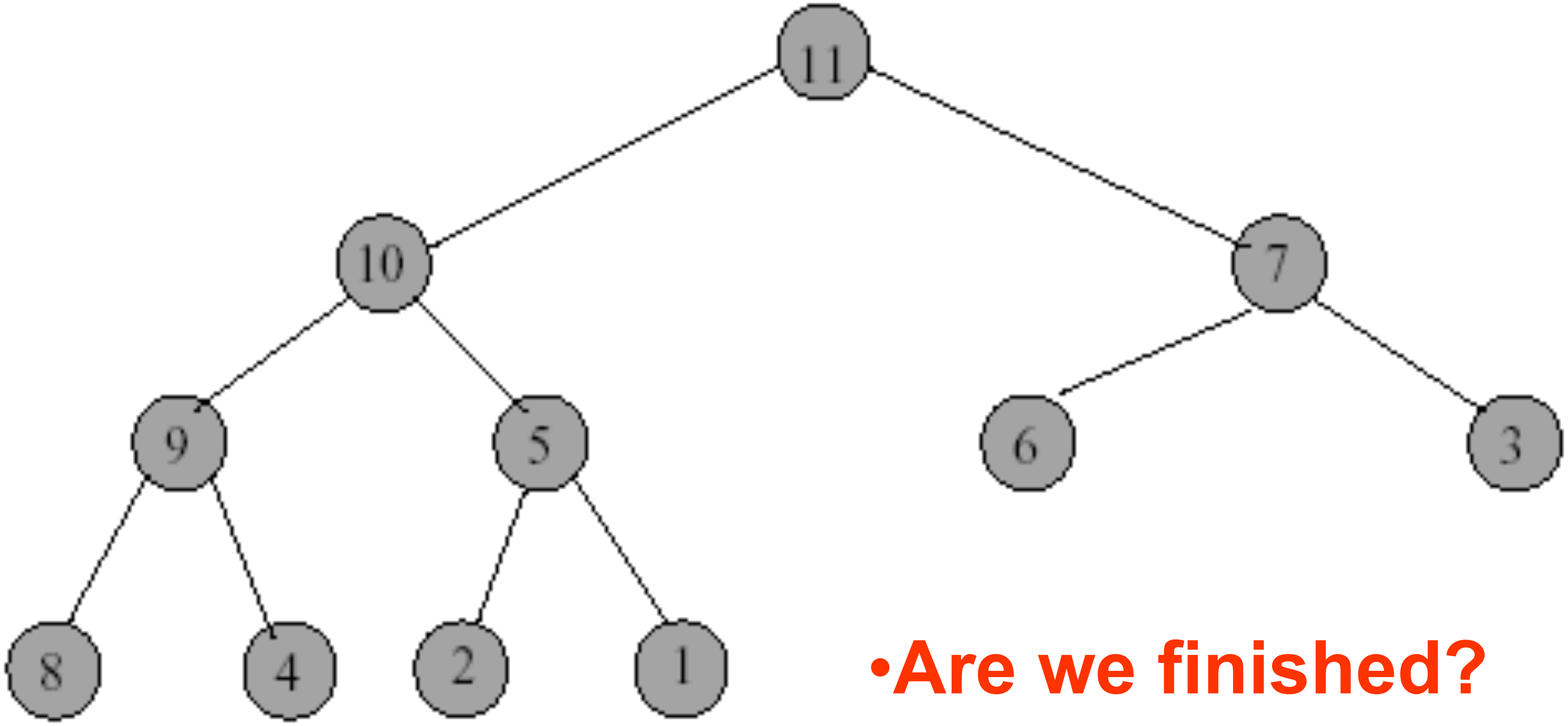
Max Heap Initialization



Max Heap Initialization



Max Heap Initialization



- **Are we finished?**
- **Done!**

Complexity of Initialization

- See Figure 12.5 for another initialization example

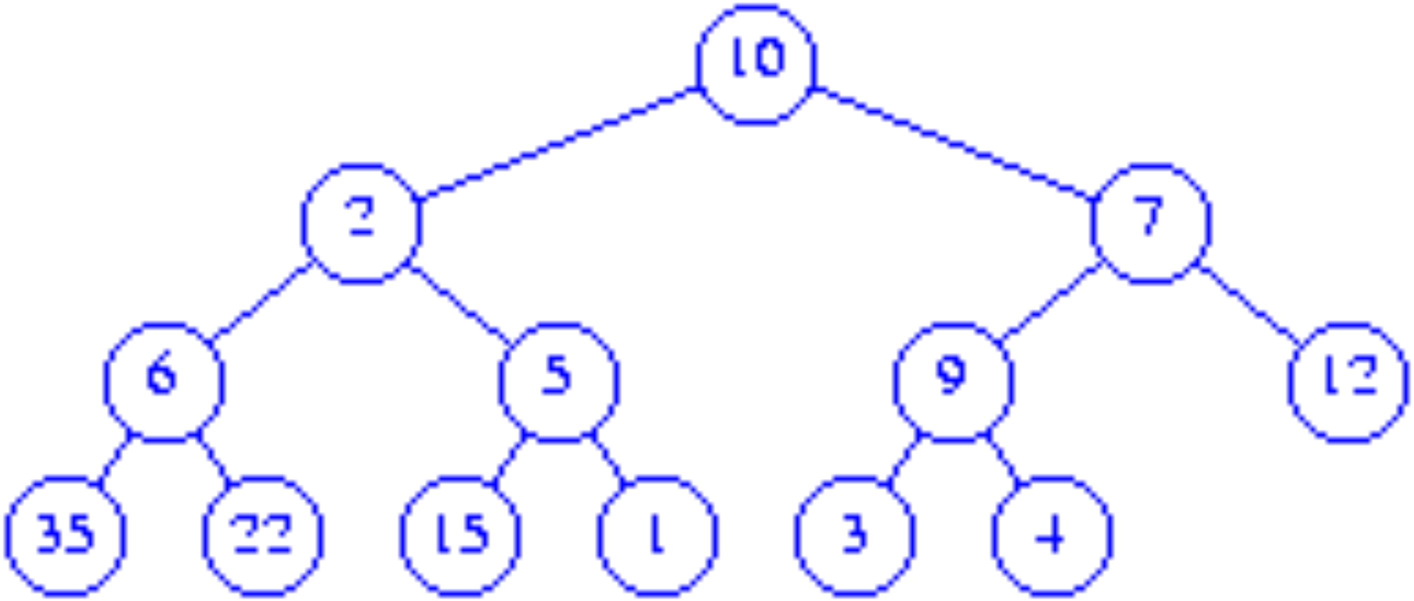
- Height of heap = h.
- Number of nodes at level j is <= 2^{j-1}.
- Time for each node at level j is O(h-j+1).
- Time for all nodes at level j is <= $2^{j-1}(h-j+1) = t(j)$.
- Total time is t(1) + t(2) + … + t(h) = $O(2^h)$ = **O(n)**.

