Priority Queues (Heaps)

Priority Queue: Motivating Example

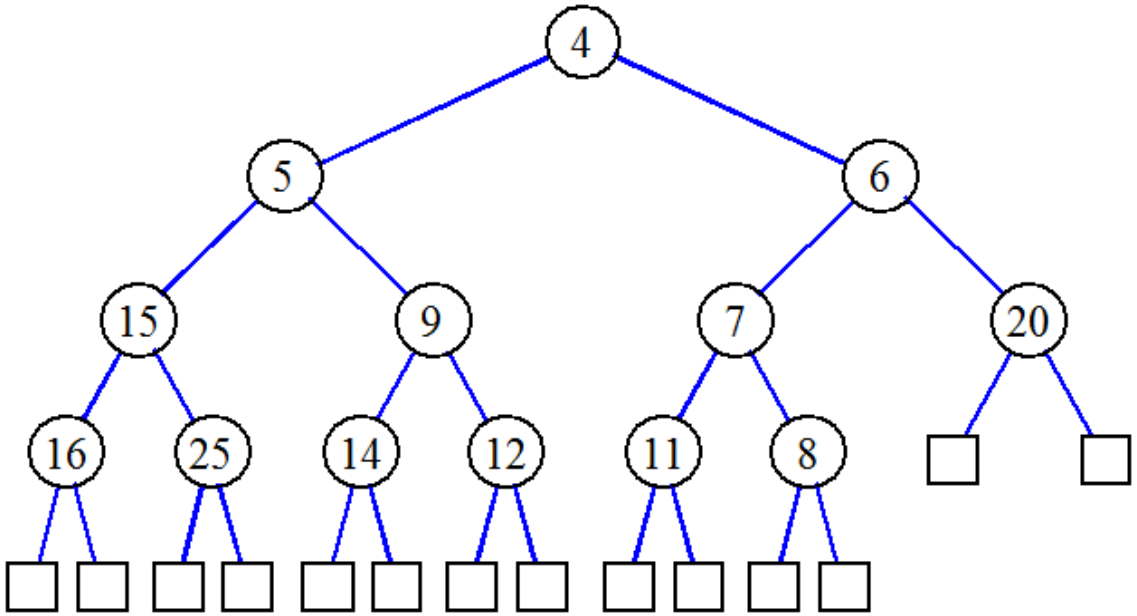
- 3 jobs have been submitted to a printer in the order A, B, C.
  - Job A -100 pages
  - Job B - 10 pages
  - Job C - 1 page

- Average waiting time with FIFO service:
  (100+110+111) / 3 = 107 time units
- Average waiting time for shortest-job-first service:
  (1+11+111) / 3 = 41 time units
- A queue be capable to insert and deletemin?

