Winter 2016
COMP-250: Introduction to Computer Science

Lecture 20, March 24, 2016

Public Announcement

GETTING YOUR DREAM TECH INTERNSHIP

MARCH 29TH 6PM-7:30PM

TROTTIER 0070

COME LEARN HOW TO APPLY TO COMPANIES, PREPARE FOR INTERVIEWS, AND WHAT COURSES TO TAKE! PRESENTED BY SUCCESSFUL INTERNS:

Lucille Hua
CS U3
Airbnb, Google, Sony Ericsson

Michael Ho
Soft. Eng. U3
Facebook, Apple, Microsoft, Yahoo

Kevin Luk
CS/Bio U3
Knewton, SAP, CIHR

Pizza and swag provided by

Google Canada

CSUS

Public Announcement

Course : COMP 250 Sect: 1	Title: Intro to Computer Science
Exam Date: 4/28/2016	**Exam Time:** 2:00:00 PM

Public Announcement

Mercury Course Evaluations

MERCURY
COURSE EVALUATION

Course evaluations matter. Evaluate your courses and instructors!

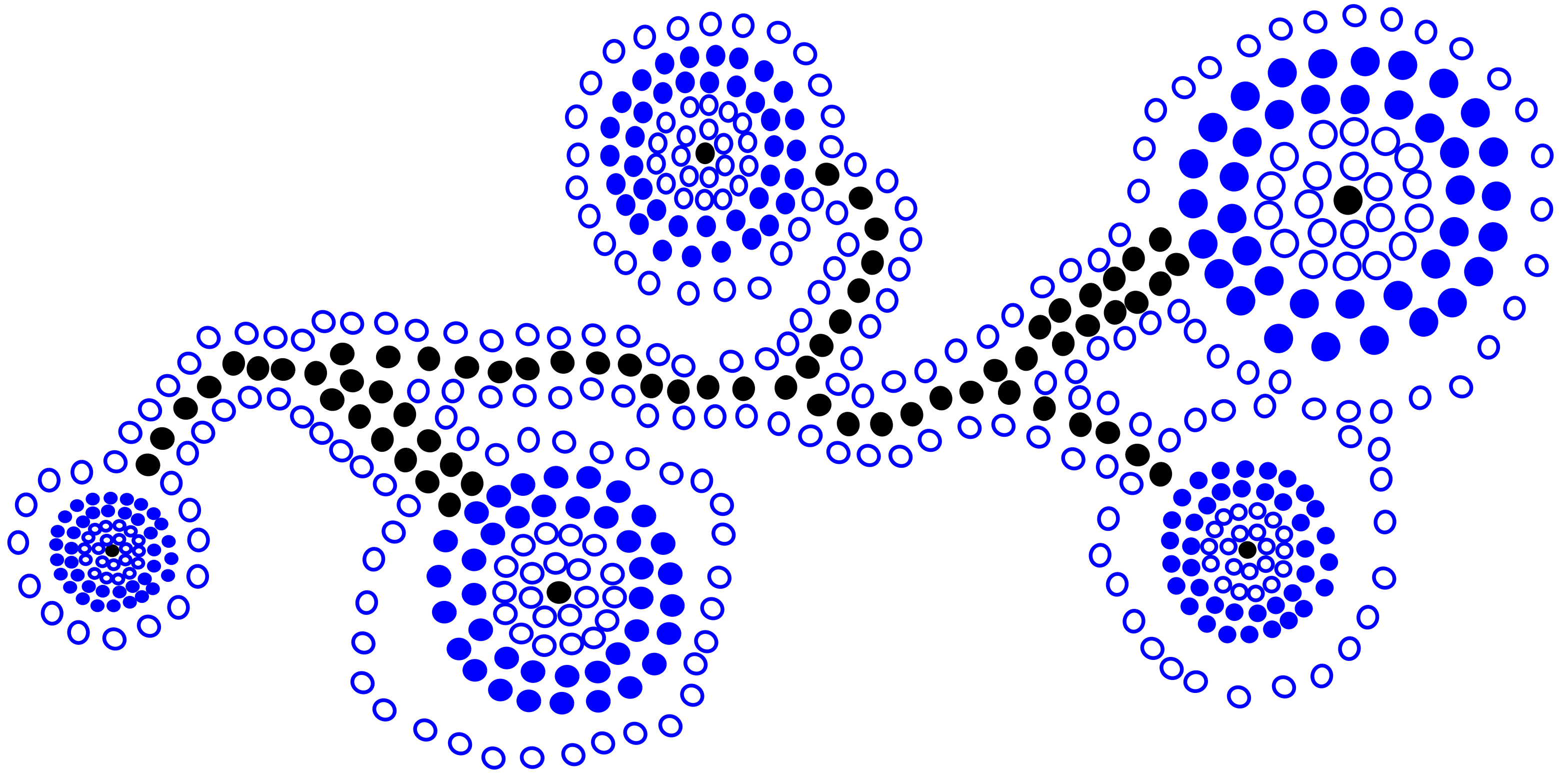
Default period:
March 21 - May 1

Condensed period:
March 21 - April 17

Click **HERE** to complete your course evaluations.

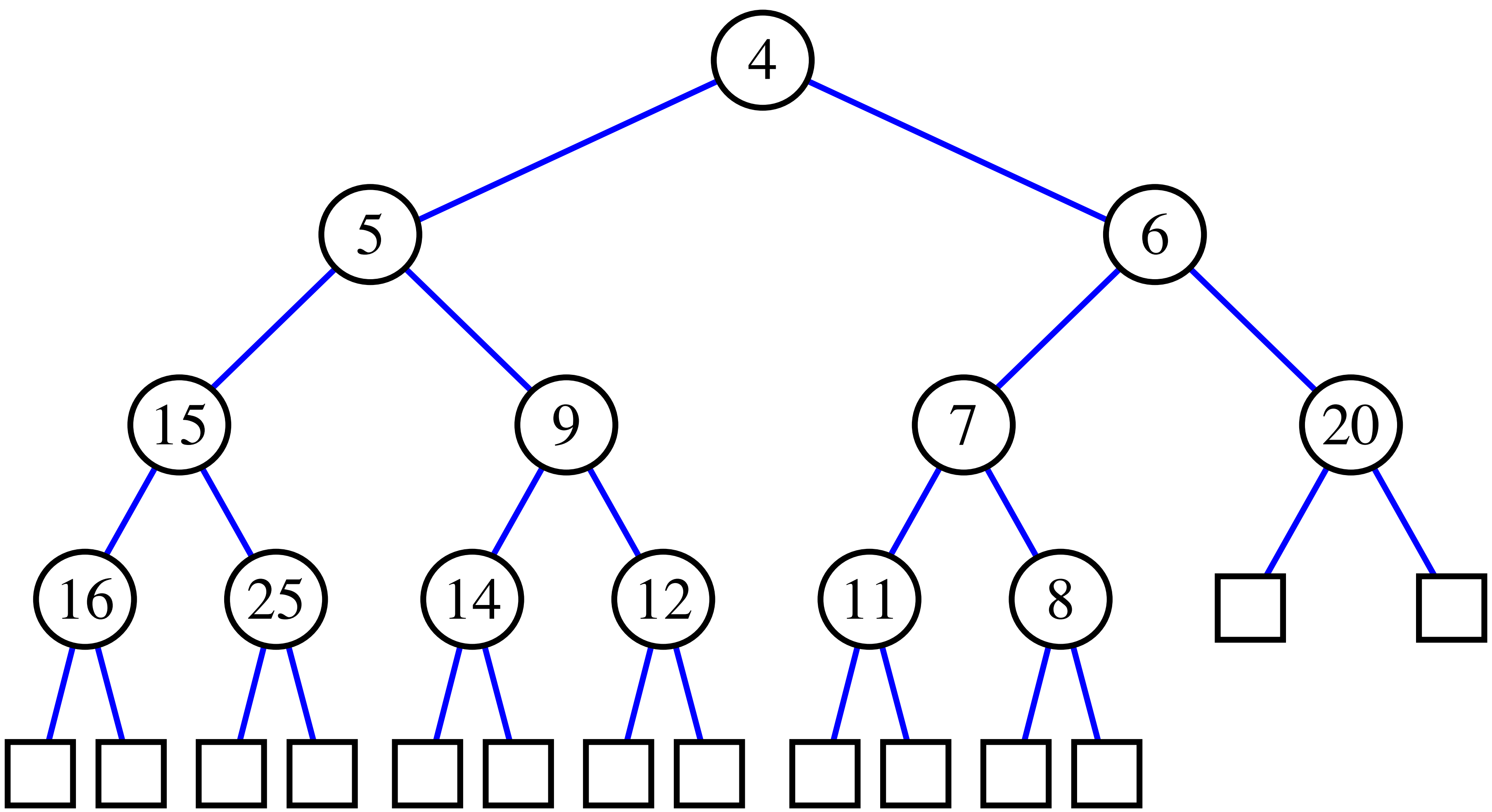
HEAPS I

- Heaps
- Properties
- Insertion and Deletion



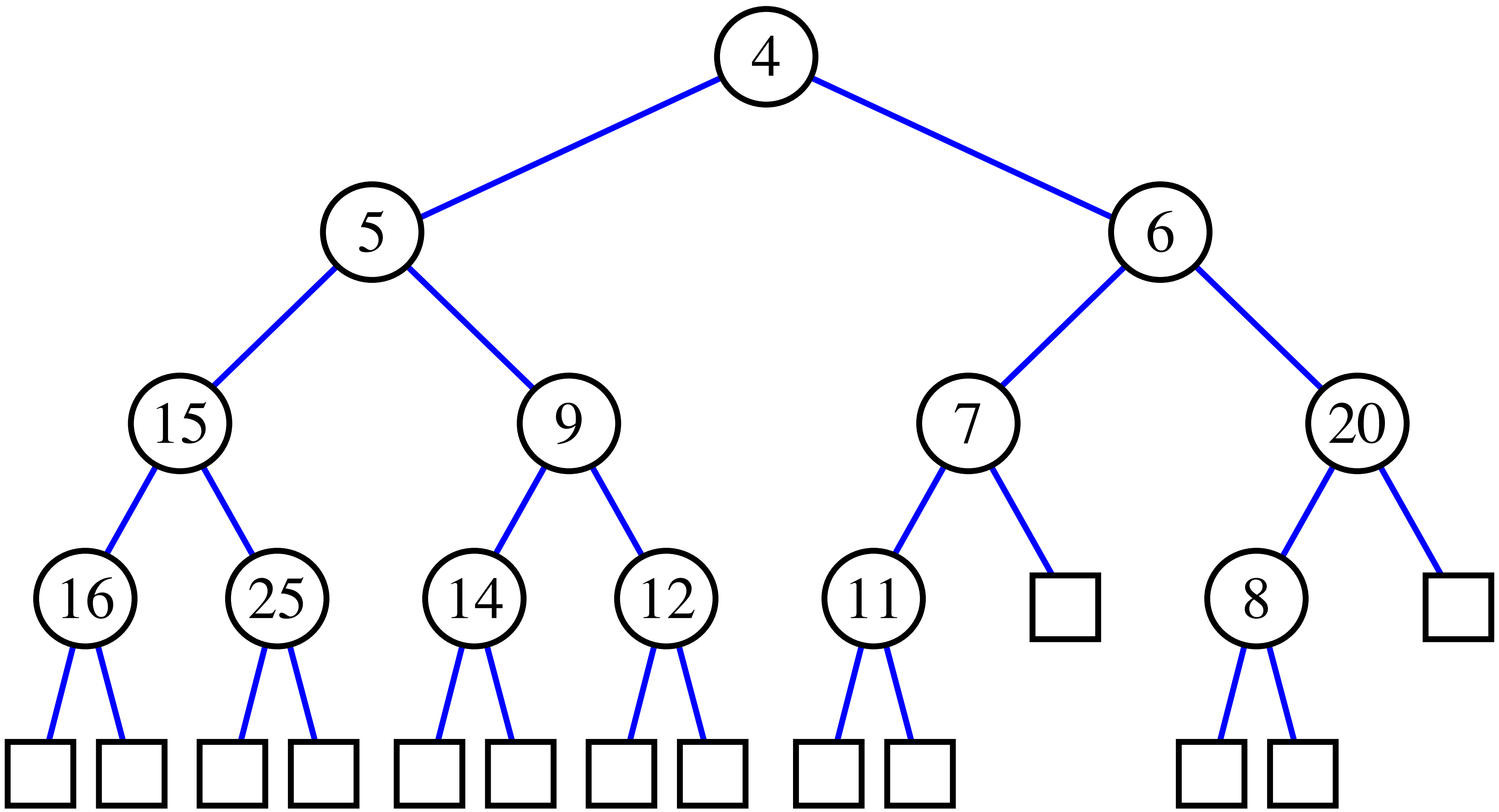
Heaps

- A ***heap*** is a binary tree *T* that stores a collection of keys (or key-element pairs) at its internal nodes and that satisfies two additional properties:
  - ***Order Property:*** key(parent) ≤ key(child)
  - ***Structural Property***: all levels are full, except the last one, which is left-filled (***complete binary tree***)



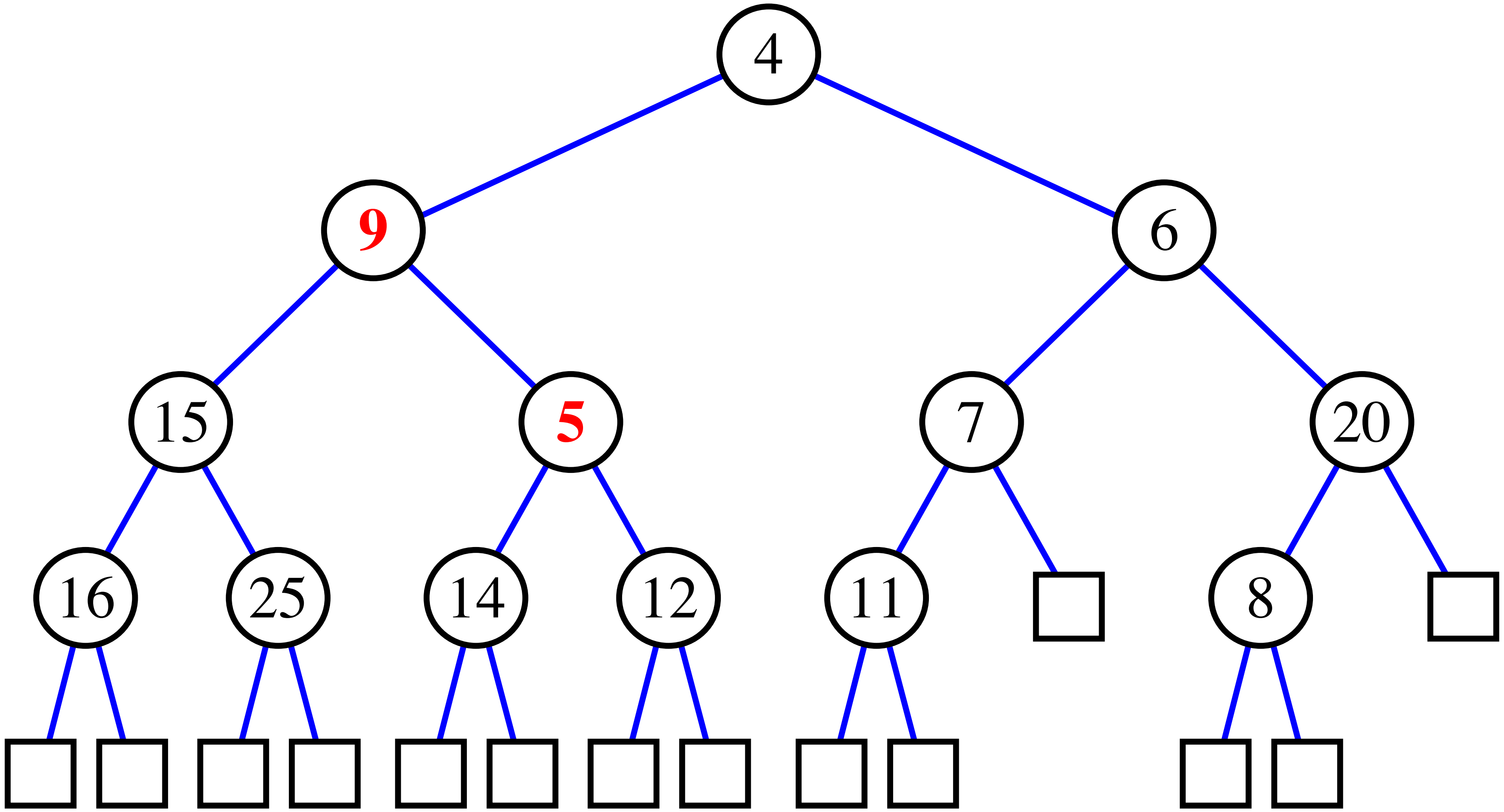
Not Heaps

- bottom level is not left-filled



Not Heaps

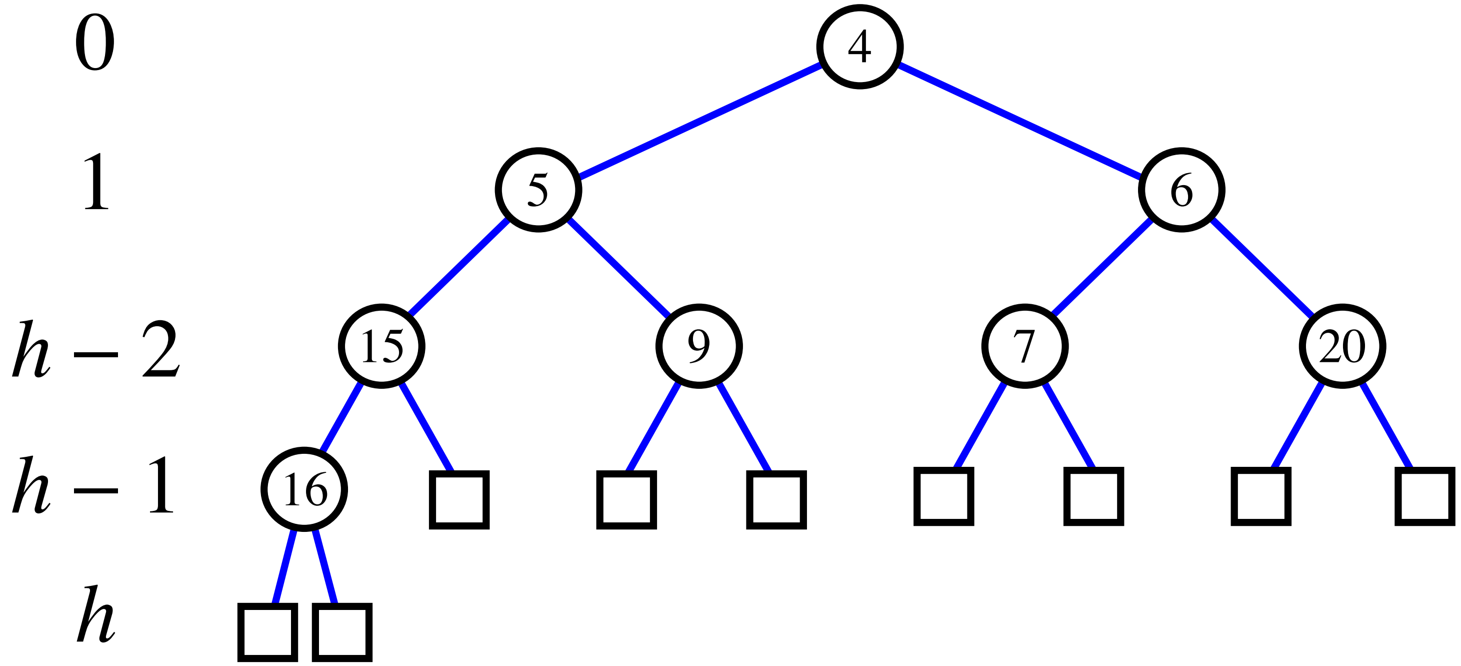
- key(parent)> key(child)