Exercise 12.7

- Do Exercise 12.7
  - theHeap = [-, 10, 2, 7, 6, 5, 9, 12, 35, 22, 15, 1, 3, 4]

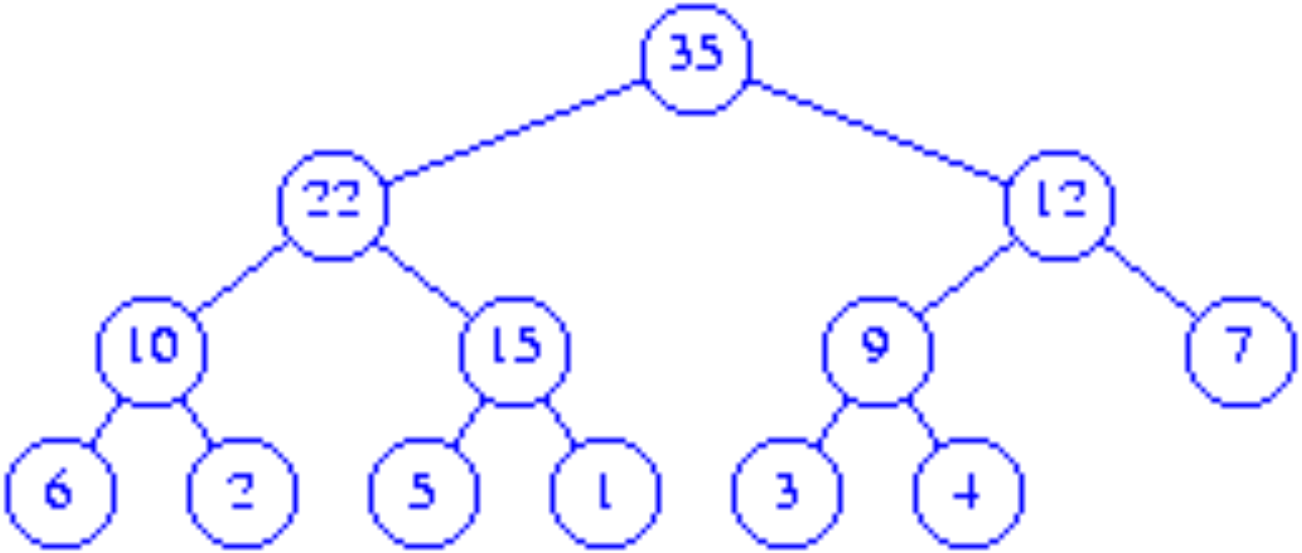
Exercise 12.7 (a)

- 12.7 (a) – complete binary tree



Exercise 12.7 (b)

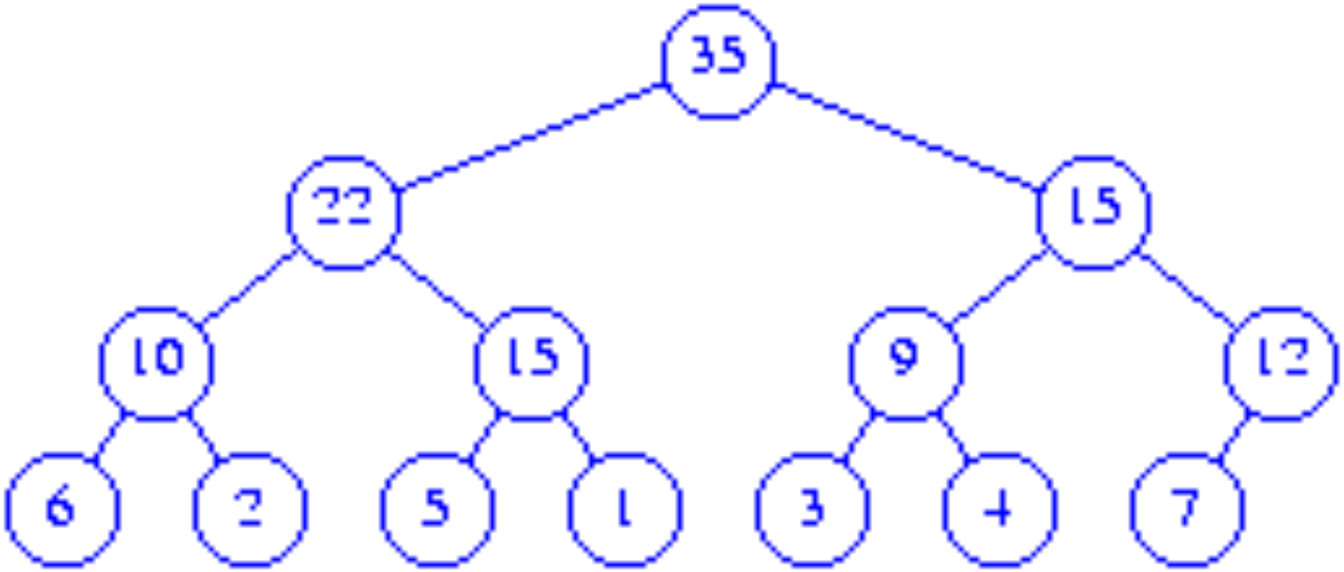
- 12.7 (b) – The heapified tree



[35, 22, 12, 10, 15, 9, 7, 6, 2,5, 1, 3, 4]