Priority Queue

Heaps

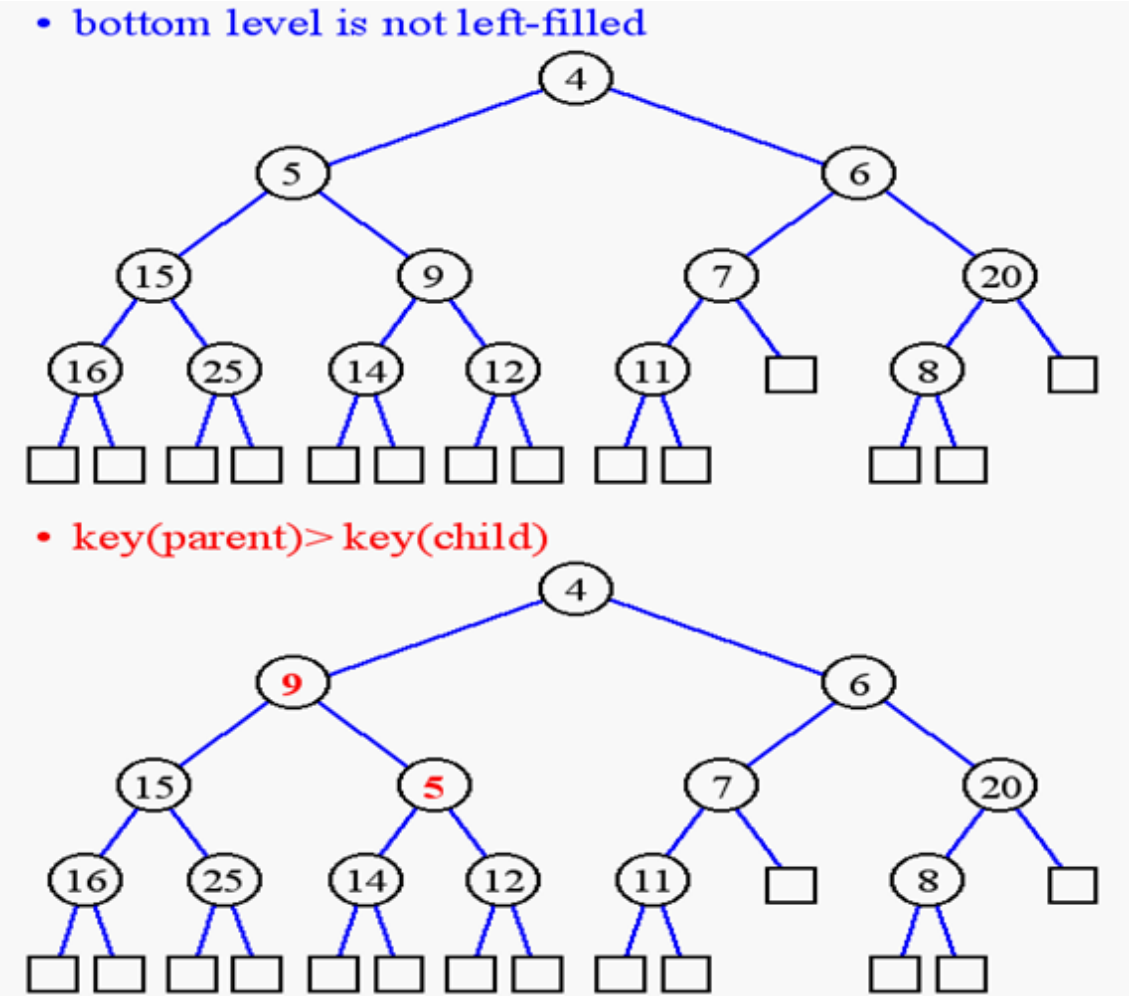
- A *heap* is a binary tree T that stores a key-element pairs at its internal nodes
- It satisfies two properties:
  - MinHeap: key(parent) $\leq$ key(child)
    OR MaxHeap: key(parent) $\geq$ key(child)
  - all levels are full, except the last one, which is left-filled, i.e., a complete binary tree.



What are Heaps Useful for?

- To implement priority queues
- Priority queue = a queue where all elements have a ”priority” associated with them
- Remove in a priority queue removes the element with the smallest priority
  - insert
  - removeMin

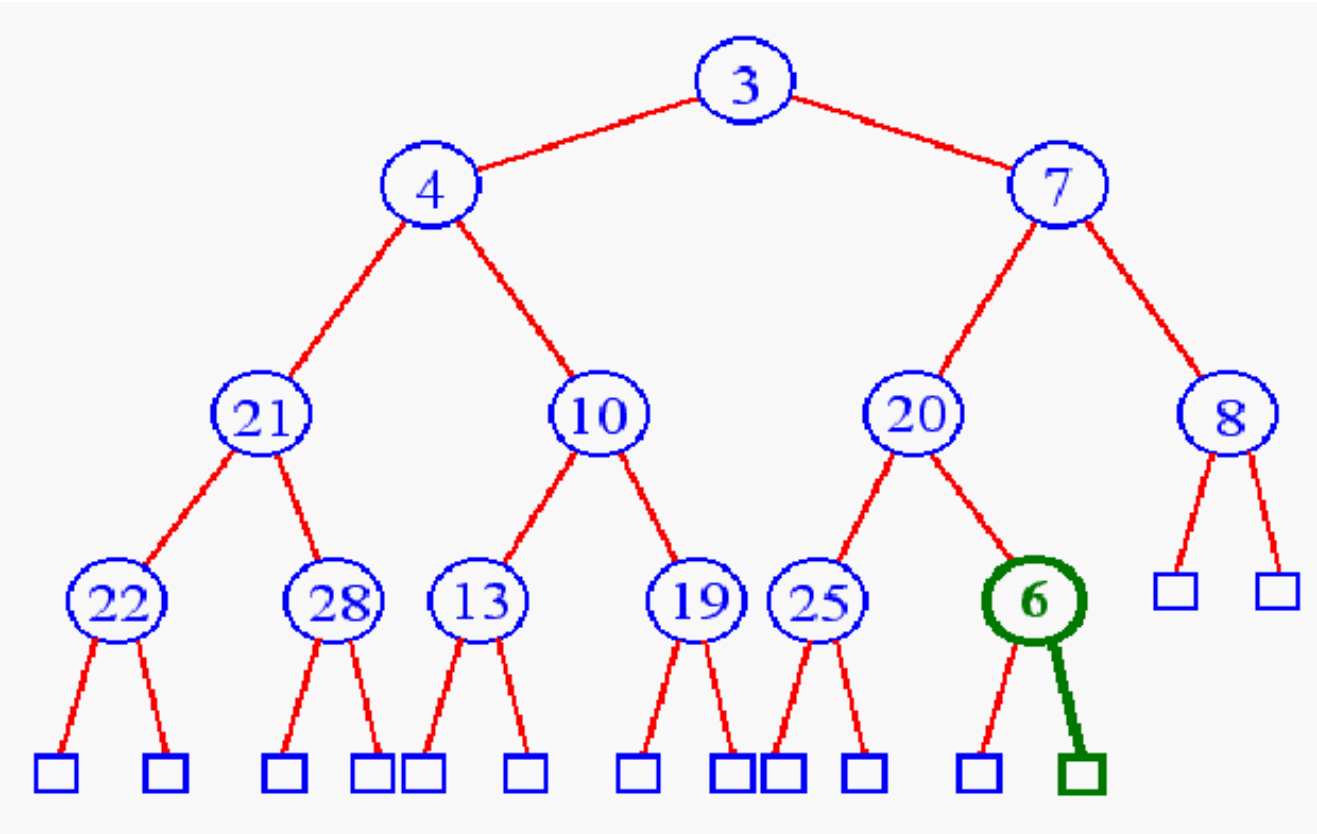
Heap or Not a Heap?