Height of a Heap

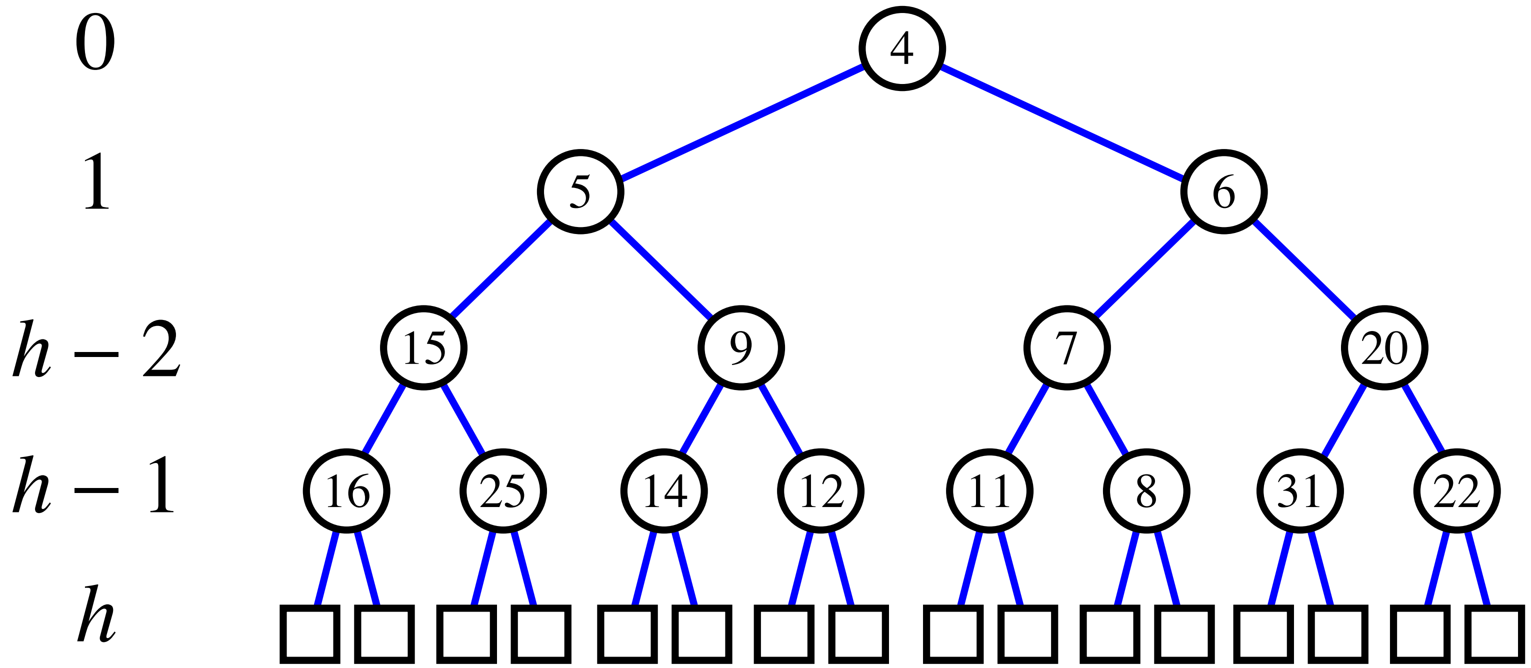
A heap T storing n keys has height $h = \lceil \log(n + 1) \rceil$, which is $O(\log n)$

- $n \geq 1 + 2 + 4 + \ldots + 2^{h-2} + 1 = 2^{h-1} - 1 + 1 = 2^{h-1}$



Height of a Heap

- $n \leq 1 + 2 + 4 + \ldots + 2^{h-1} = 2^h - 1$

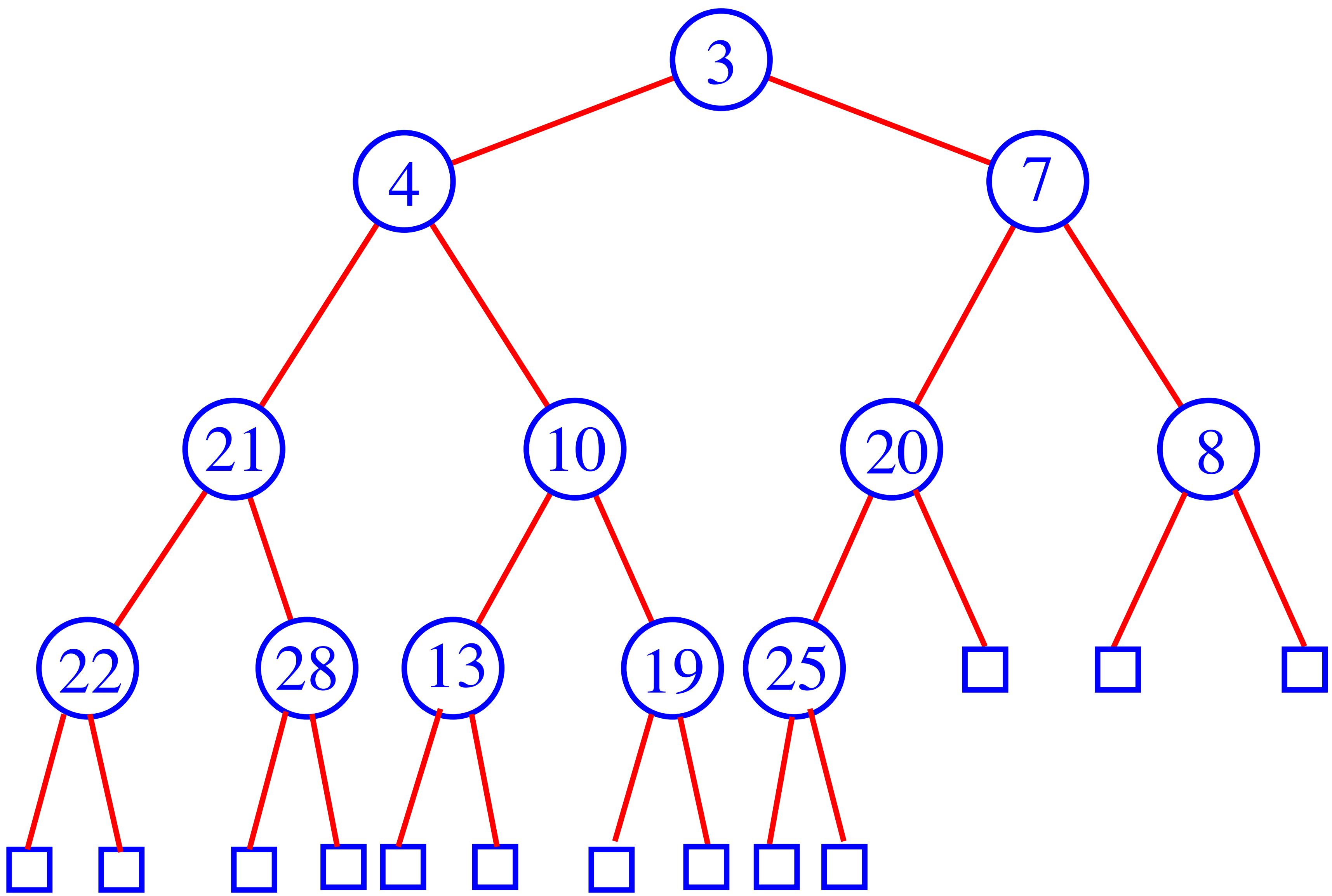


- Therefore $2^{h-1} \leq n \leq 2^h - 1$
- Taking logs, we get $\log(n+1) \leq h \leq \log n + 1$
- Which implies $\boldsymbol{h = \lceil \log(n+1) \rceil}$

Heap Insertion

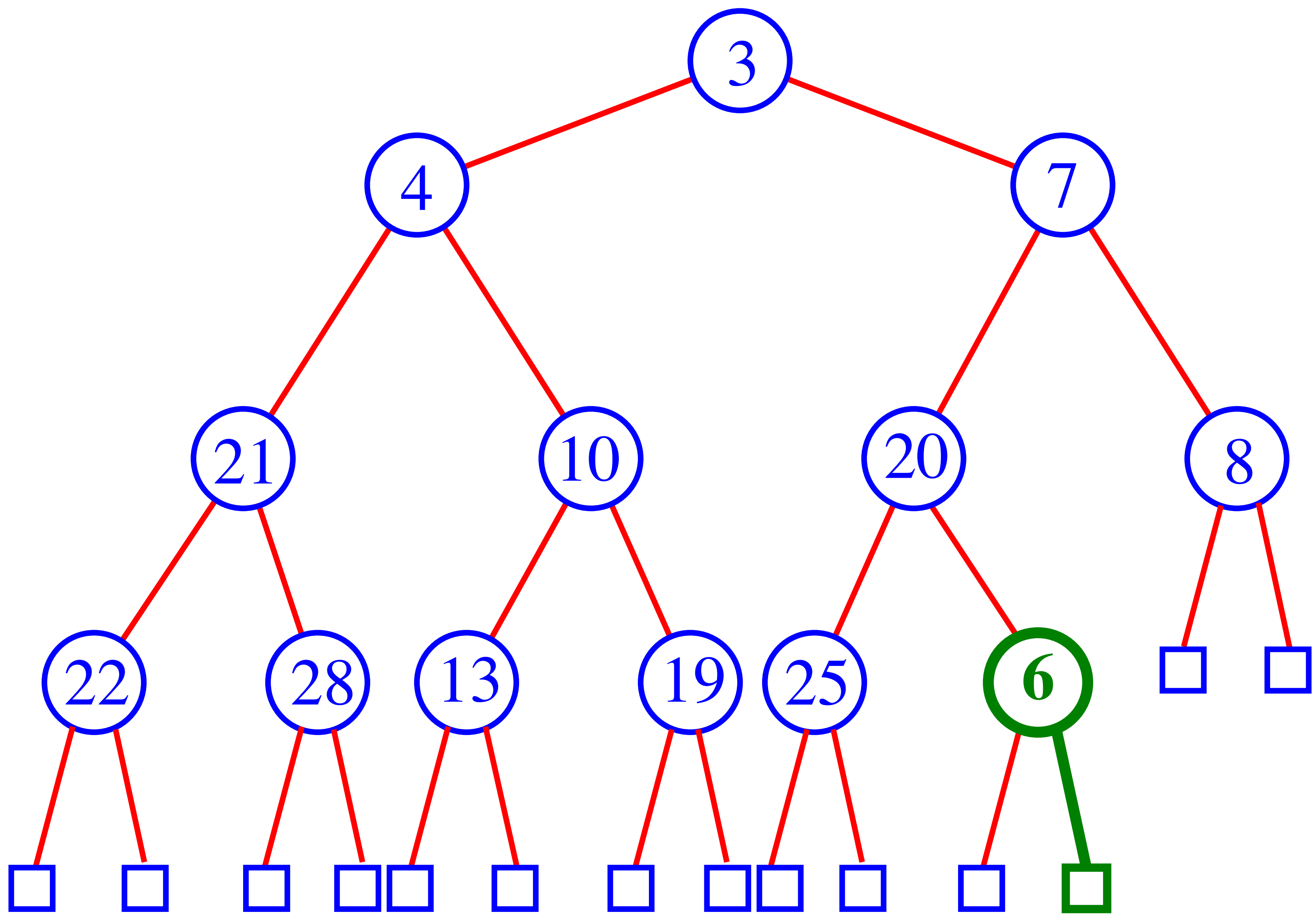
So here we go ...

The key to insert is **6**



Heap Insertion

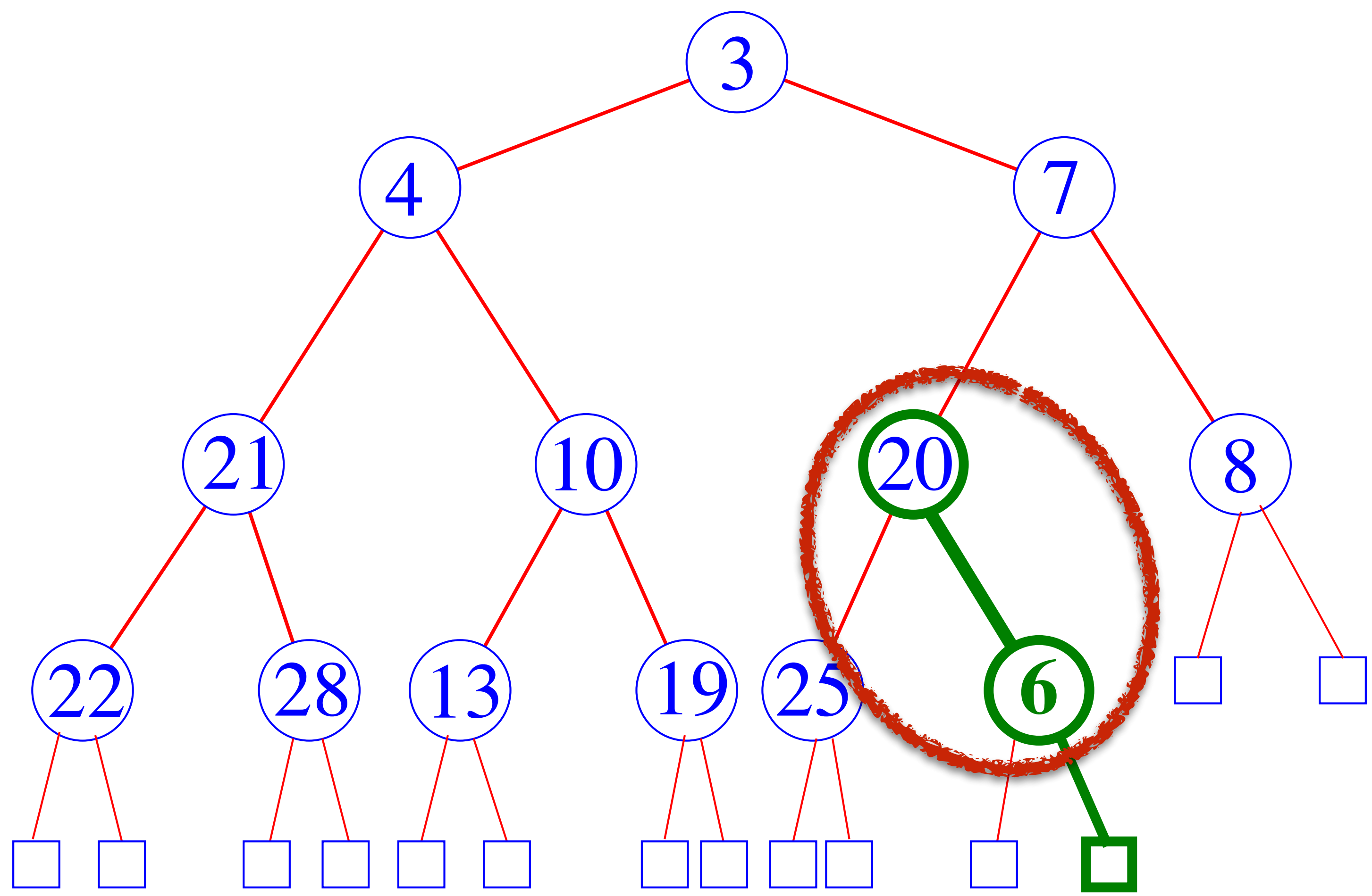
Add the key in the ***next available position*** in the heap.



Now begin ***Upheap***.