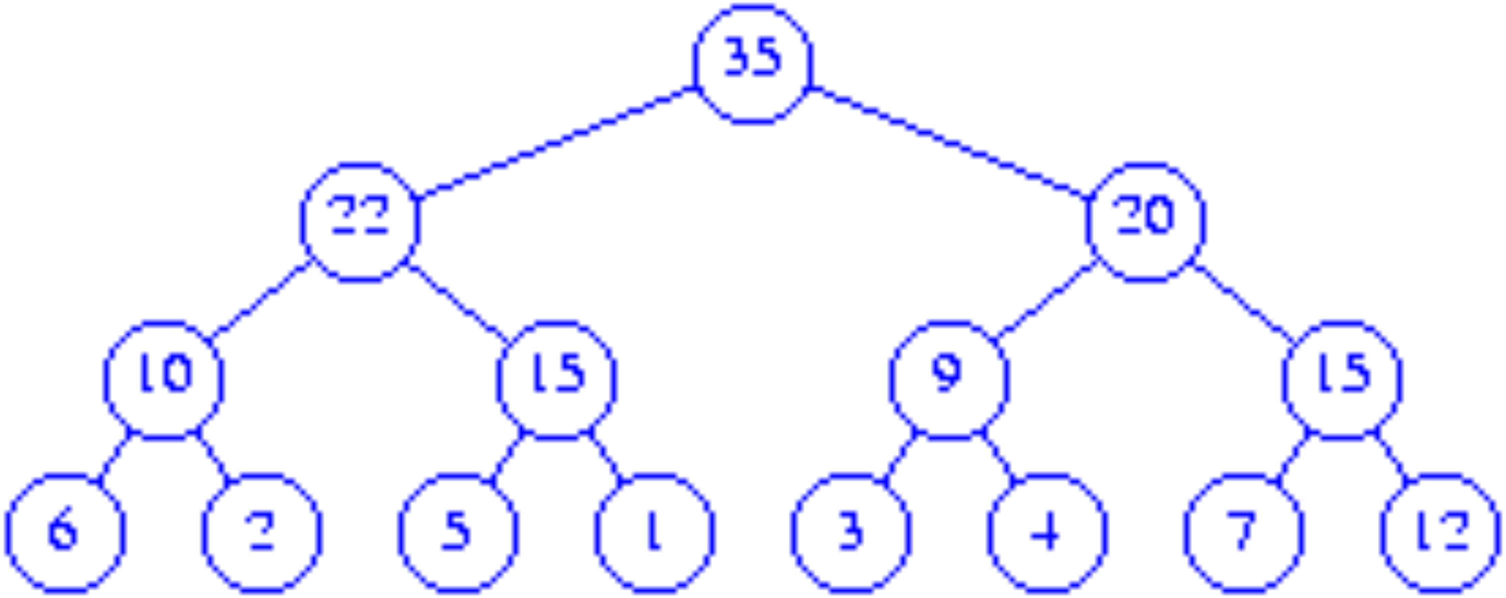
Exercise 12.7 (c)

- 12.7 (c) – The heap after 15 is inserted is:



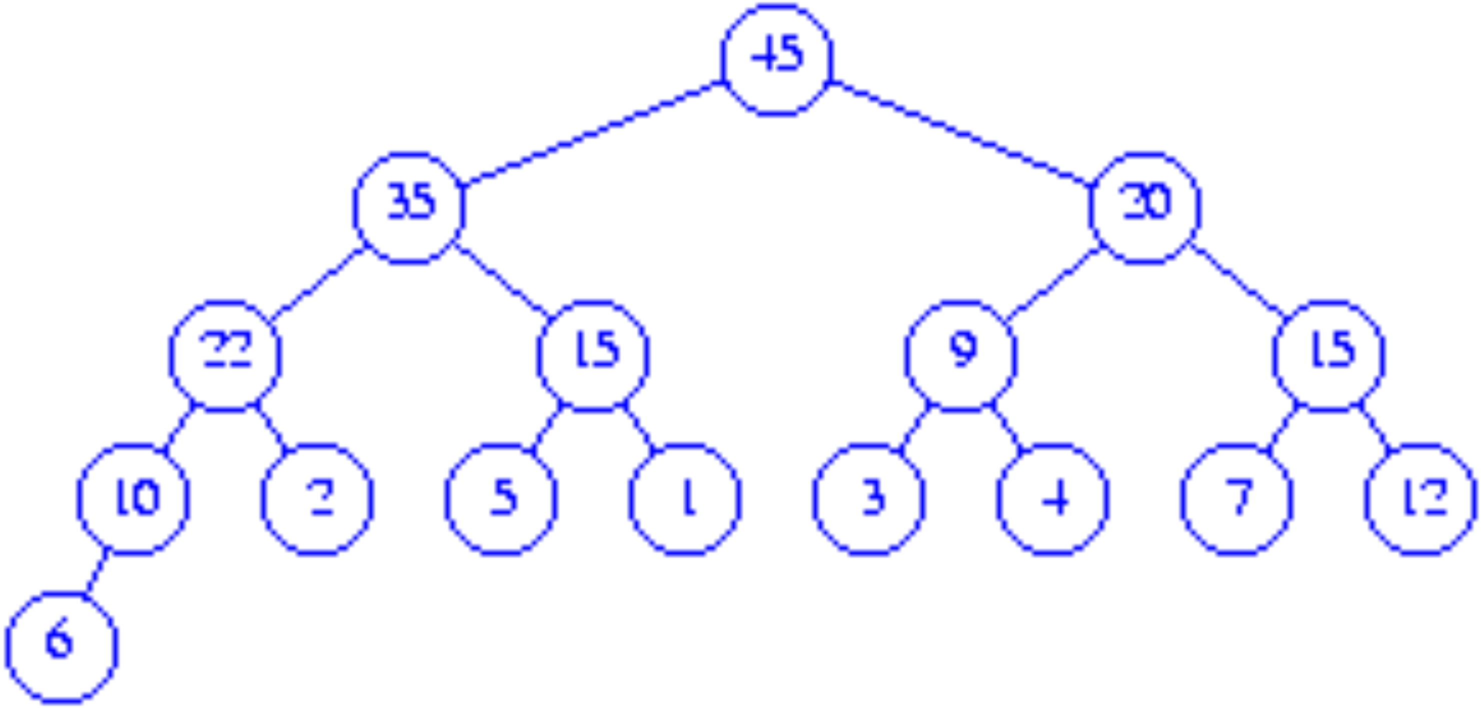
Exercise 12.7 (c)

- 12.7 (c) – The heap after 20 is inserted is:



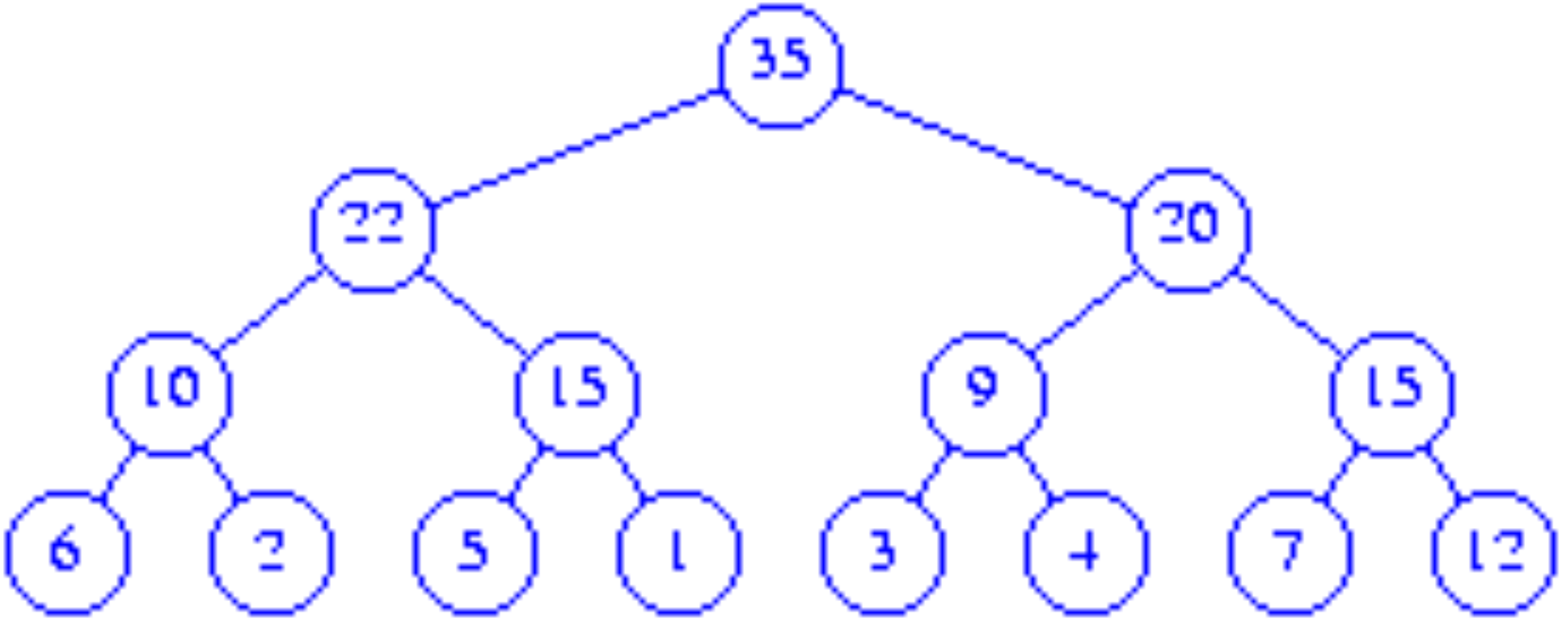
Exercise 12.7 (c)

- 12.7 (c) – The heap after 45 is inserted is:



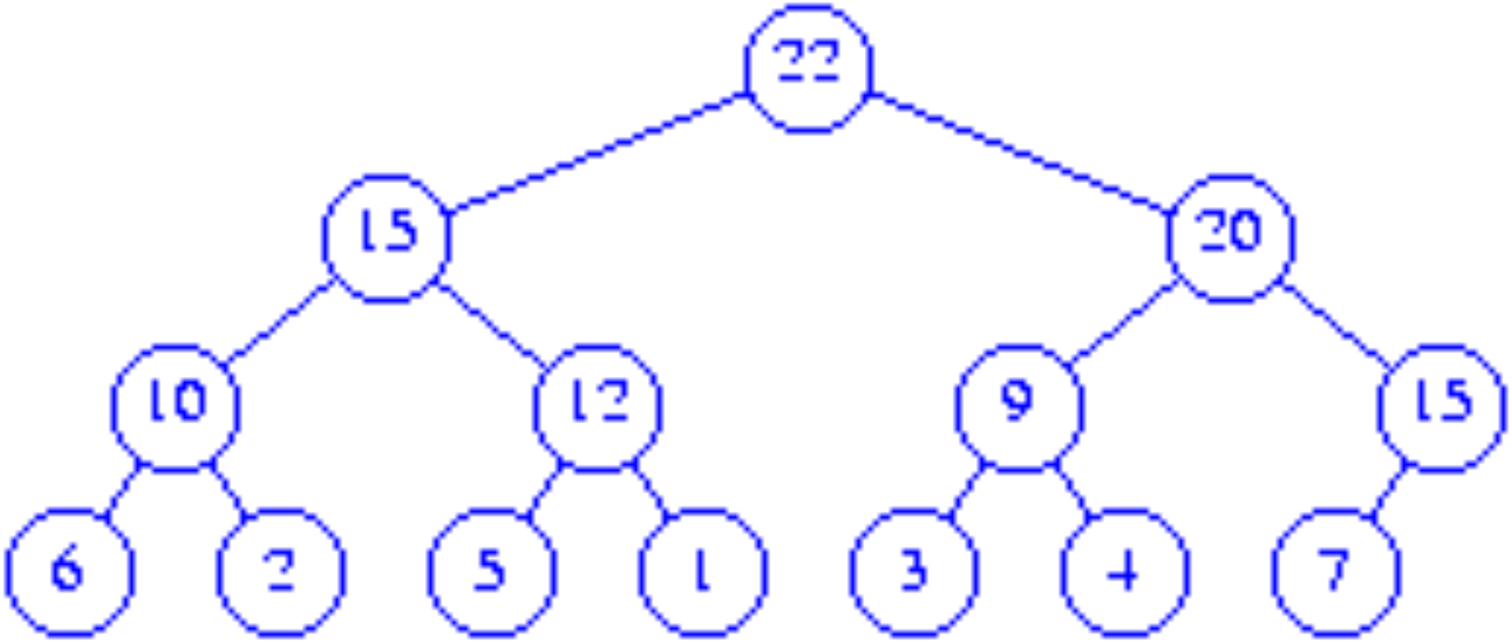
Exercise 12.7 (d)