- A heap T storing n keys has height $h = \lfloor log(n+1) \rfloor$, which is $O(logn)$

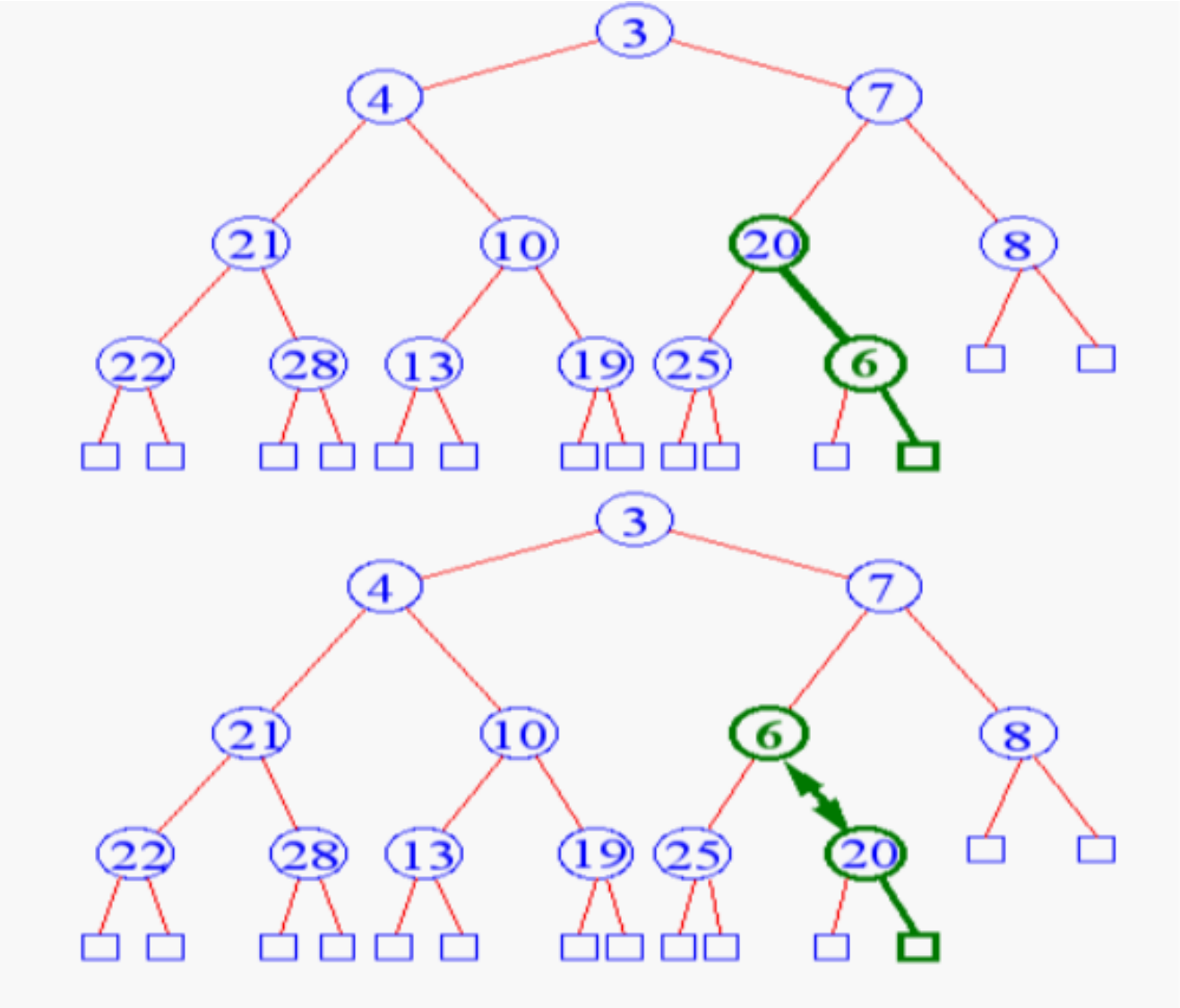