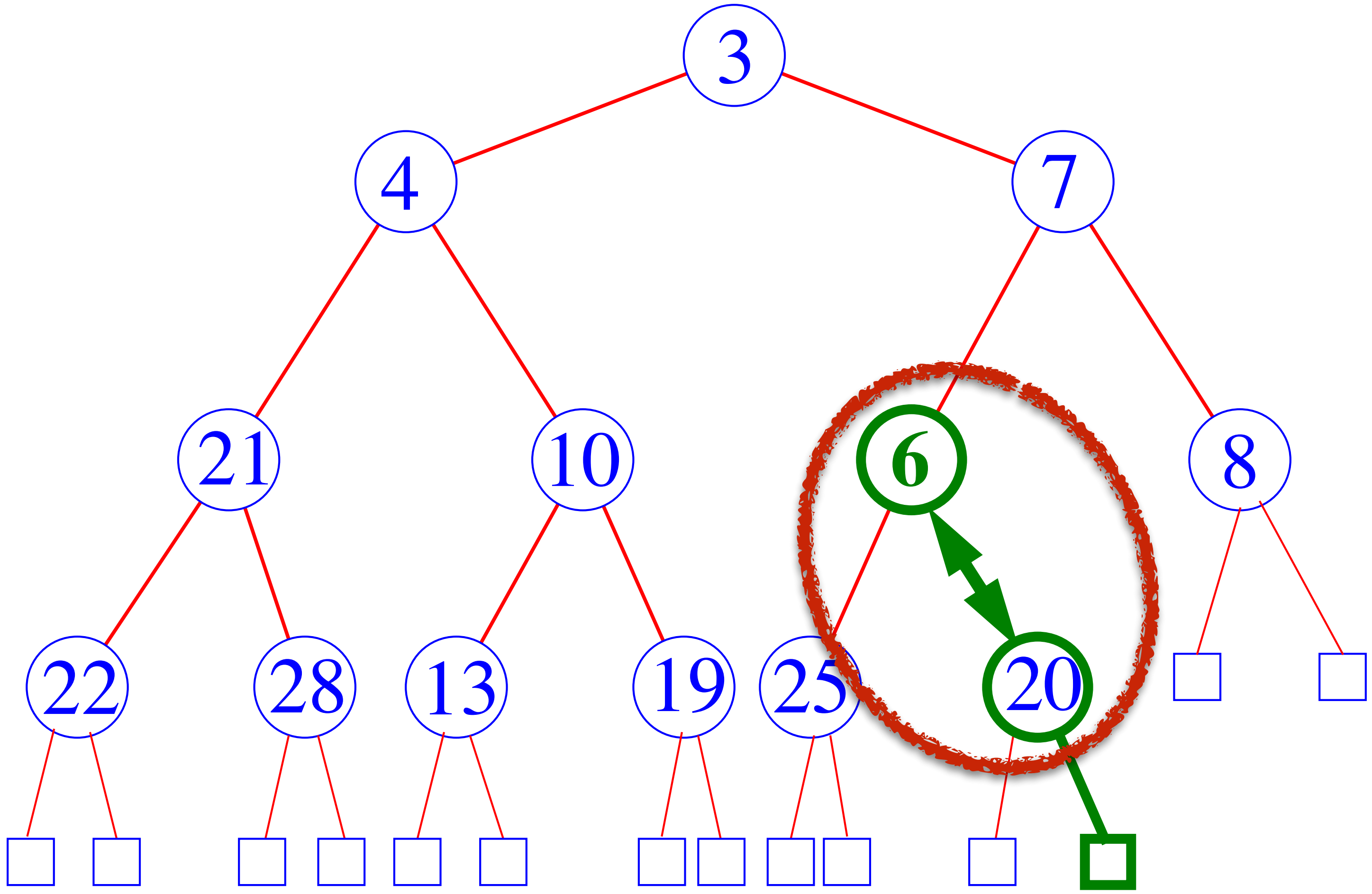
Upheap

- *Swap parent-child keys out of order*

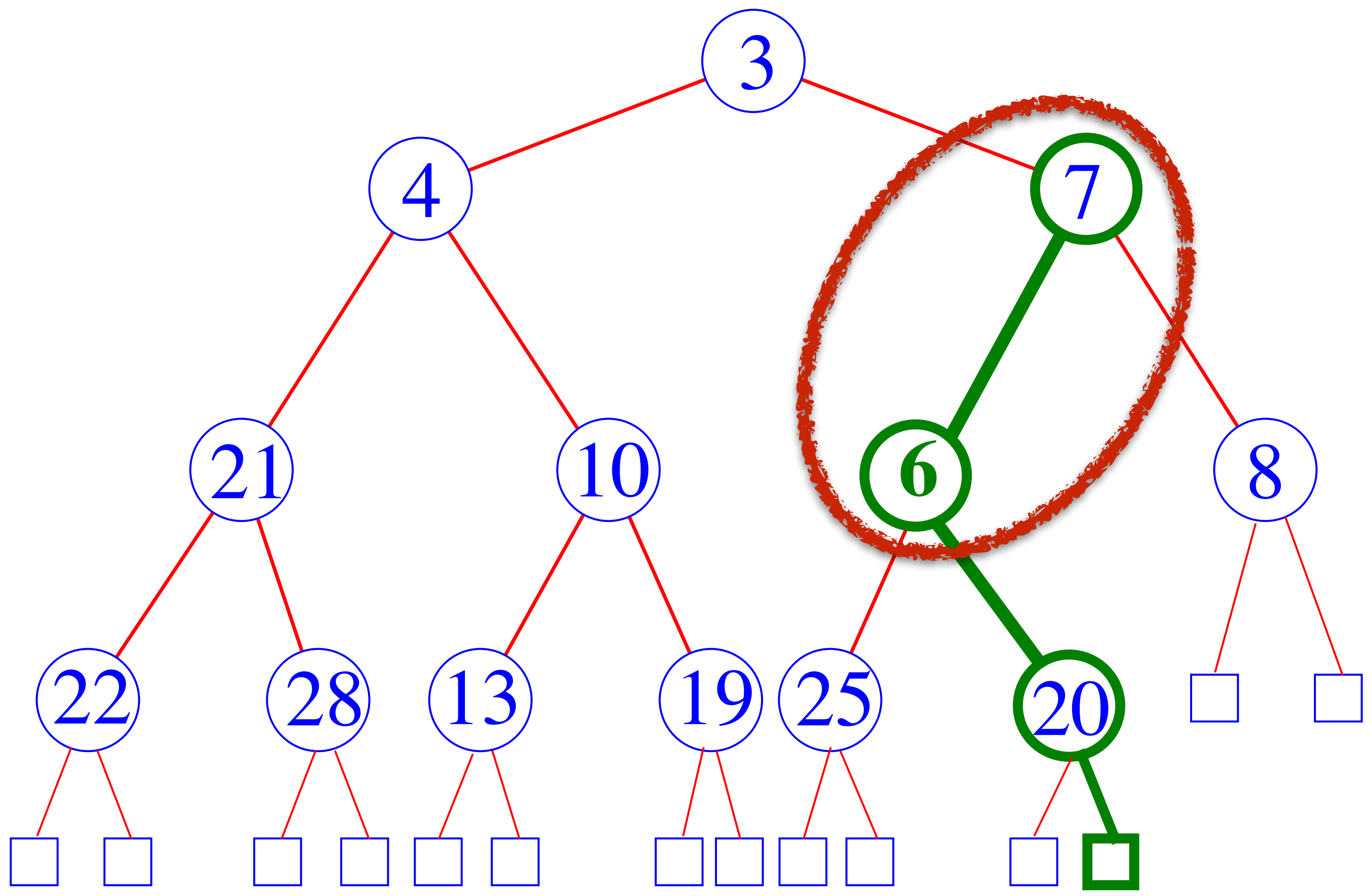


Upheap

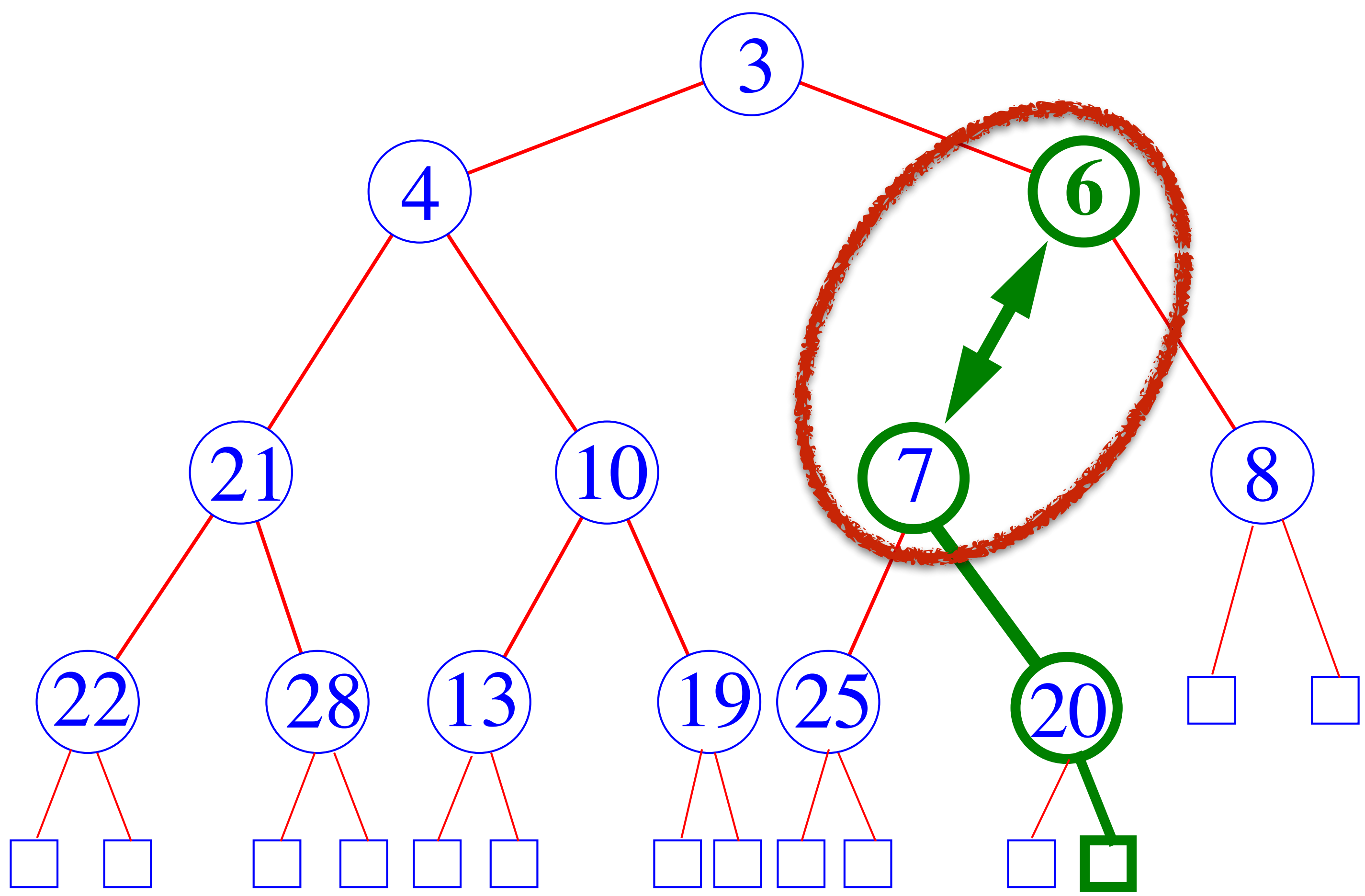
- ***Swap parent-child keys out of order***



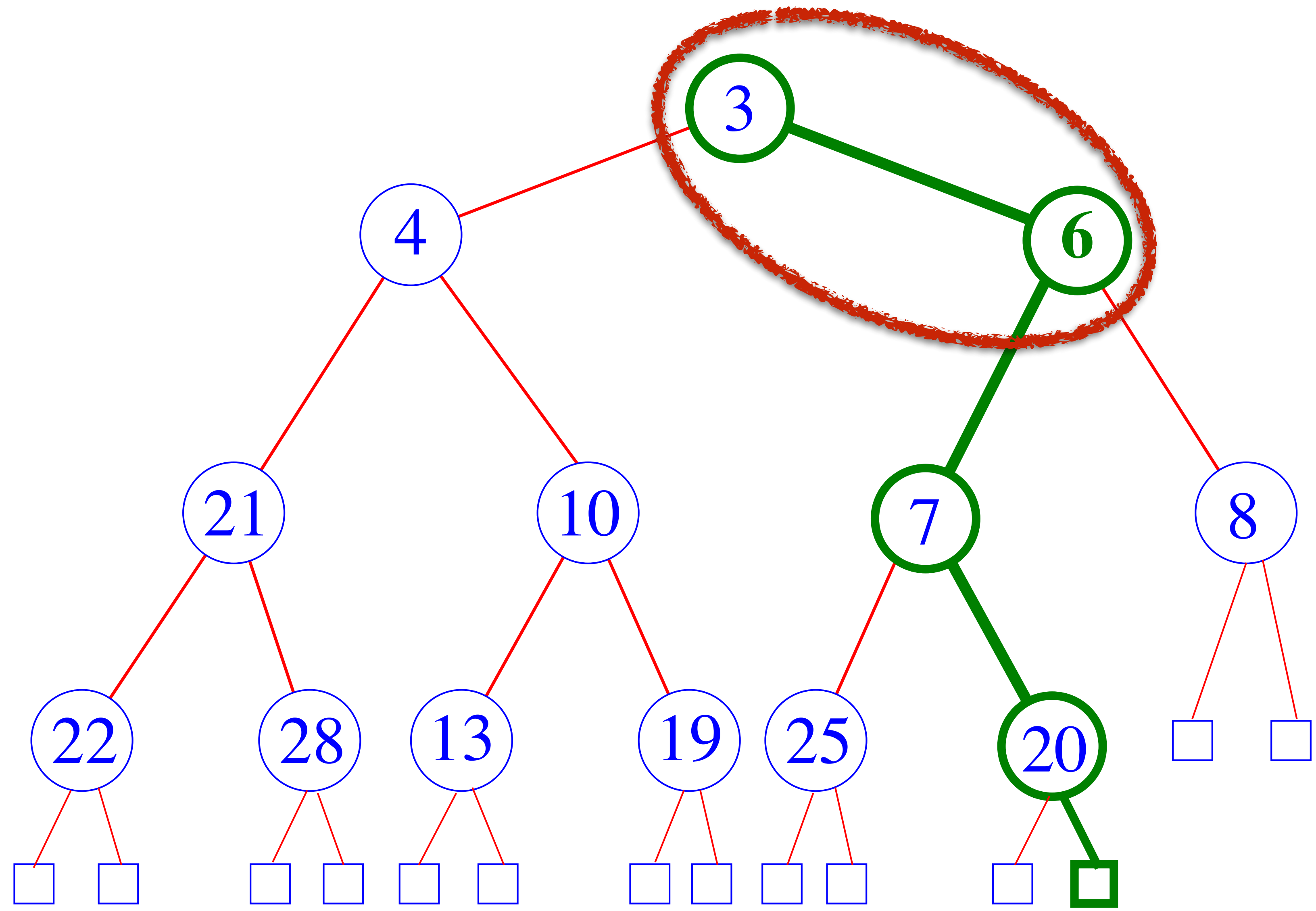
Upheap Continues



Upheap Continues



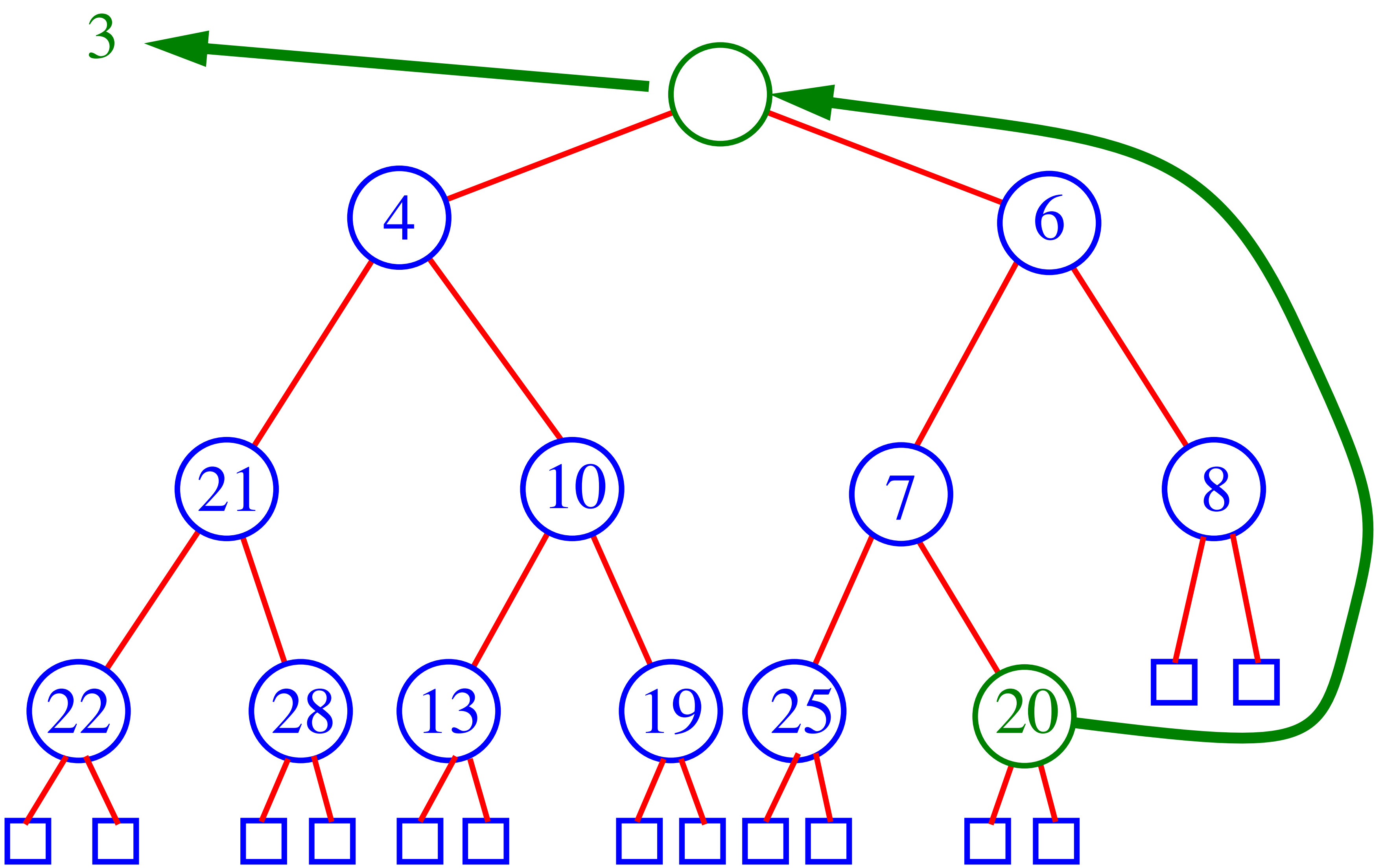
End of Upheap

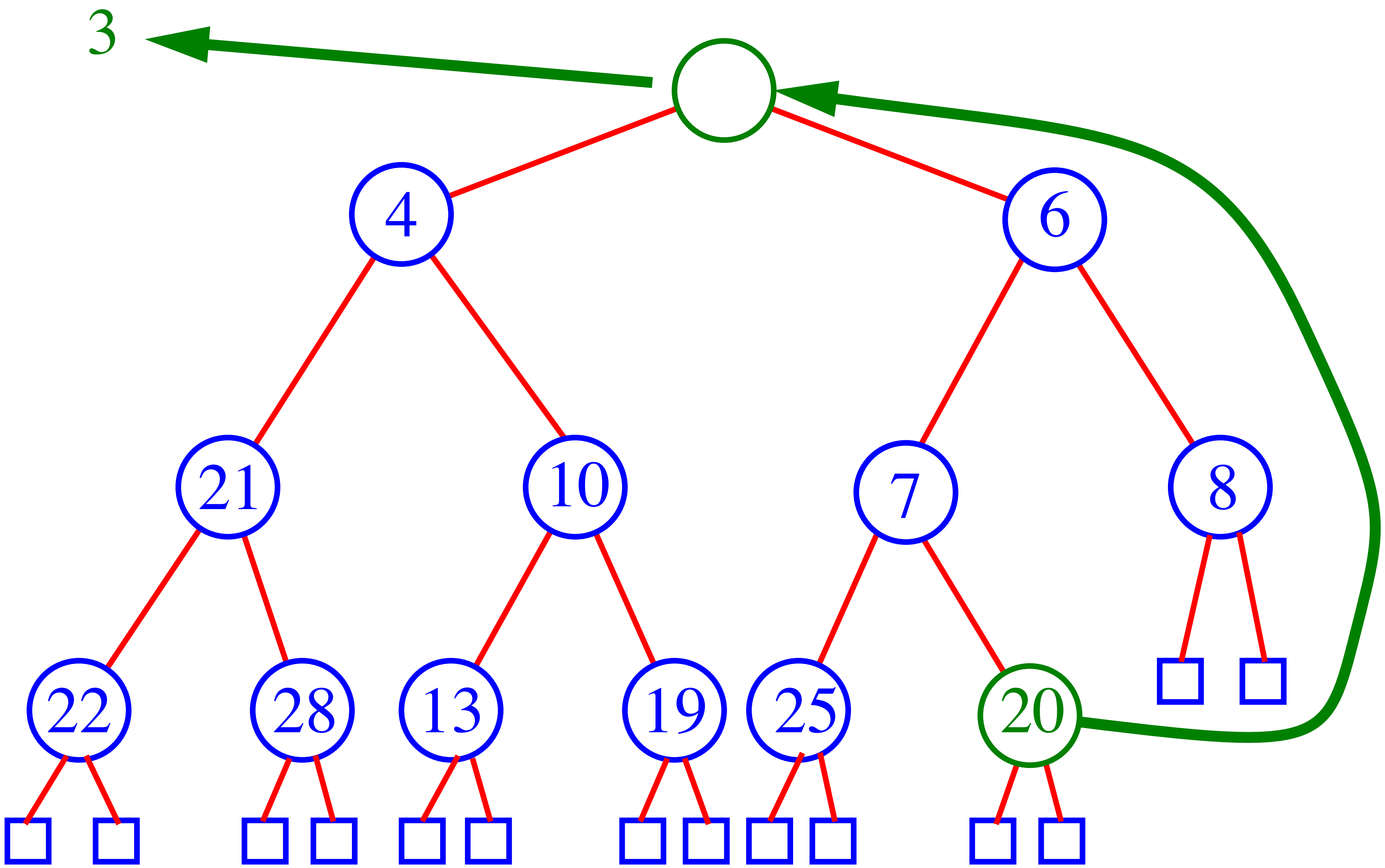


- ***Upheap*** terminates when new key is greater than the key of its parent **or** the top of the heap is reached
- (total #swaps) $\leq$ $(h - 1)$, which is $O(\log n)$