- 12.7 (d) – **The heap after the first remove max operation is:**



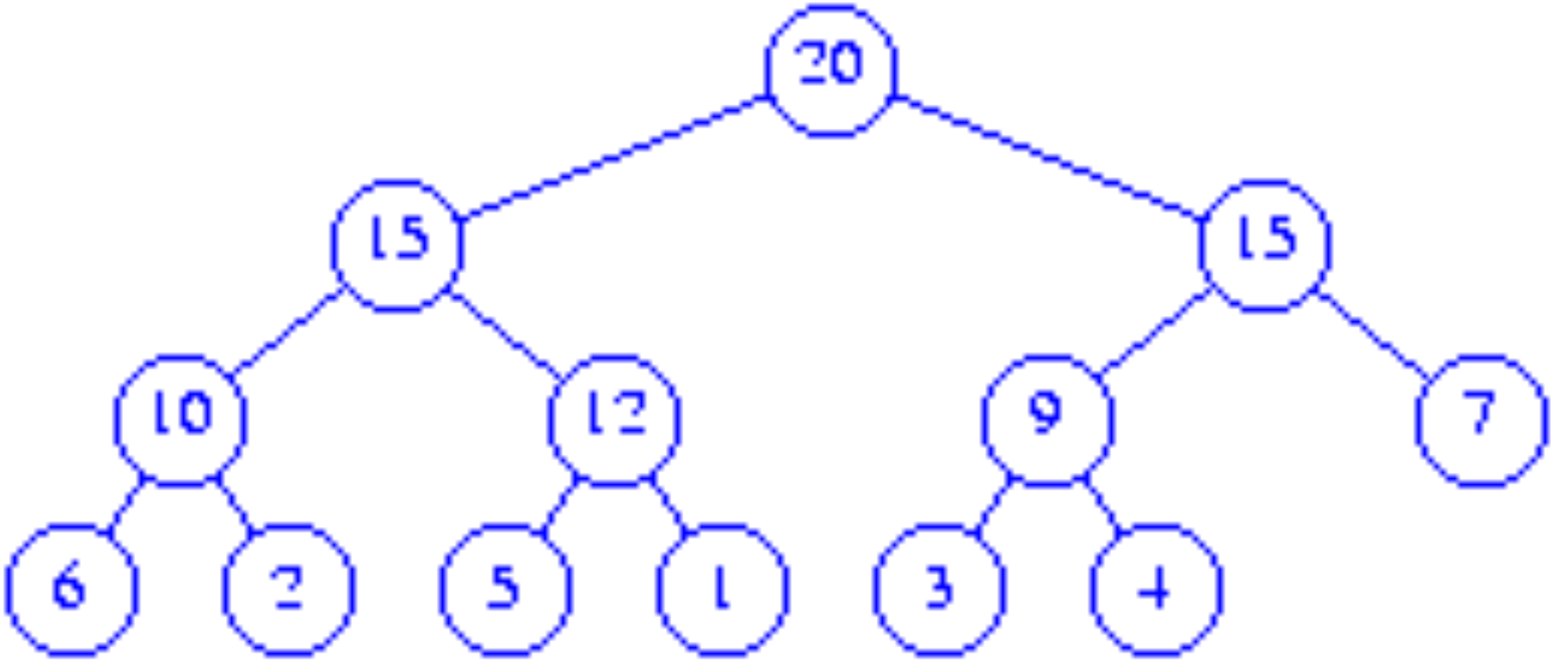
Exercise 12.7 (d)

- 12.7 (d) – **The heap after the second remove max operation is:**



Exercise 12.7 (d)

- 12.7 (d) – **The heap after the third remove max operation is:**



Leftist Trees

- Despite heap structure being both space and time efficient, it is NOT suitable for all applications of priority queues
- Leftist tree structures are useful for applications
  - to **meld** (i.e., combine) pairs of priority queues
  - using multiple queues of varying size
- **Leftist tree** is a linked data structure suitable for the implementation of a priority queue
- A tree which tends to **“lean” to the left**.

Applications of Heaps

- Sort (heap sort)
- Machine scheduling
- Huffman codes

Heap Sort

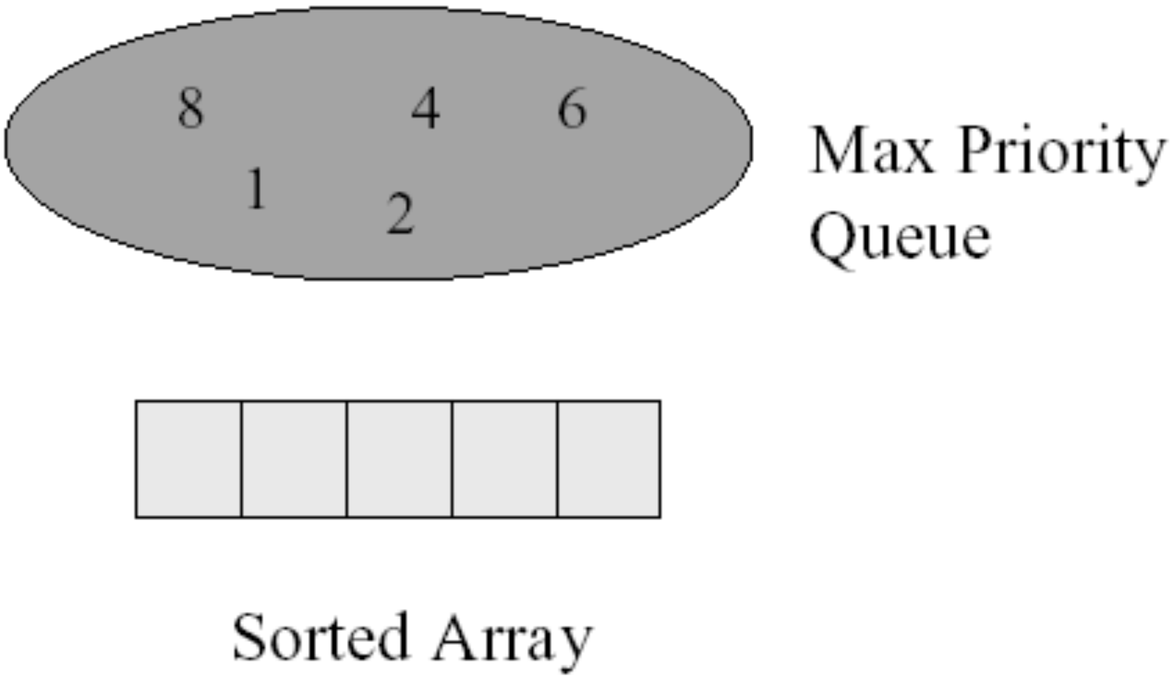
- use element key as priority

Algorithm

- put elements to be sorted into a priority queue (i.e., initialize a heap)
- extract (delete) elements from the priority queue
  - if a min priority queue is used, elements are extracted in increasing order of priority
  - if a max priority queue is used, elements are extracted in decreasing order of priority