Removal From a Heap

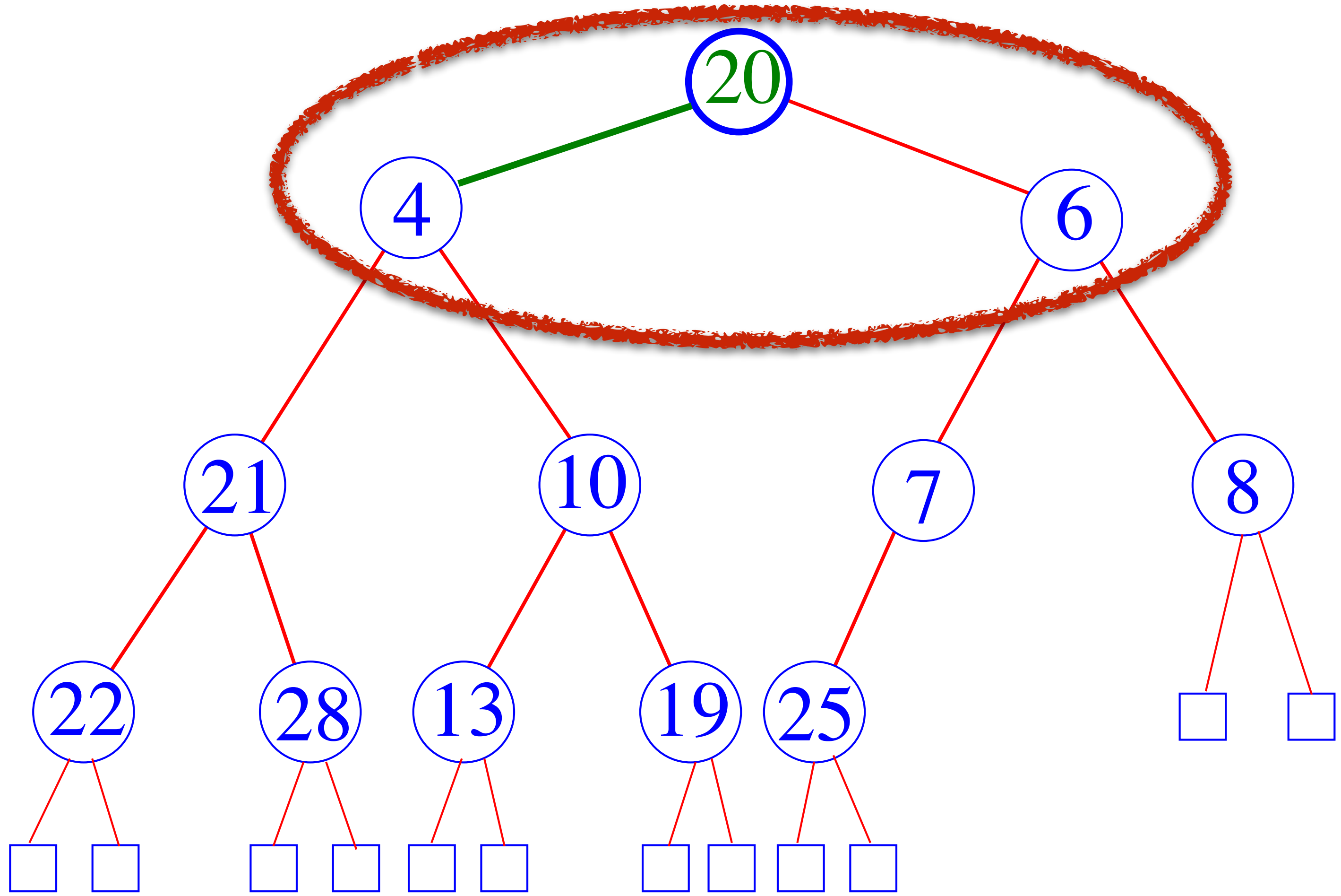
RemoveMin()



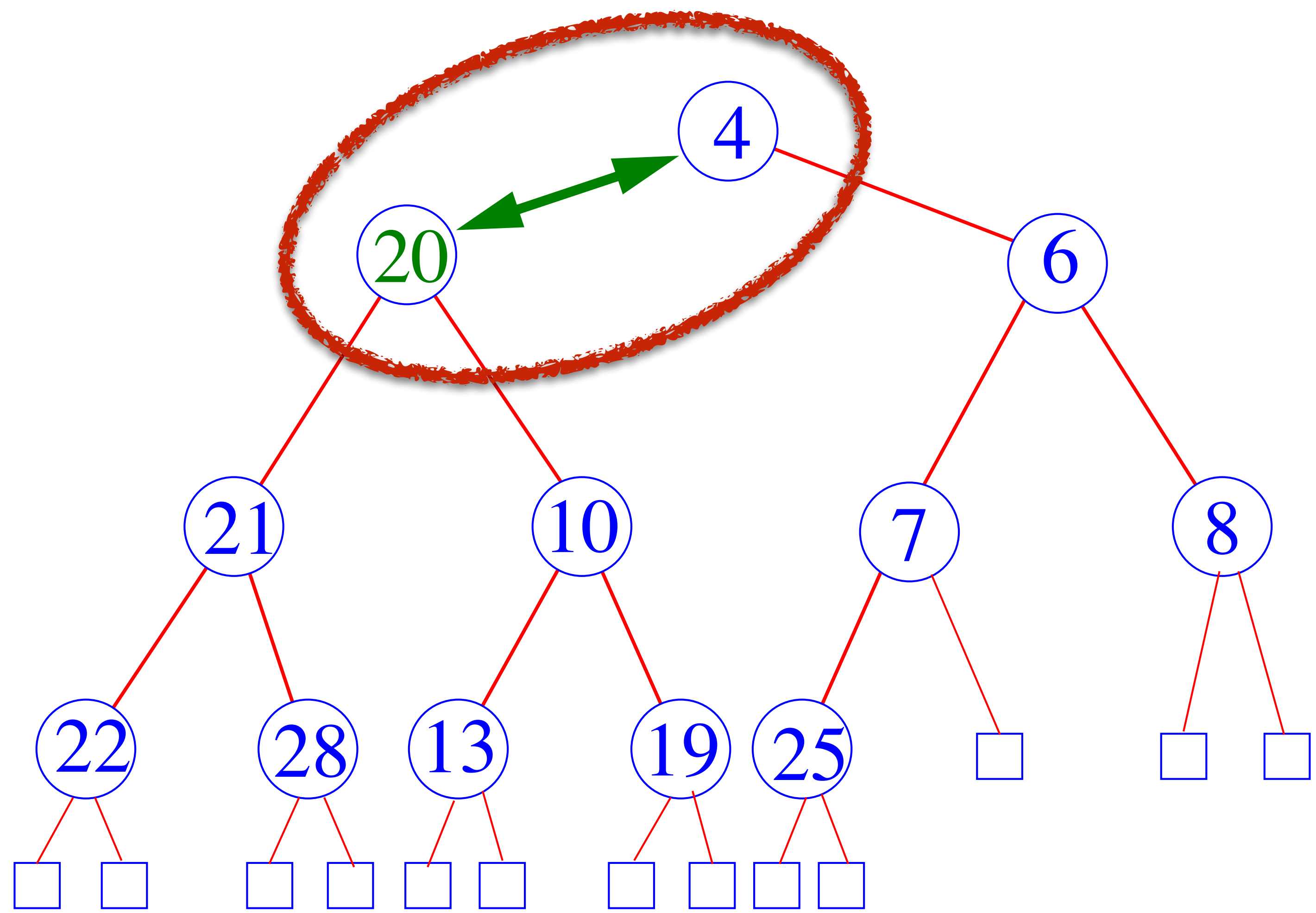


- The removal of the top key leaves a hole
- We need to fix the heap
- First, replace the hole with the last key in the heap
- Then, begin ***Downheap***

Downheap

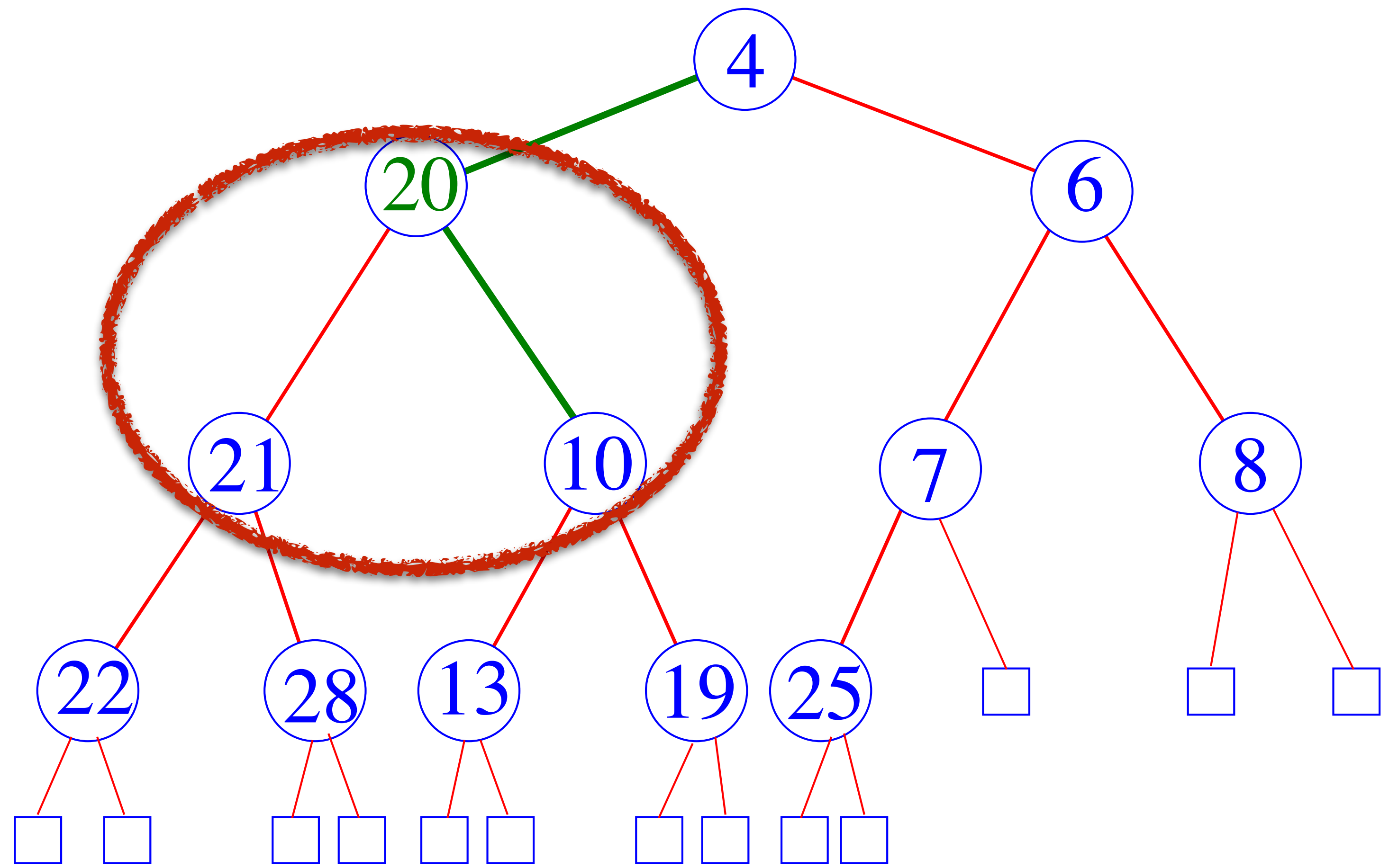


Downheap

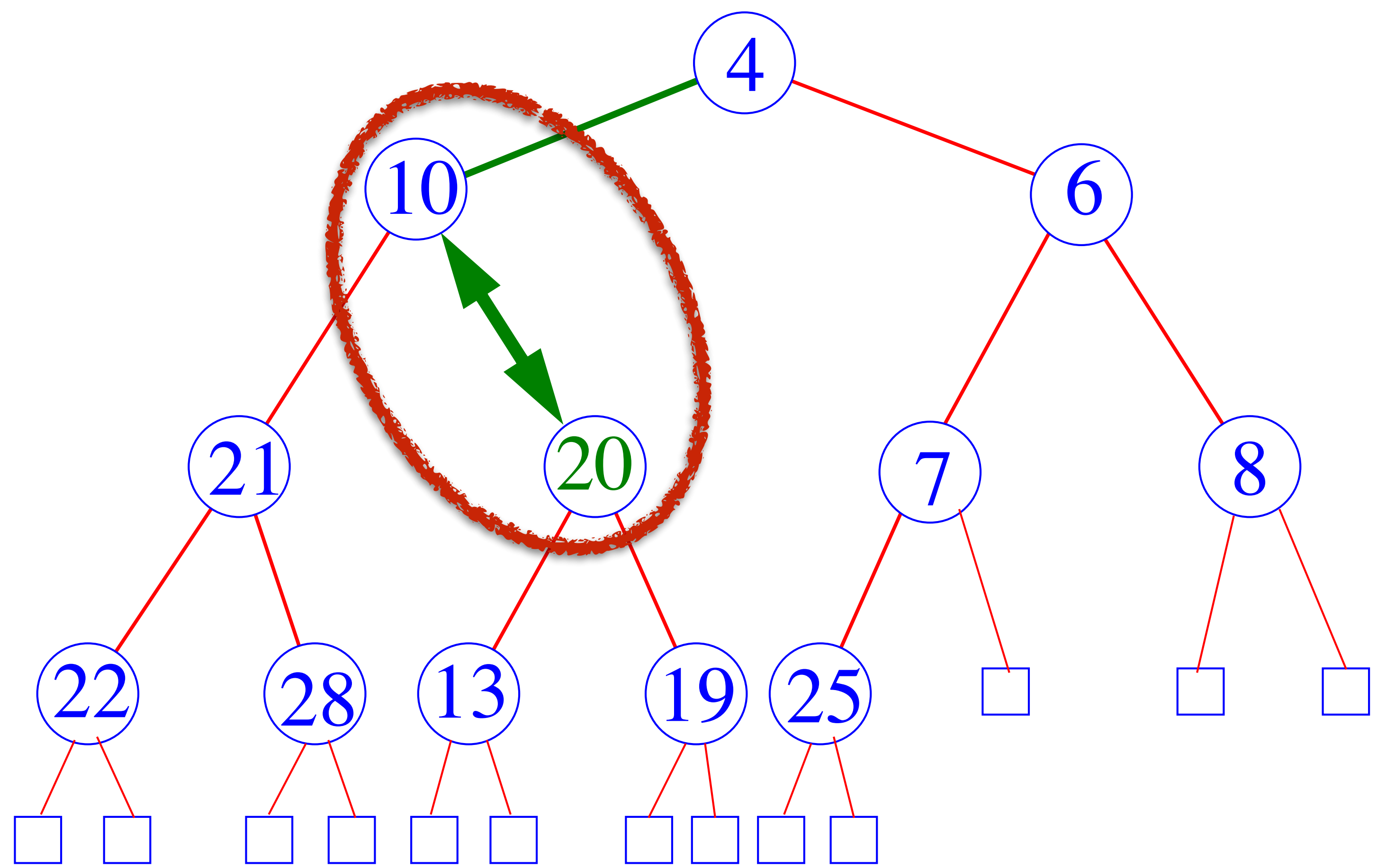


Downheap compares the parent with the smallest child. If the child is smaller, it switches the two.

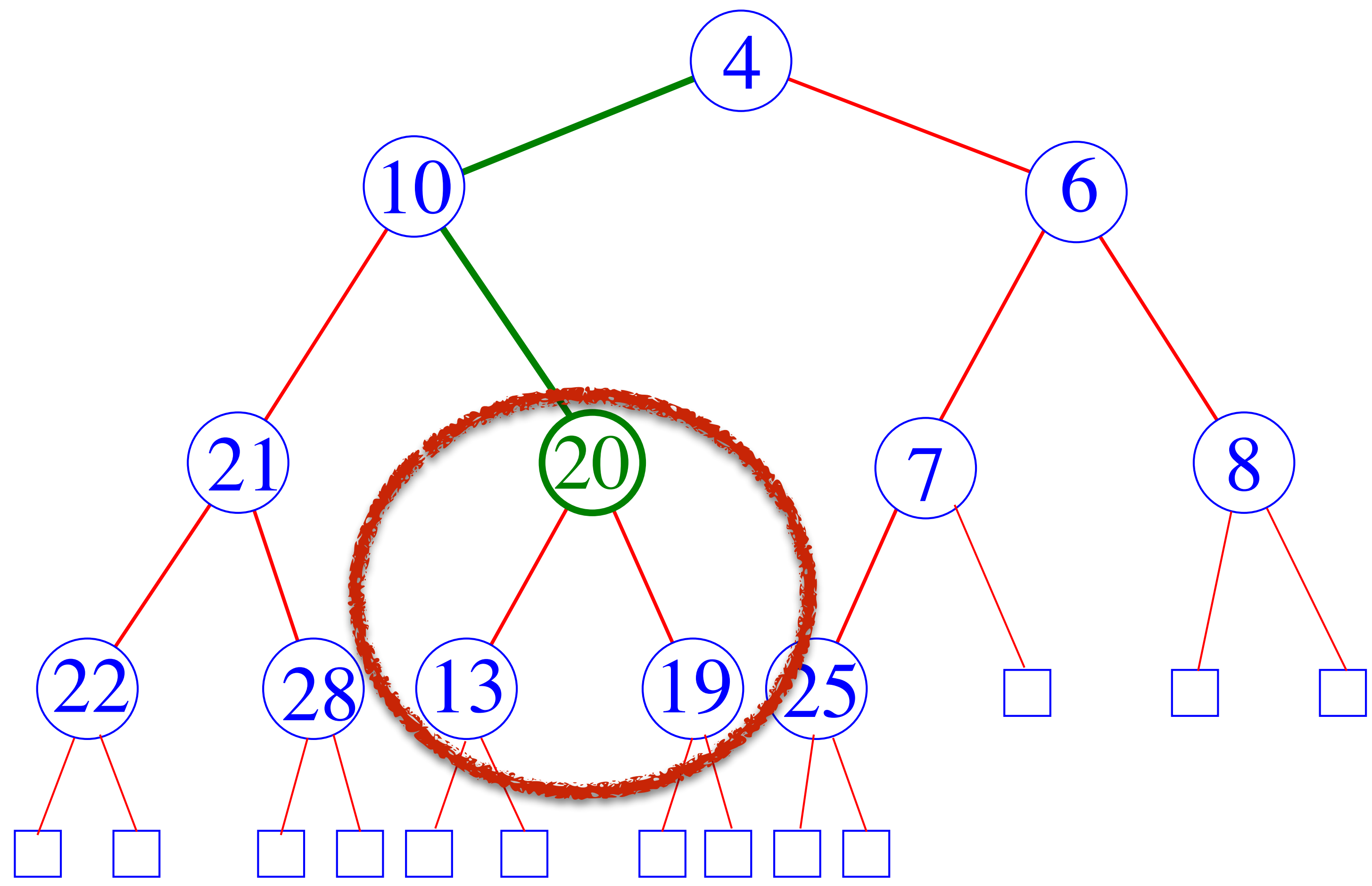
Downheap Continues



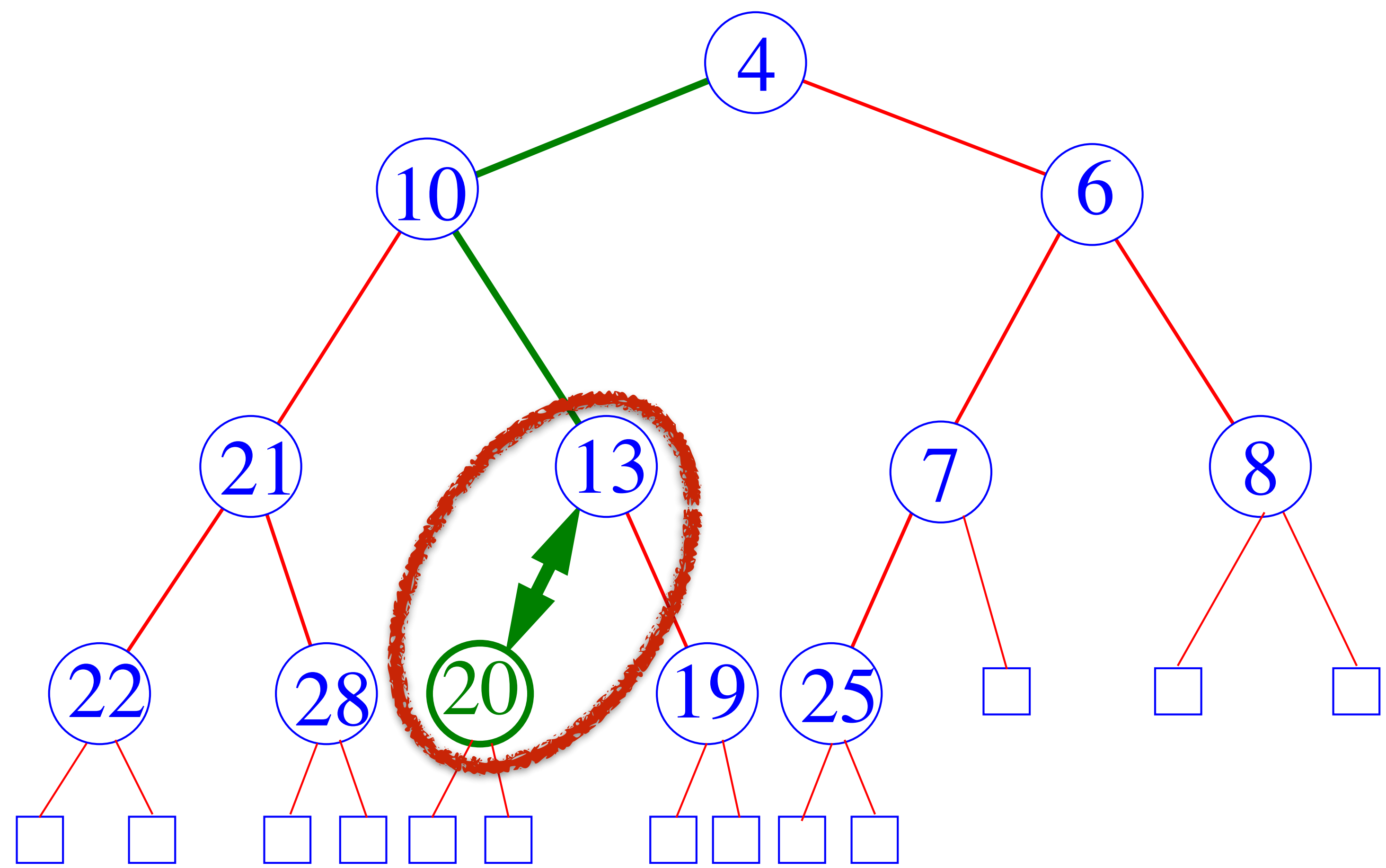
Downheap Continues



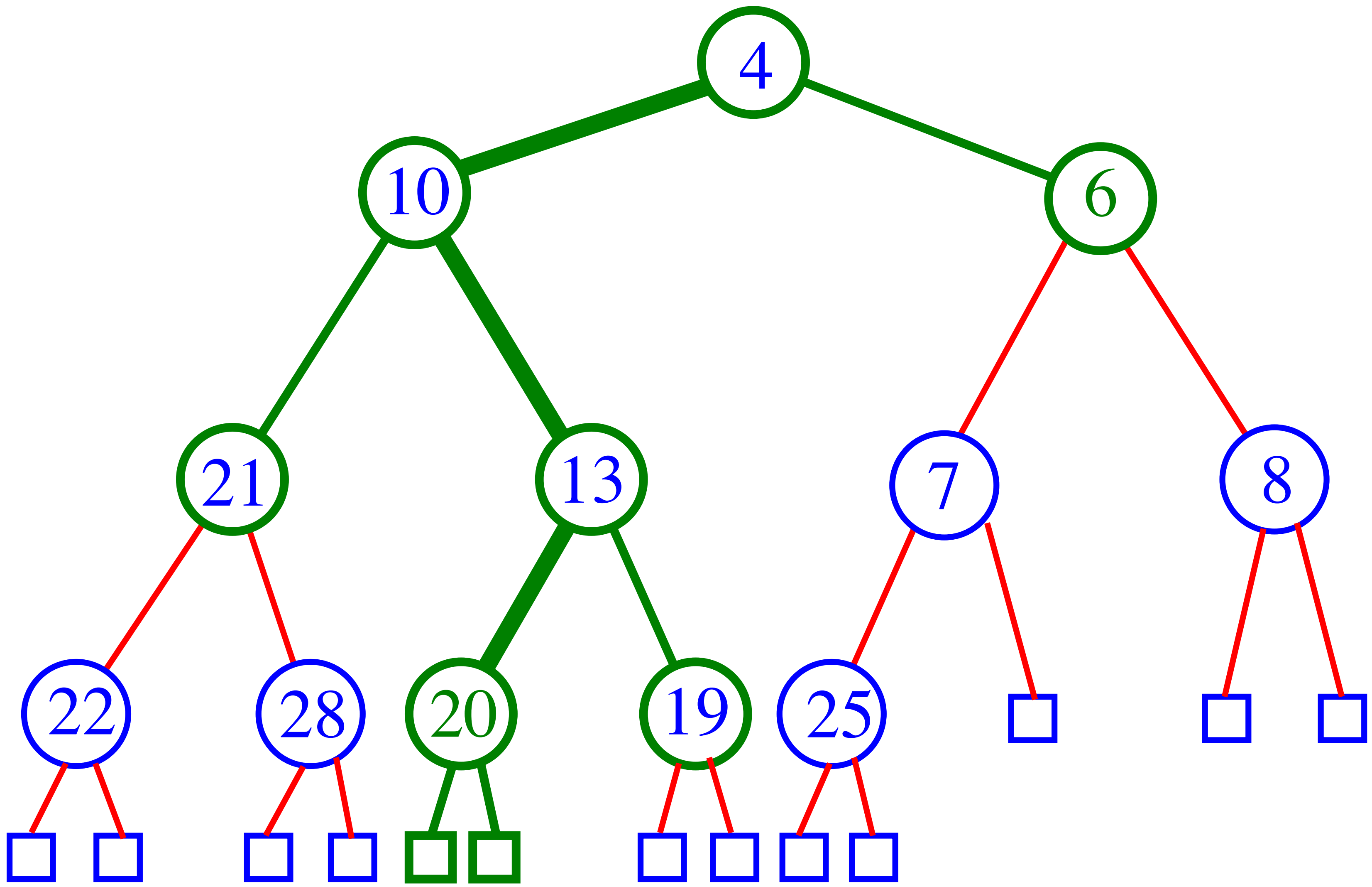
Downheap Continues



Downheap Continues



End of Downheap

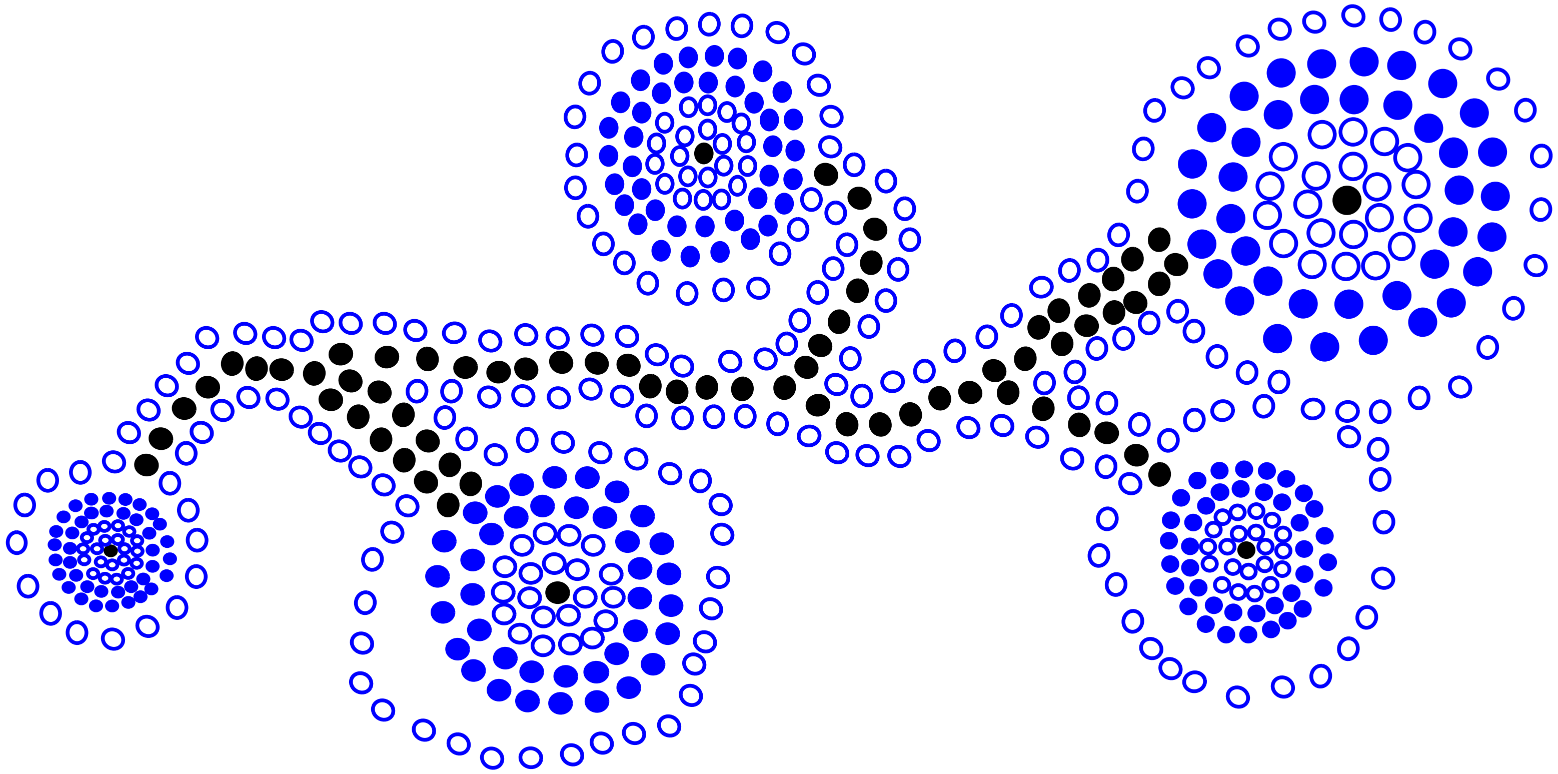


- ***Downheap*** terminates when the key is greater than the keys of both its children **or** the bottom of the heap is reached.

- (total #swaps) $\leq$ $(h - 1)$, which is $O(\log n)$

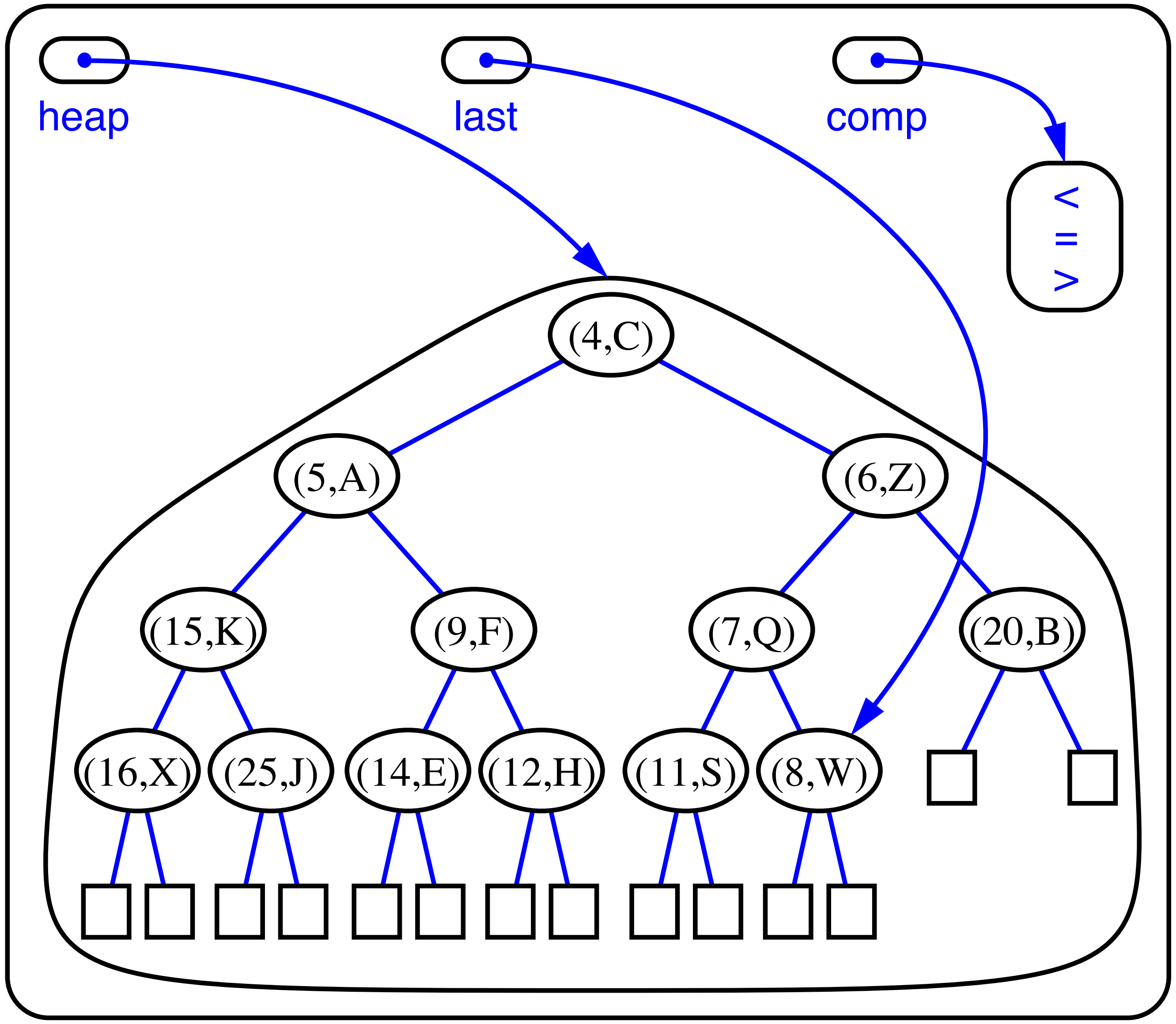
HEAPS II

- Implementation
- HeapSort
- Bottom-Up Heap Construction
- Locators



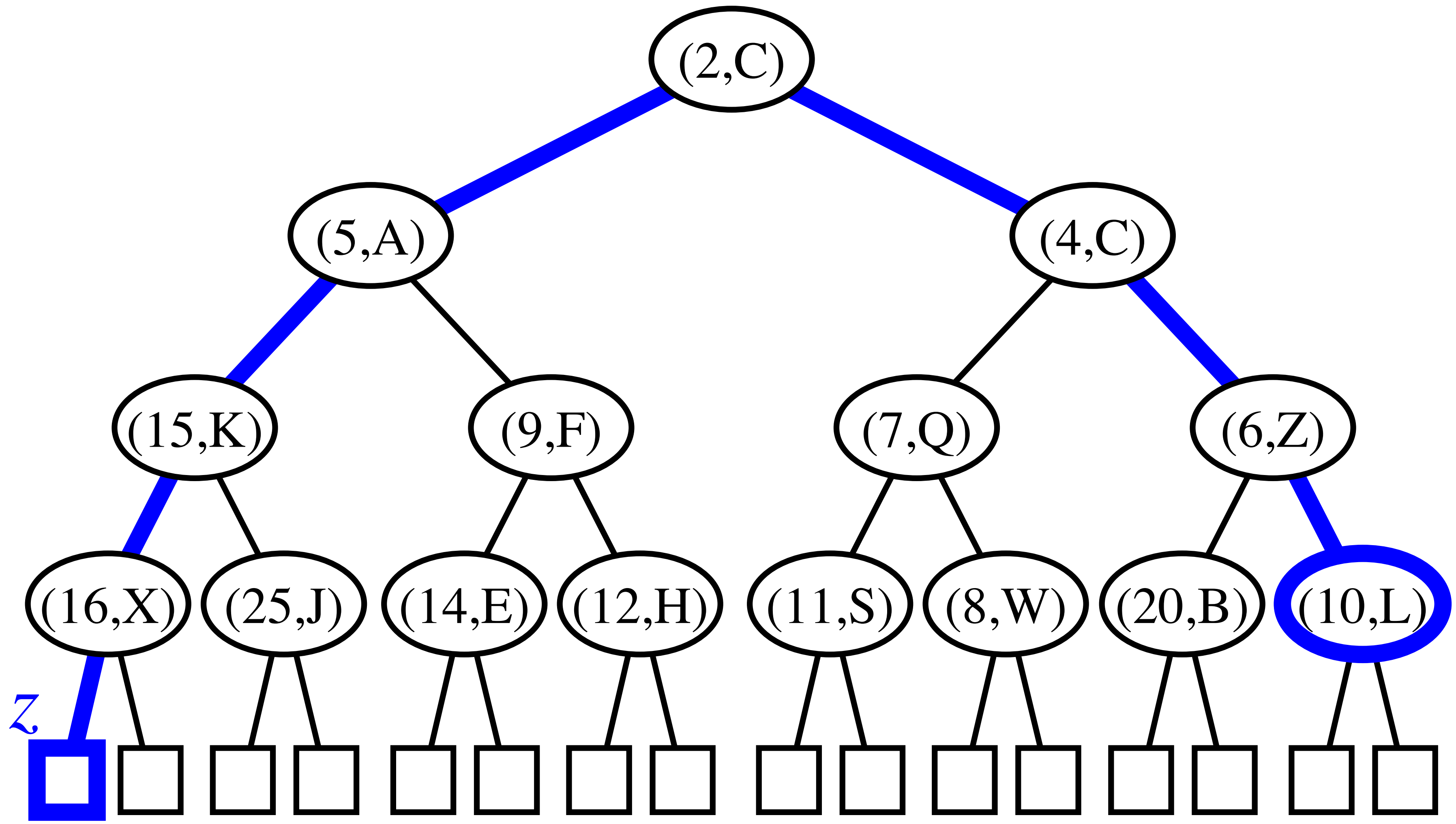
Implementation of a Heap

```
public class HeapPriorityQueue implements PriorityQueue
{
   BinaryTree T;
   Position last;
   Comparator comparator;
   ...
}
```