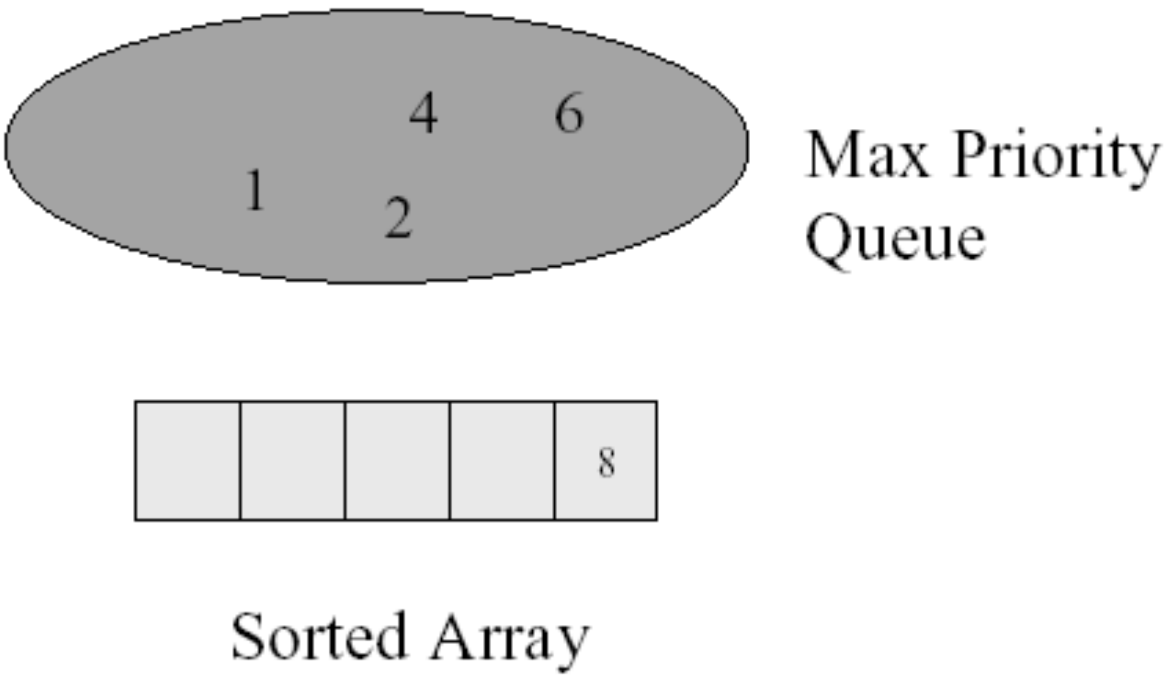
Heap Sort Example

- After putting into a max priority queue



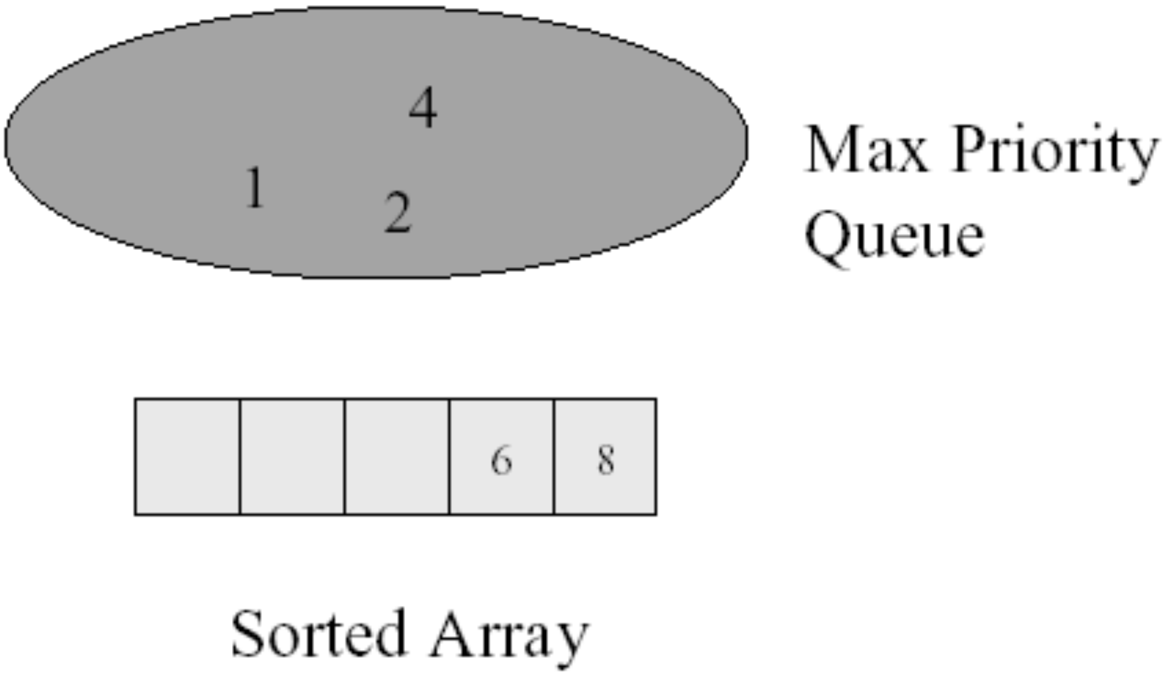
Sorting Example

- After first remove max operation



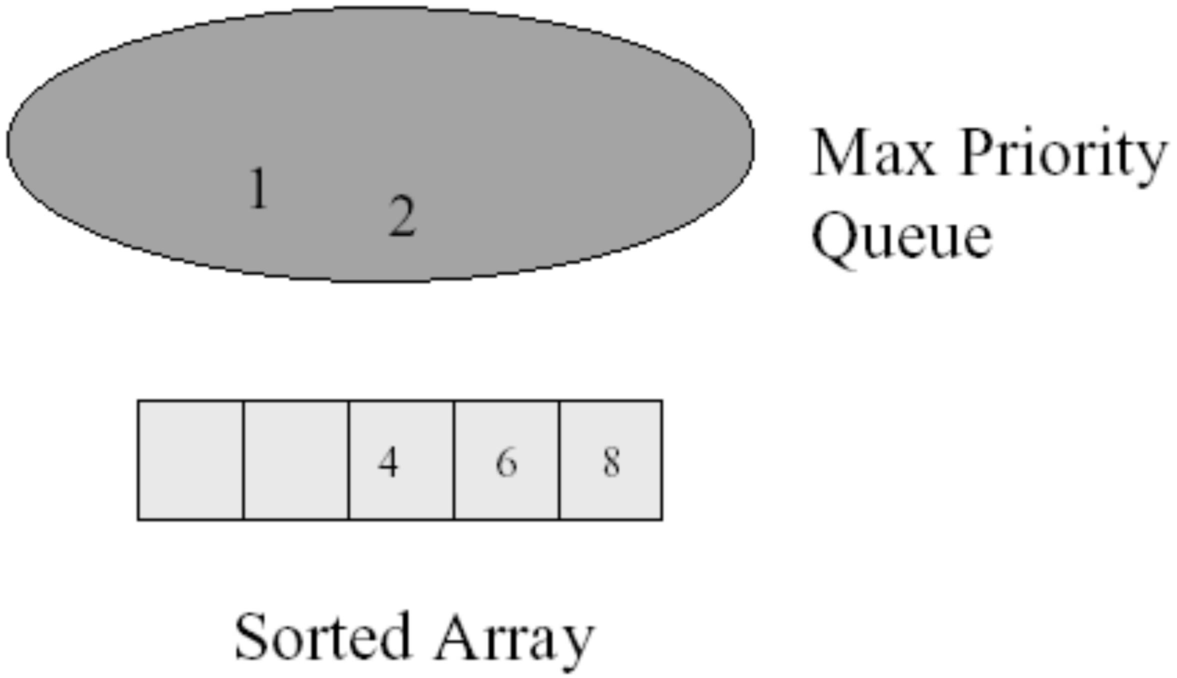
Sorting Example

- After second remove max operation



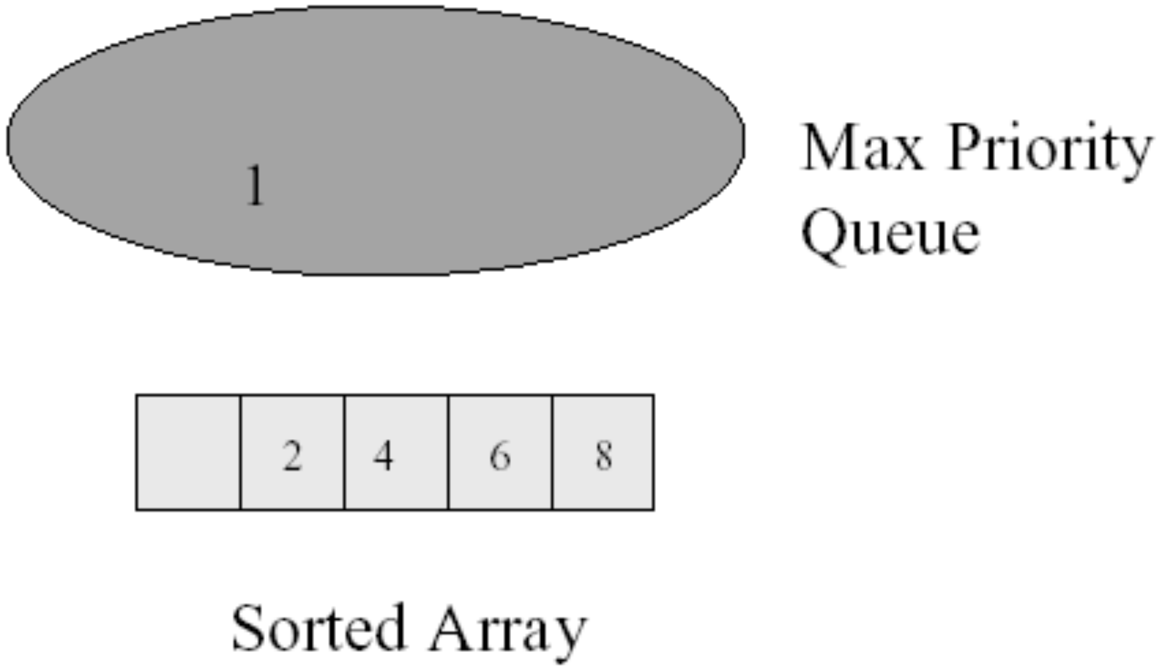
Sorting Example

- After third remove max operation



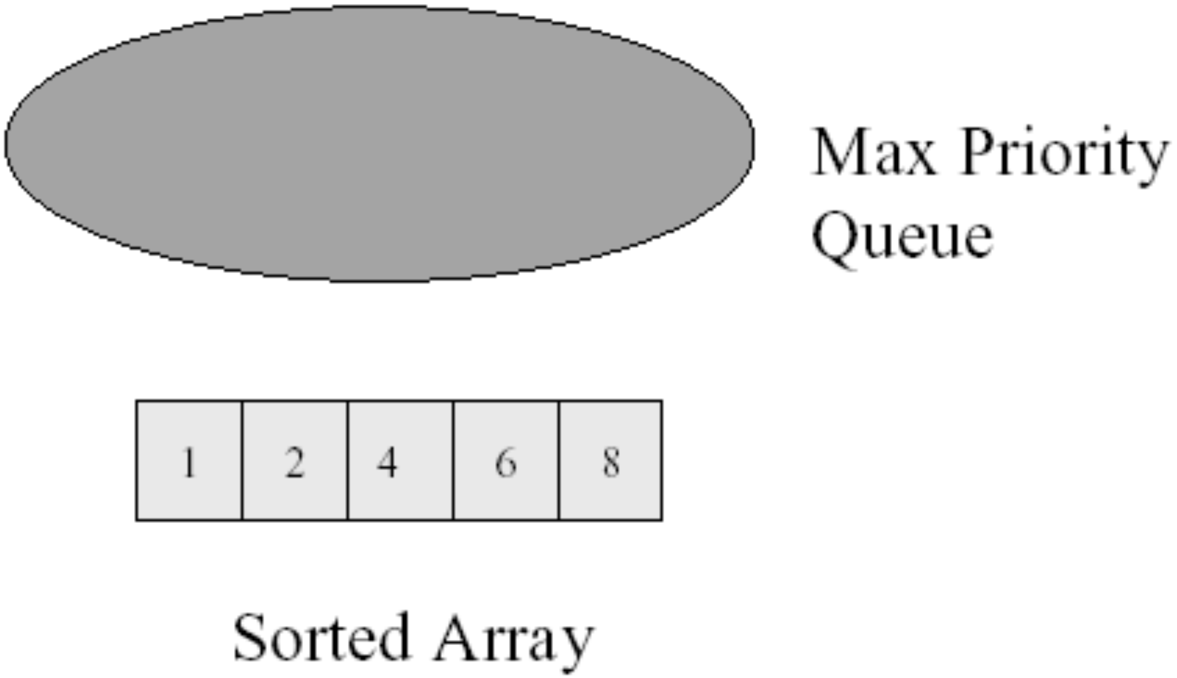
Sorting Example

- After fourth remove max operation



Sorting Example