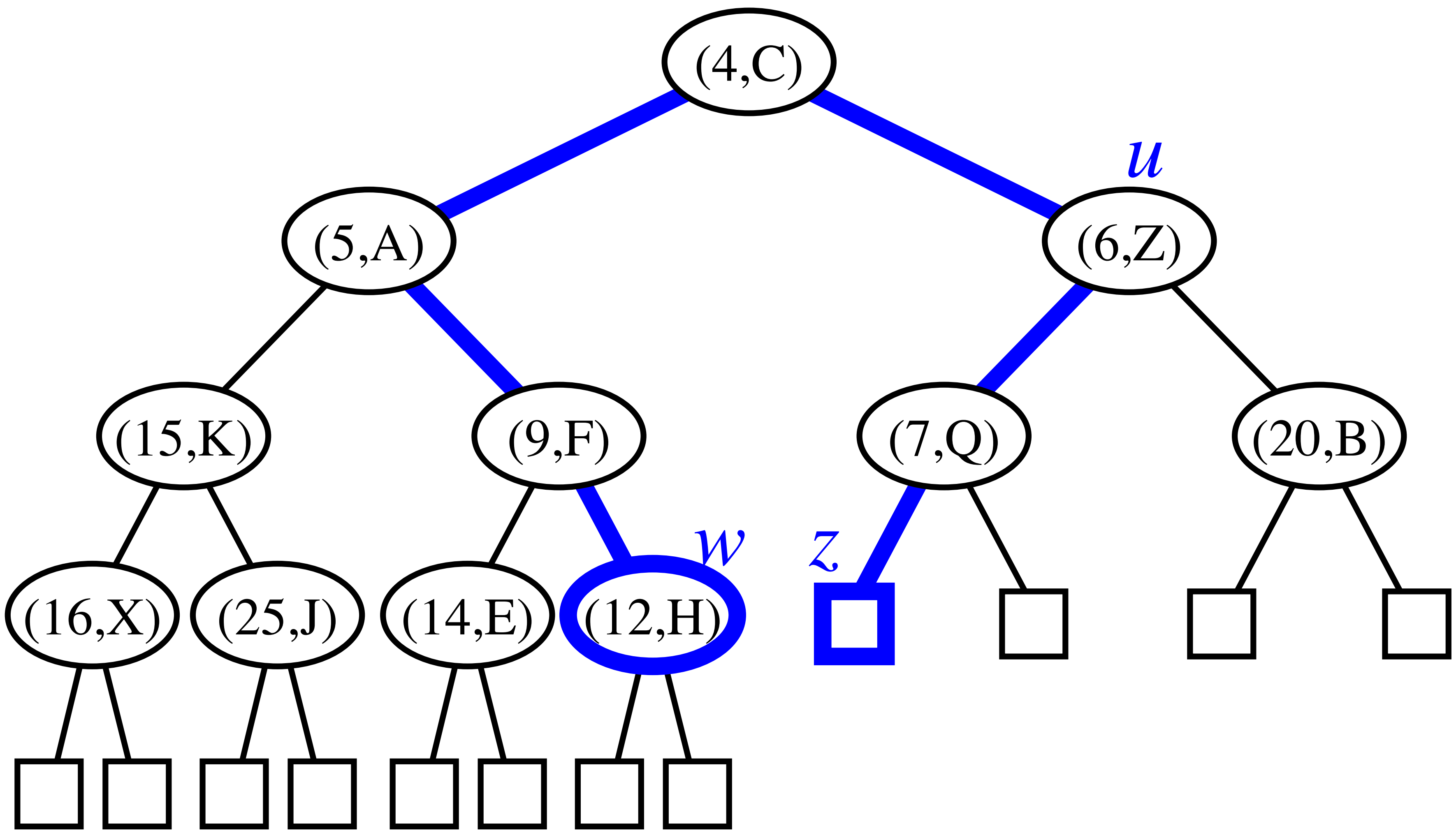
Implementation of a Heap(cont.)

- Two ways to find the insertion position z in a heap:



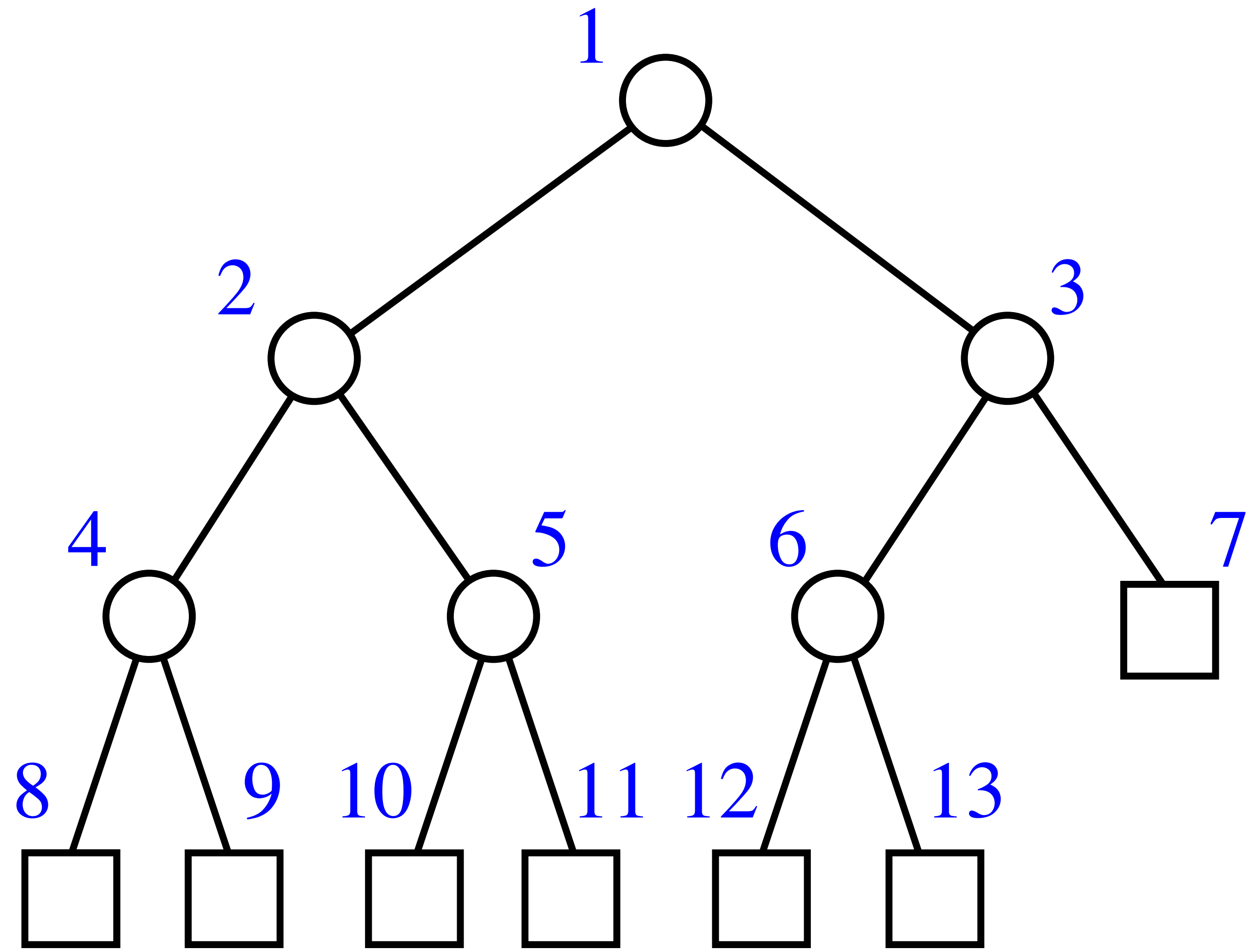
Implementation of a Heap(cont.)

- Two ways to find the insertion position z in a heap:

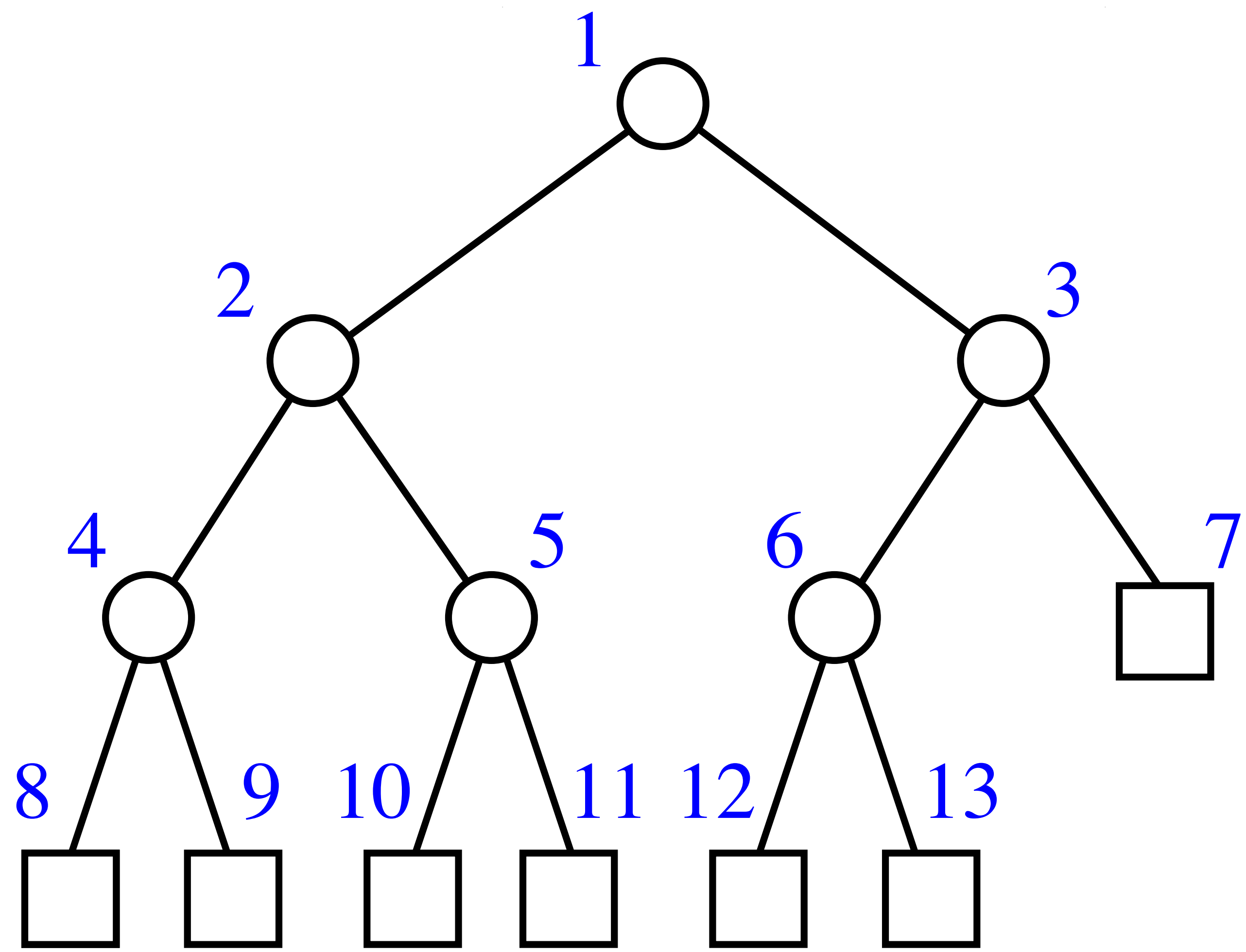


Vector Based Implementation

- Updates in the underlying tree occur only at the "last element"
- A heap can be represented by a vector, where the node at rank i has
  - left child at rank $2i$ and
  - right child at rank $2i + 1$



Vector Based Implementation



- The leaves do no need to be explicitly stored
- Insertion and removals into/from the heap correspond to insertLast and removeLast on the vector, respectively

Heap Sort

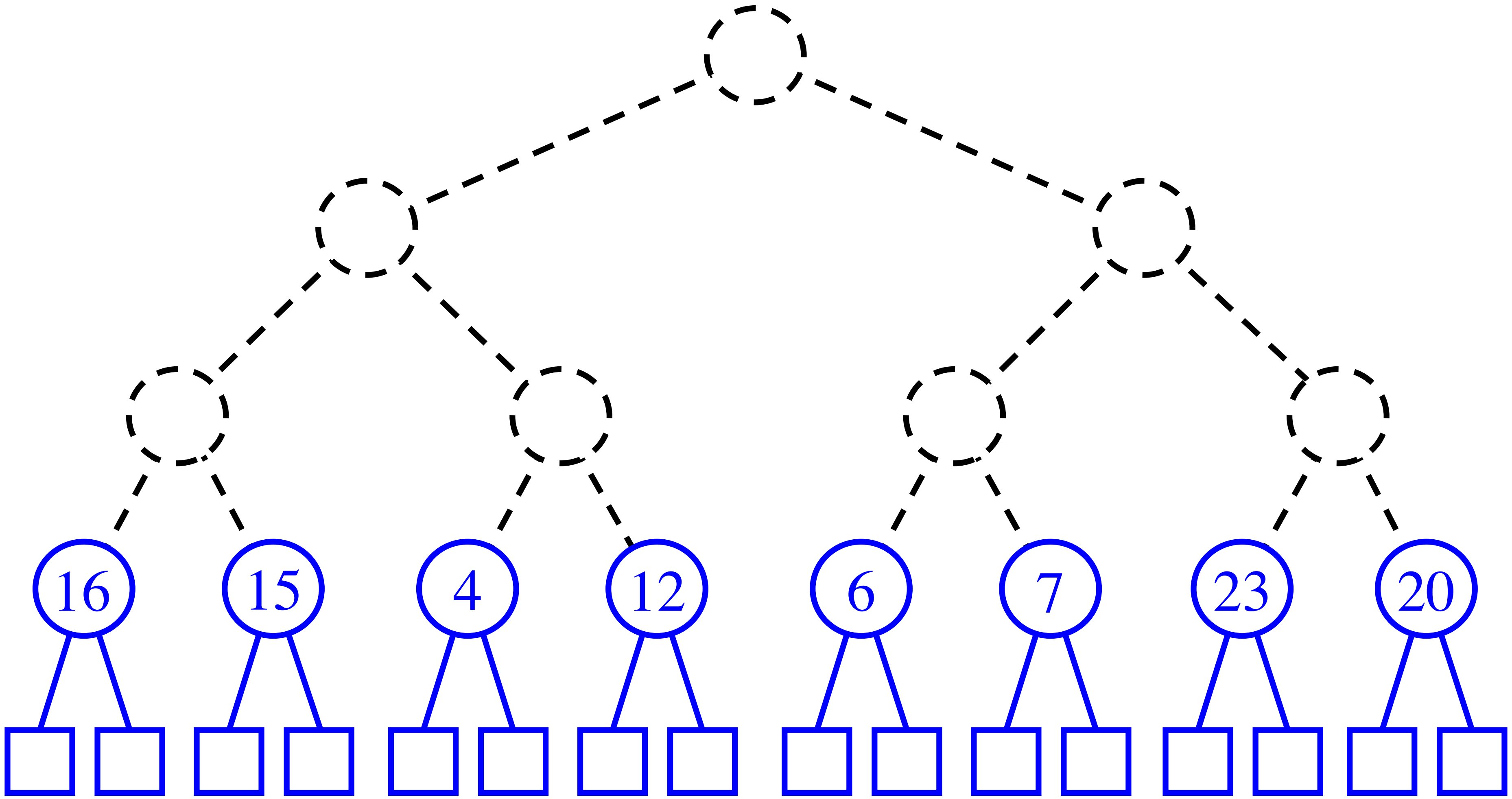
- All heap methods run in logarithmic time or better
- If we implement PriorityQueueSort using a heap for our priority queue, insertItem and removeMin each take O(log k), k being the number of elements in the heap at a given time.
- We always have at most n elements in the heap, so the worst case time complexity of these methods is O(log n).
- Thus each phase takes O(n log n) time, so the algorithm runs in O(n log n) time also.
- This sort is known as ***heap-sort***.
- The O(n log n) run time of heap-sort is much better than the O(n^2) run time of selection and insertion sort.

In-Place Heap-Sort

- Do not use an external heap
- Embed the heap into the sequence, using the vector representation

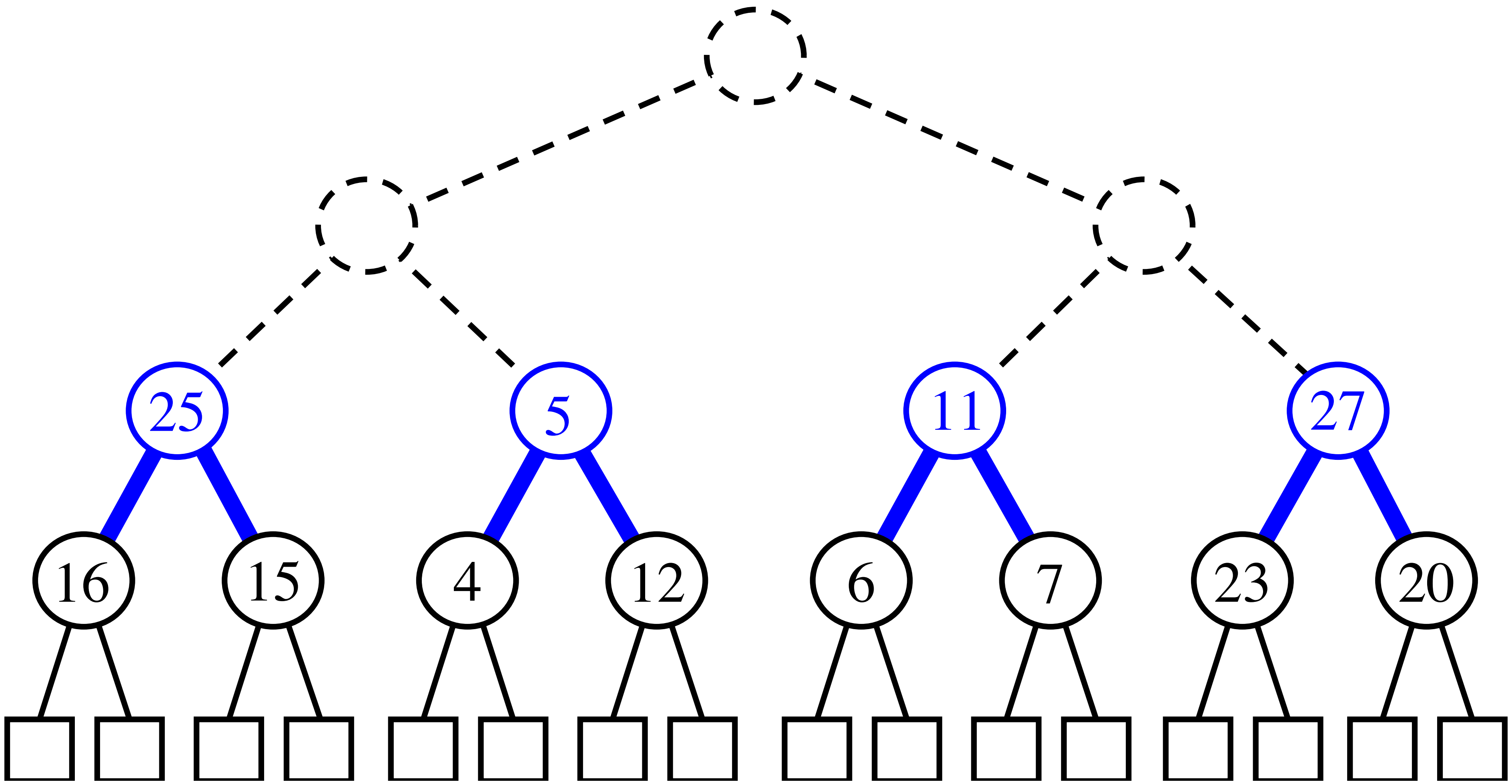
Bottom-Up Heap Construction

- build $(n + 1)/2$ trivial one-element heaps



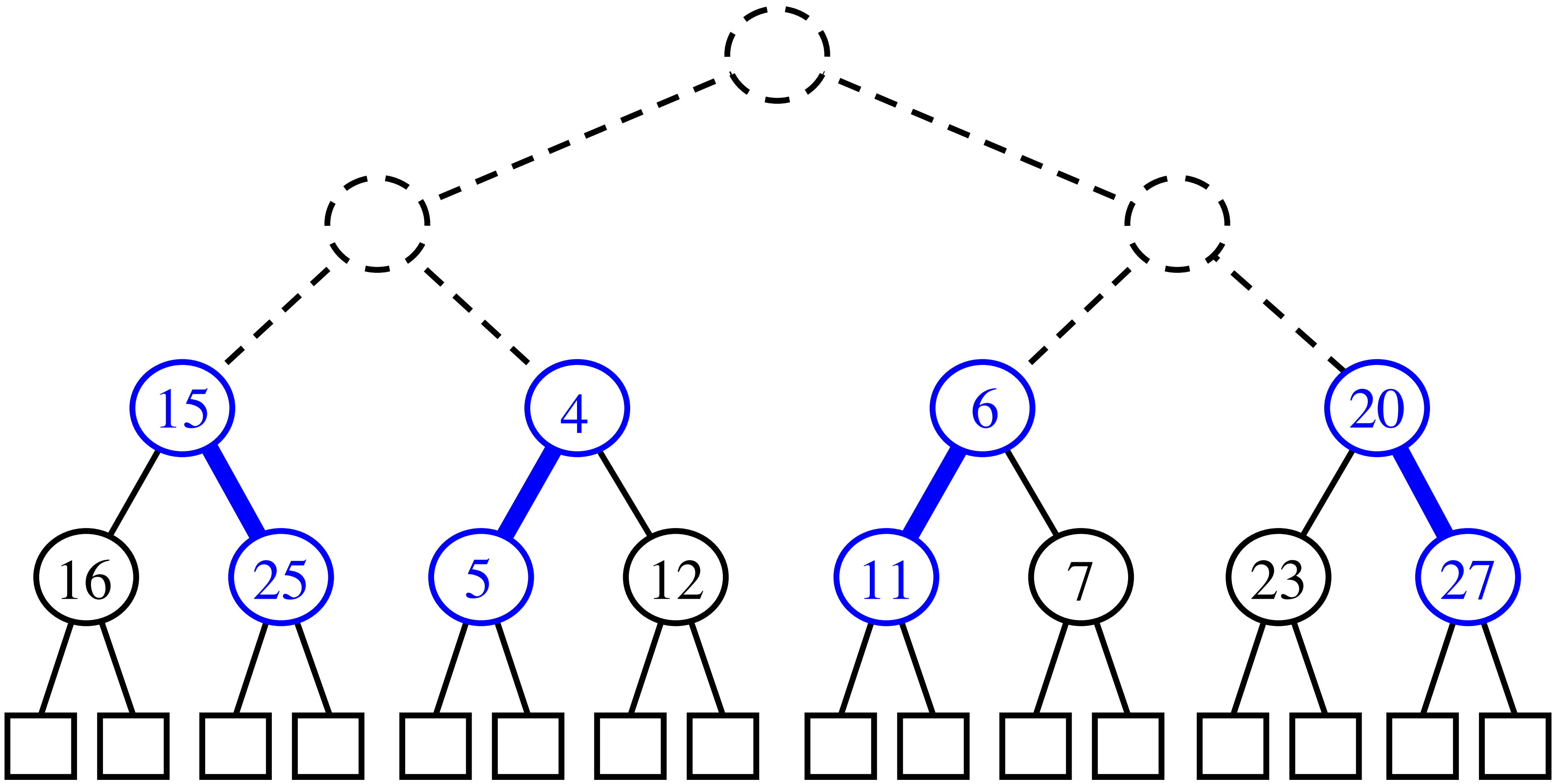
Bottom-Up Heap Construction

- now build three-element heaps on top of them



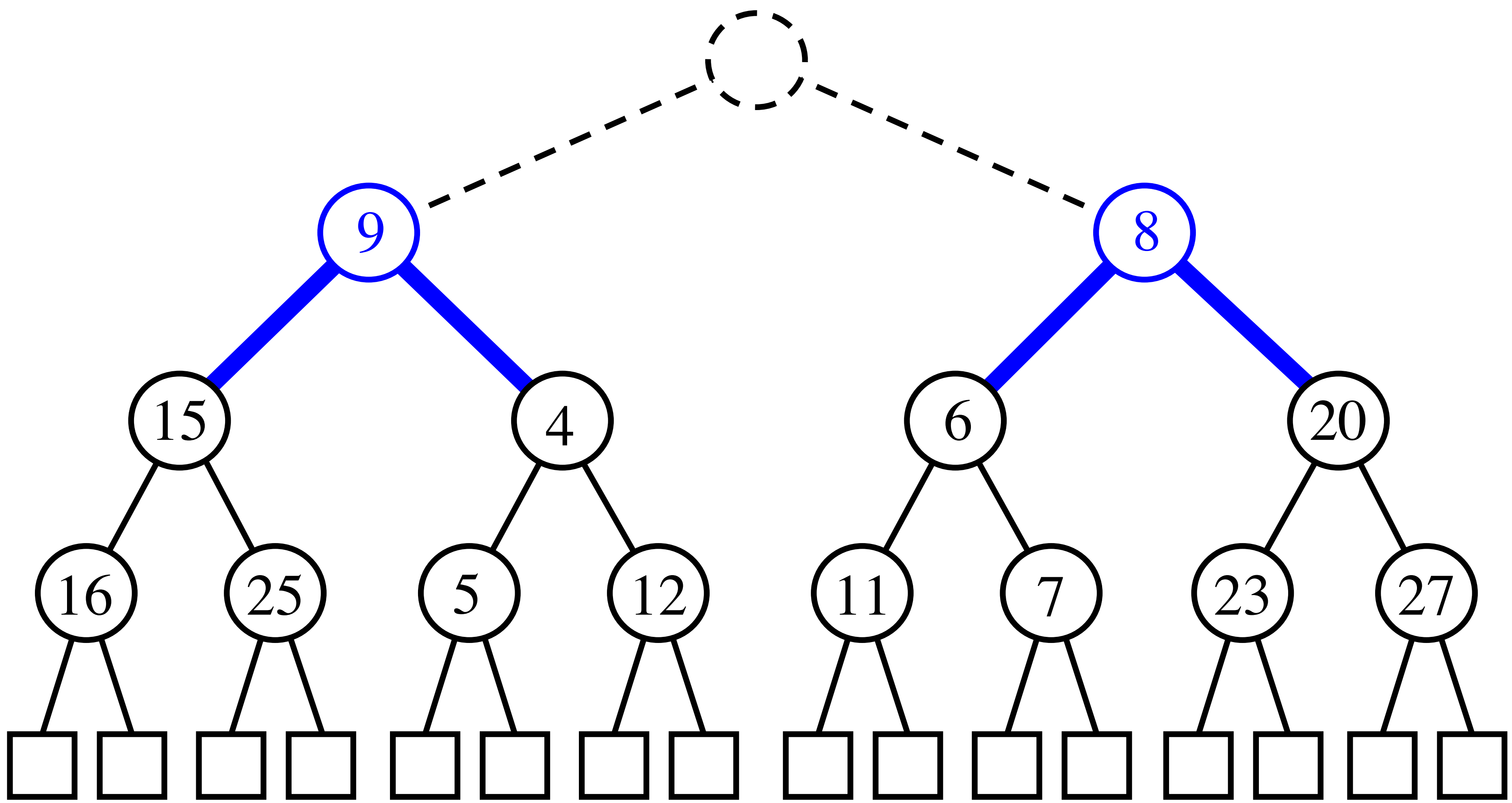
Bottom-Up Heap Construction

- ***downheap*** to preserve the order property

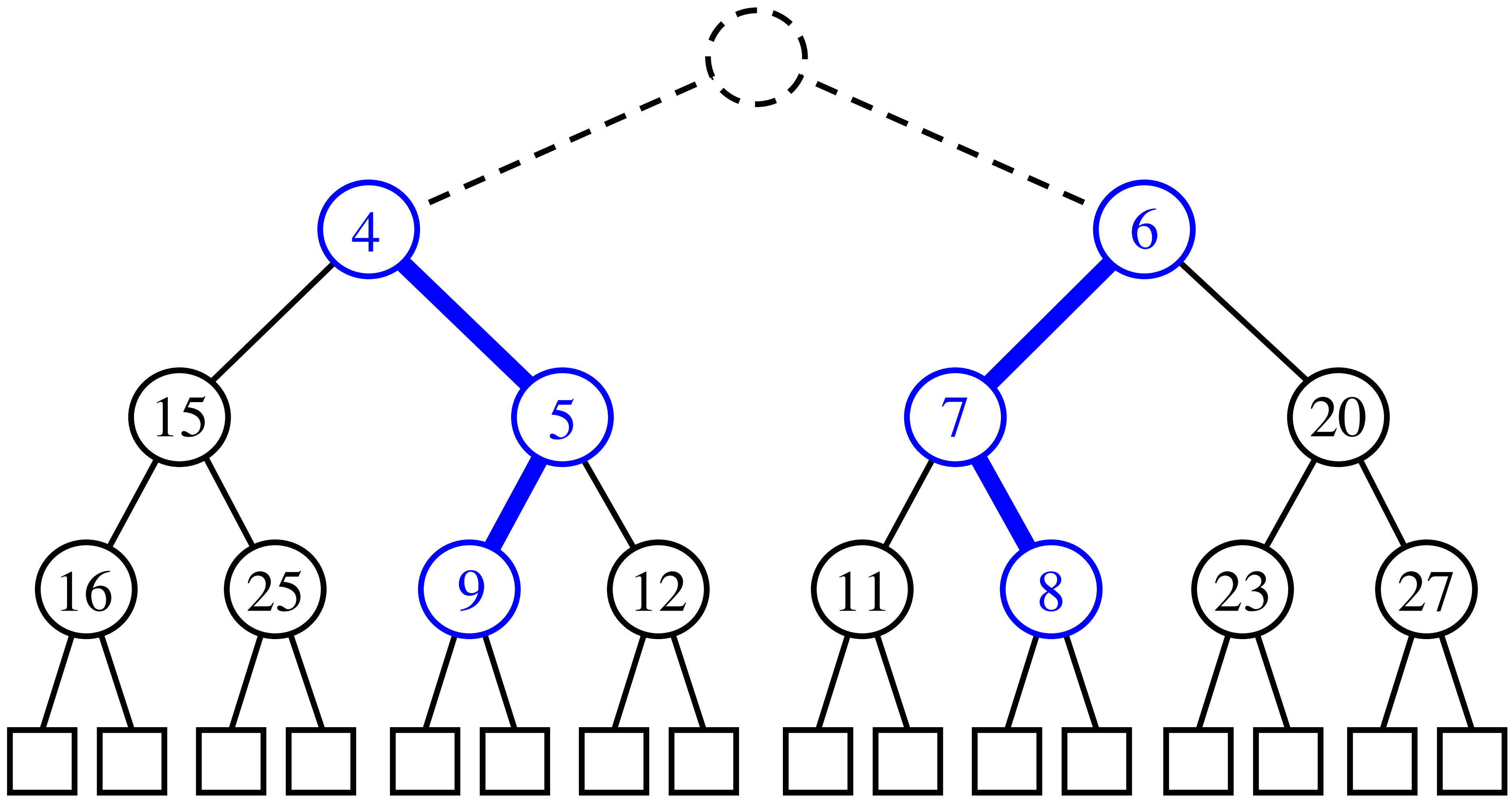


Bottom-Up Heap Construction

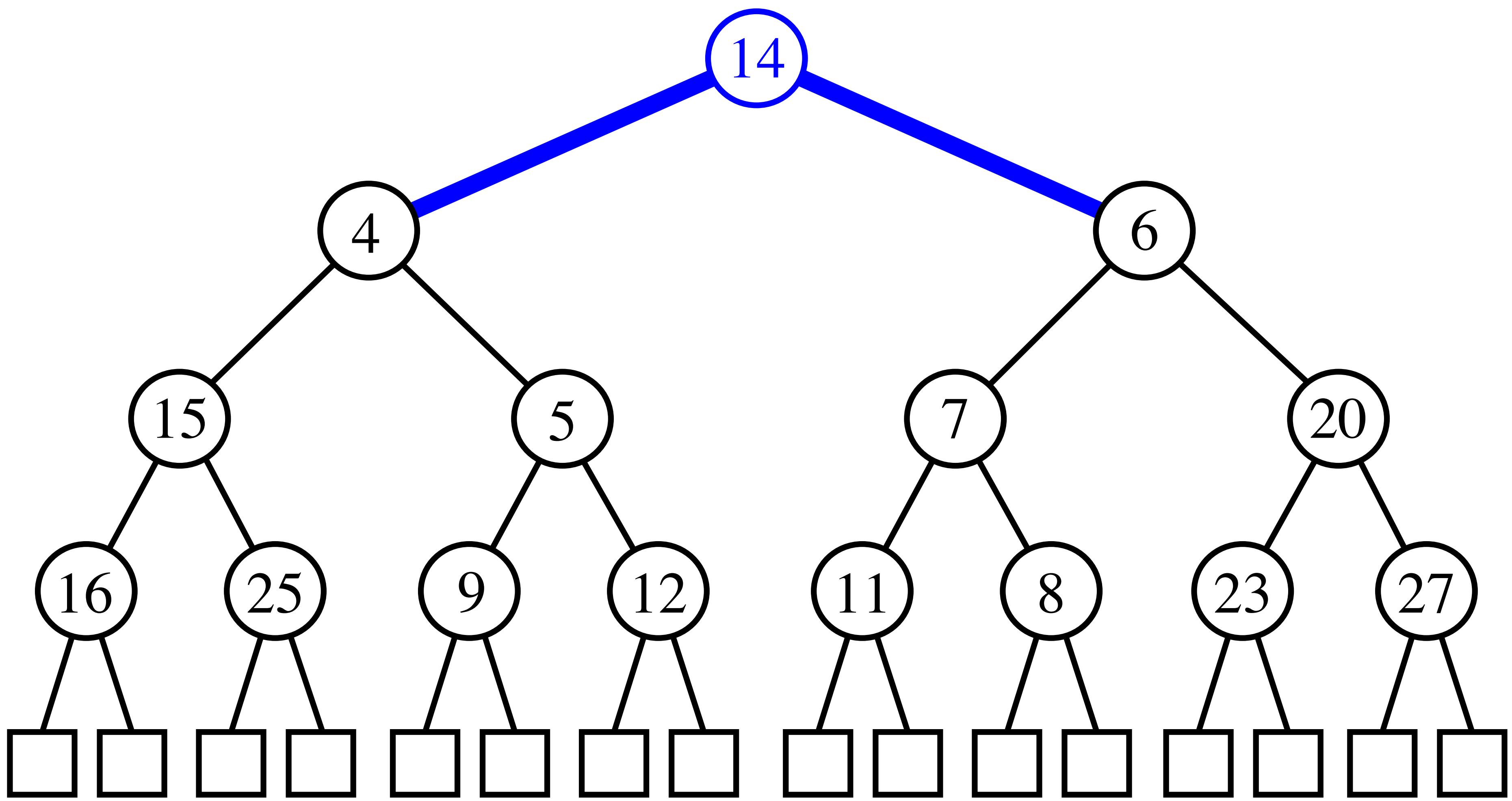
- now form seven-element heaps



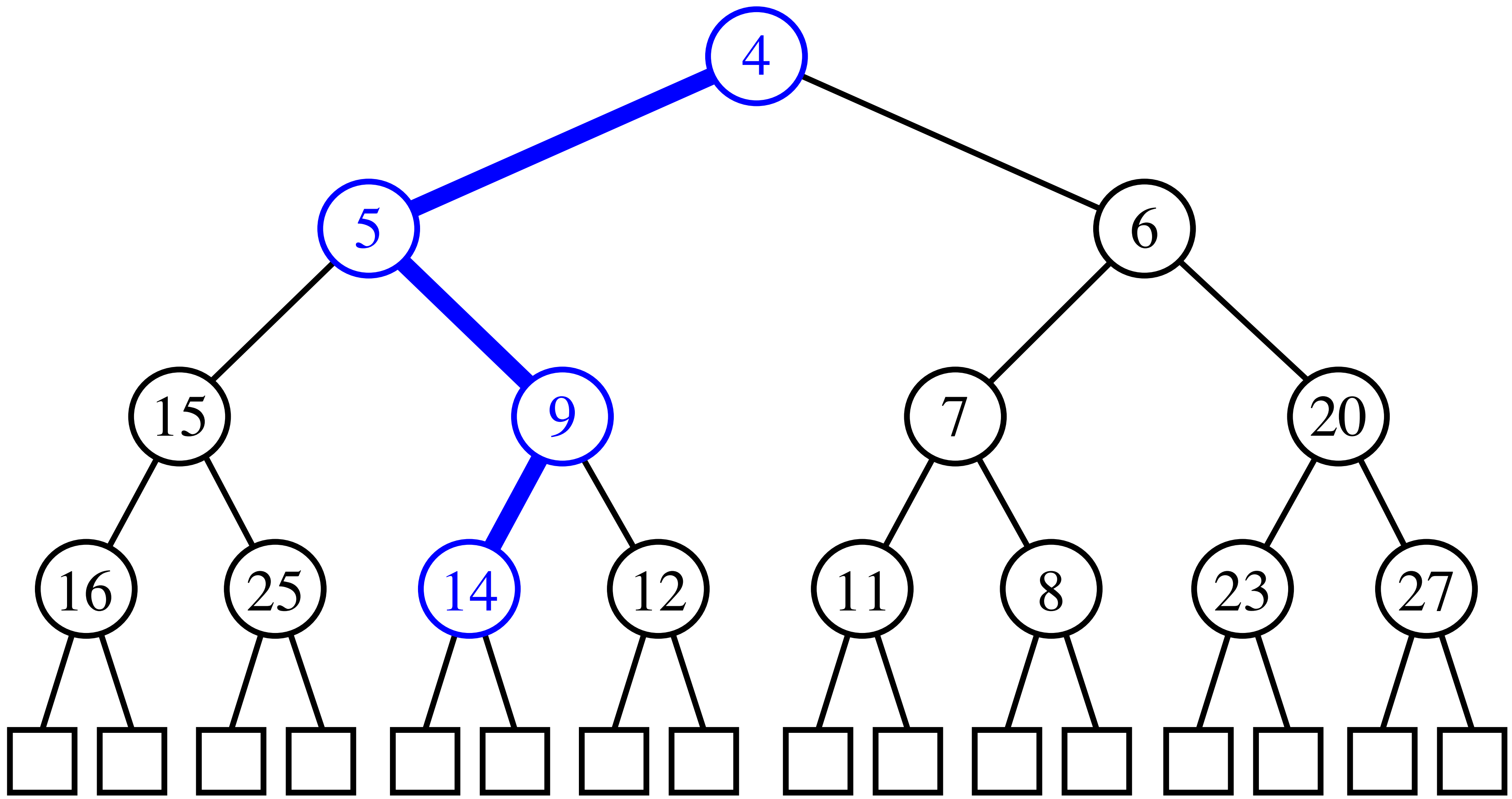
Bottom-Up Heap Construction (cont.)