- After fifth remove max operation



Complexity Analysis of Heap Sort

- See Program 12.8 for Heap Sort
- See Figure 12.9 for another Heap Sort example

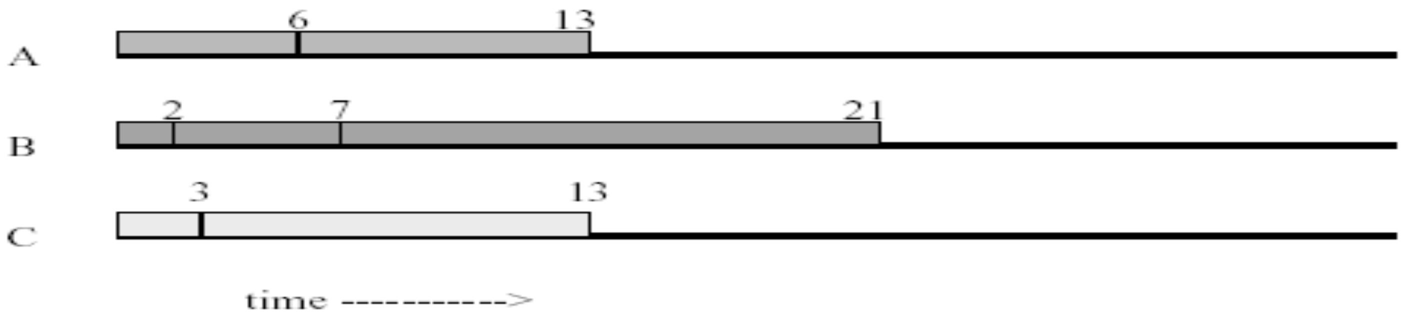
- Heap sort *n* elements.
  - Initialization operation takes O(*n*) time
  - Each deletion operation takes O(*log n*) time
  - Thus, the total time is **O(*n log n*)** **- Why?**
    - **→ The heap has to be reinitialized (melded) after each delete operation**
  - compare with $O(n^2)$ for sort methods of Chapter 2