Bottom-Up Heap Construction (cont.)



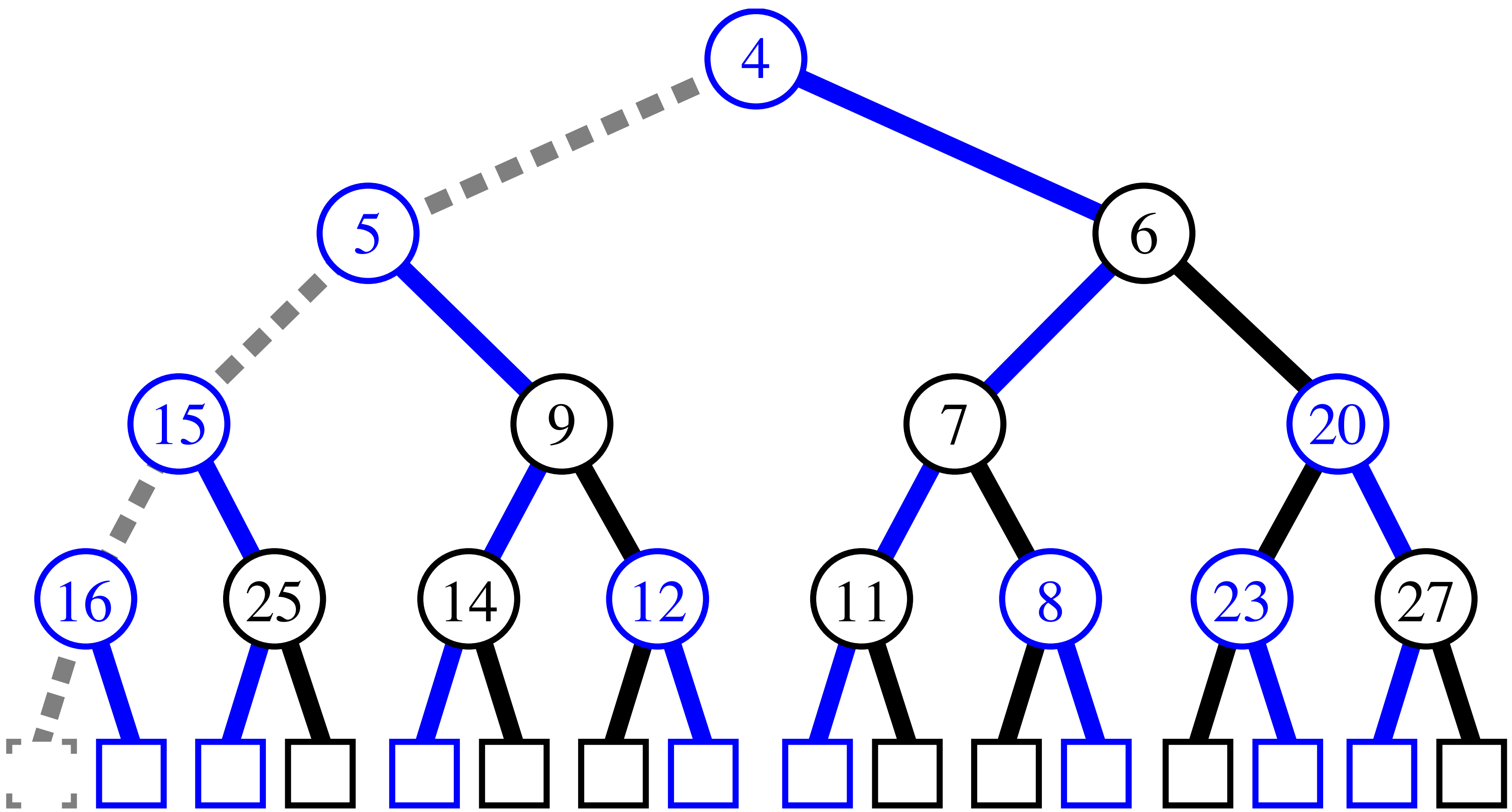
Bottom-Up Heap Construction (cont.)



The End

Analysis of Bottom-Up Heap Construction

- Proposition: Bottom-up heap construction with n keys takes $O(n)$ time.
  - Insert $(n + 1)/2$ nodes
  - Insert $(n + 1)/4$ nodes and downheap them
  - Insert $(n + 1)/8$ nodes and downheap them
  - ...



- n inserts, $n/2$ upheaps with total $O(n)$ running time

Chapter 8

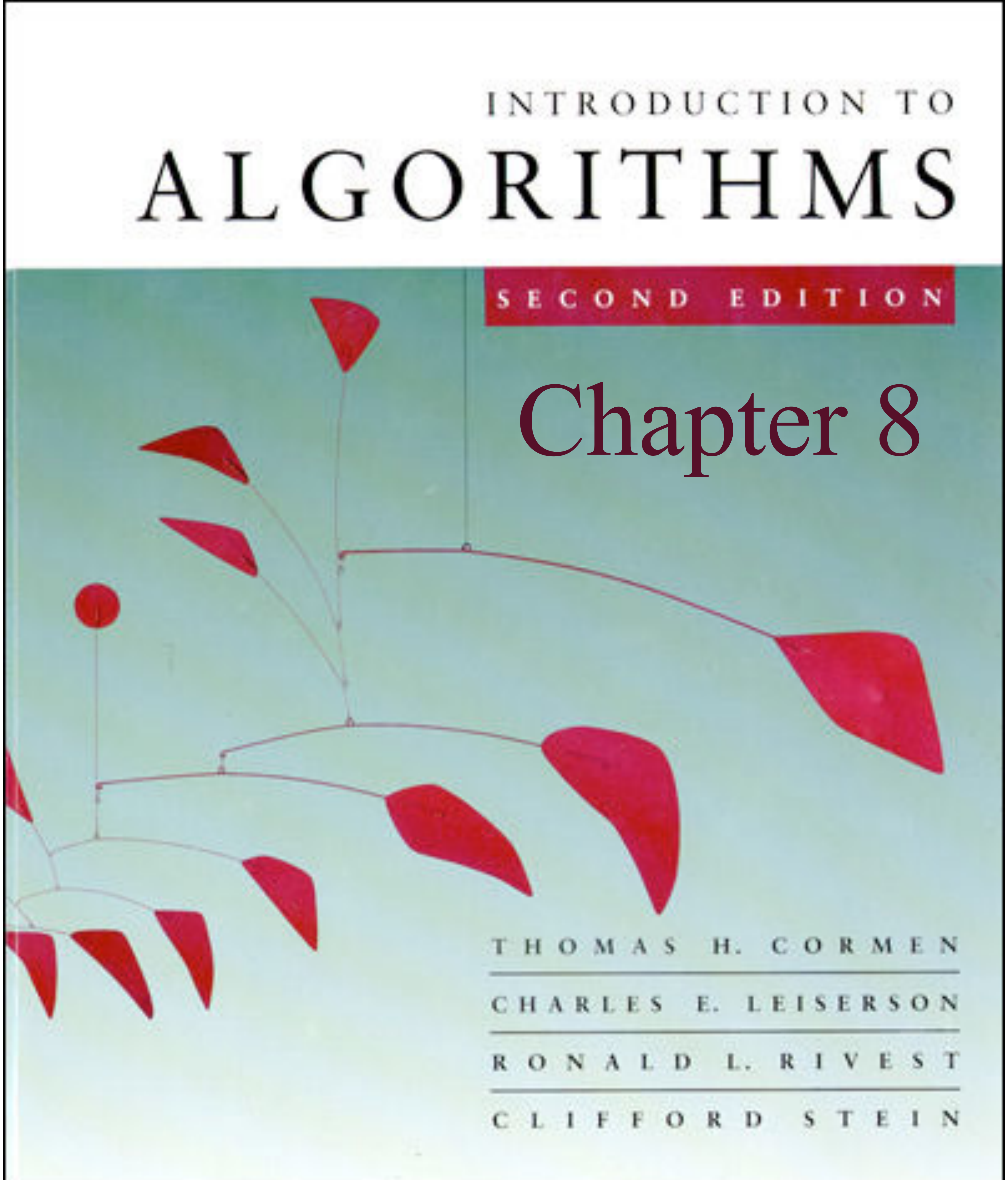
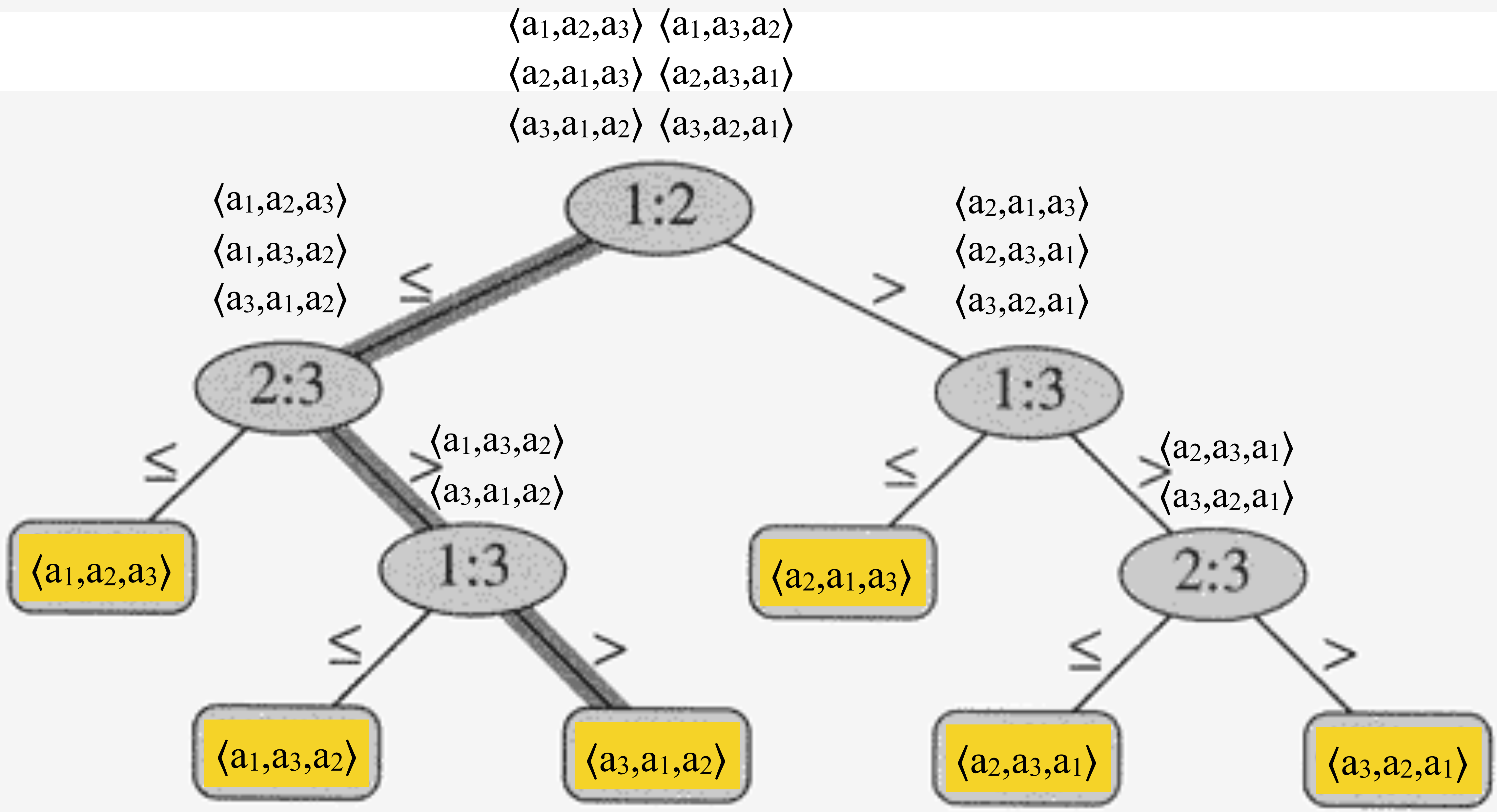
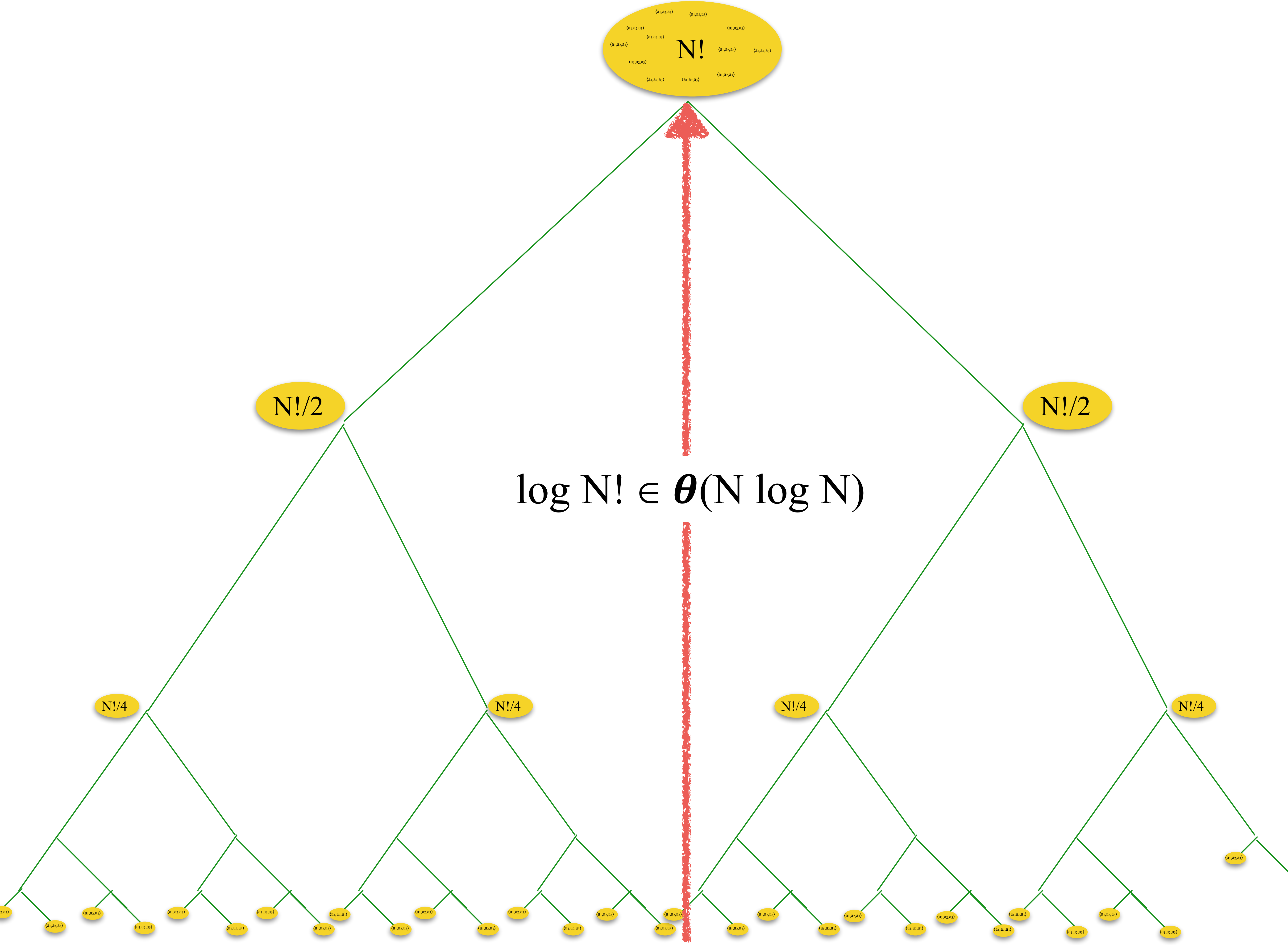

INTRODUCTION TO
ALGORITHMS
SECOND EDITION
THOMAS H. CORMEN
CHARLES E. LEISERSON
RONALD L. RIVEST
CLIFFORD STEIN

Figure 8.1 The decision tree for insertion sort operating on three elements. An internal node annotated by $i{:}j$ indicates a comparison between a_i and a_j. A leaf annotated by the permutation $\langle \pi(1), \pi(2), \ldots, \pi(n) \rangle$ indicates the ordering $a_{\pi(1)} \leq a_{\pi(2)} \leq \cdots \leq a_{\pi(n)}$. The shaded path indicates the decisions made when sorting the input sequence $\langle a_1 = 6, a_2 = 8, a_3 = 5 \rangle$; the permutation $\langle 3, 1, 2 \rangle$ at the leaf indicates that the sorted ordering is $a_3 = 5 \leq a_1 = 6 \leq a_2 = 8$. There are $3! = 6$ possible permutations of the input elements, so the decision tree must have at least 6 leaves.




N!
N!/2
N!/2
log N! ∈ θ(N log N)
N!/4
N!/4
N!/4
N!/4

Winter 2016
COMP-250: Introduction to Computer Science

Lecture 20, March 24, 2016

Hardik

Omar

Faiz

Lekan

Faizy

Chris

David B.

David B.R.

DoYeon