Machine Scheduling Problem

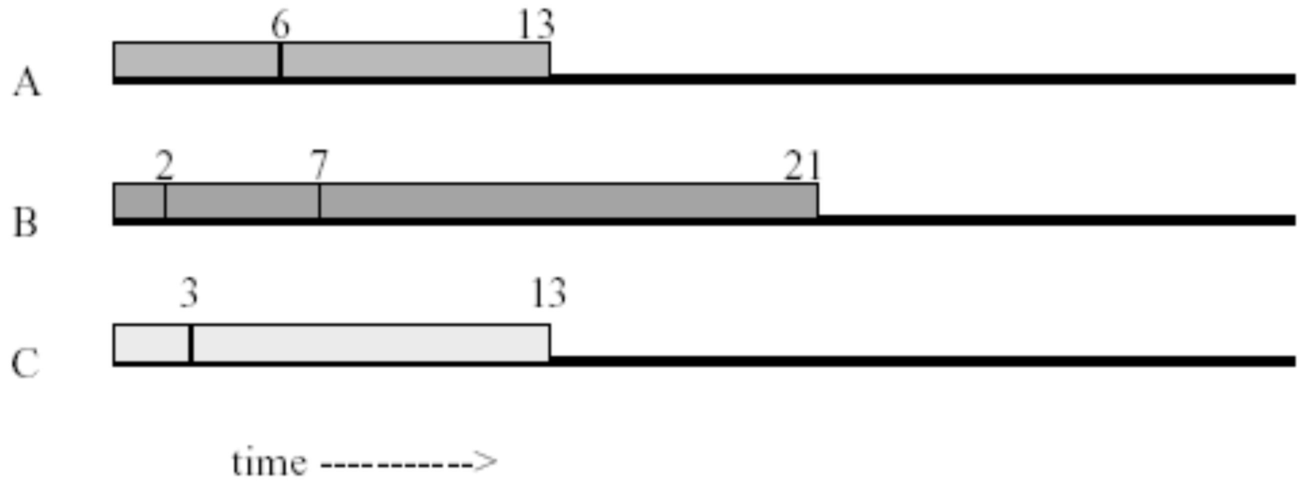
- *m* identical machines
- *n* jobs to be performed
- The machine scheduling problem is to assign jobs to machines so that the time at which the last job completes is minimum

Machine Scheduling Example

- 3 machines and 7 jobs
- job times are [6,2,3,5,10,7,14]
- What are some possible schedules?
- A possible schedule:



Machine Scheduling Example



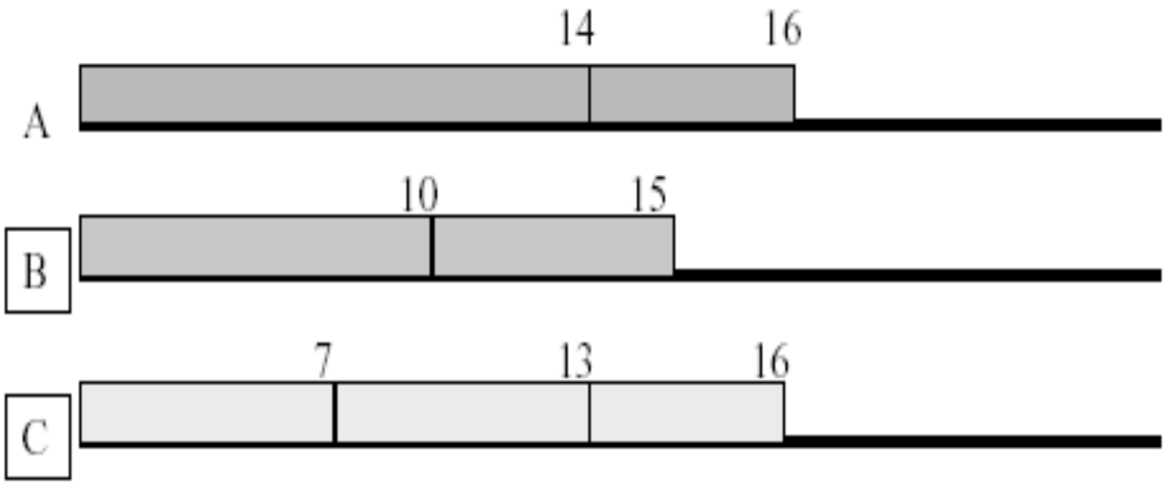
- What is the finish time (length) of the schedule?
  - ➔ 21
- Objective: Find schedules with minimum finish time
- Minimum finish time scheduling is **NP-hard**.

NP-hard Problems

- The class of problems for which no one has developed a polynomial time algorithm.
- No algorithm whose complexity is **$O(n^k m^l)$** is known for any NP-hard problem (for any constants *k* and *l*)
- NP stands for **Nondeterministic Polynomial**
- NP-hard problems are often solved by **heuristics** (or **approximation algorithms**), which do not guarantee optimal solutions
- Longest Processing Time (LPT) rule is a good heuristic for minimum finish time scheduling.

LPT Schedule & Example

- Longest Processing Time (LPT) first
- Jobs are scheduled in the descending order 14, 10, 7, 6, 5, 3, 2
- Each job is scheduled on the machine on which it finishes earliest



finish time is 16!

LPT Schedule & Example

- What is the minimum finish time with thee machines for jobs (2, 14, 4, 16, 6, 5, 3)?
- See Figure 12.10

LPT using a Min Heap

- Min Heap has the finish times of the *m* machines.
- Initial finish times are all 0.
- To schedule a job, remove the machine with minimum finish time from the heap.
- Update the finish time of the selected machine and put the machine back into the min heap.
- See Program 12.9 for LPT scheduler

Huffman Codes

- For text compression, the LZW method relies on the recurrence of substrings in a text
- **Huffman codes** is another text compression method, which relies on the **relative frequency** (i.e., the number of occurrences of a symbol) with which different symbols appear in a text
- Uses extended binary trees
- Variable-length codes that satisfy the property, where no code is a prefix of another
- **Huffman tree** is a binary tree with minimum weighted external path length for a given set of frequencies (weights)

Huffman Codes

- READ Section 12.6.3