Heap Insertion

- Insert 6 – Add key in next available position

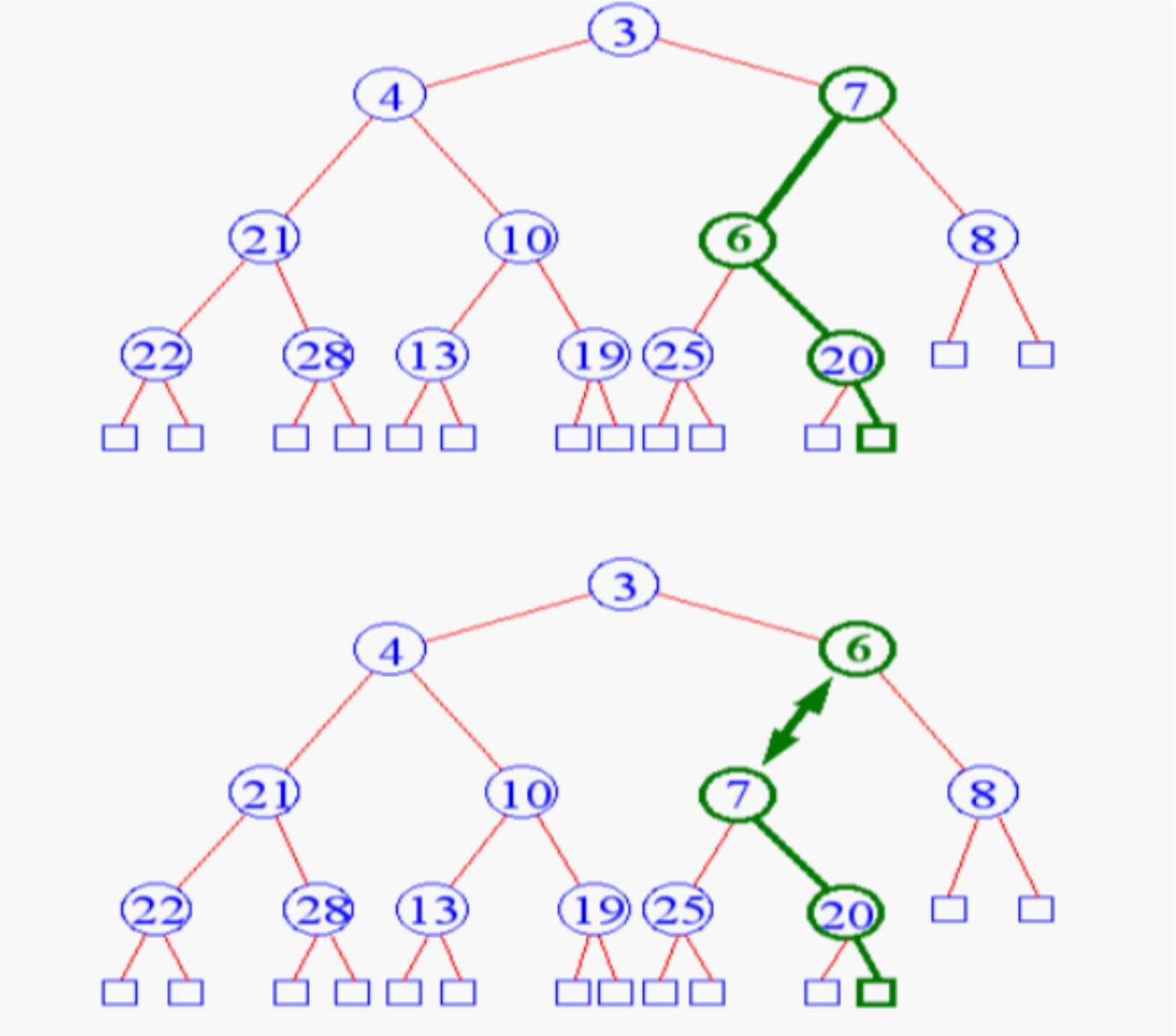


Heap Insertion

- Begin Unheap

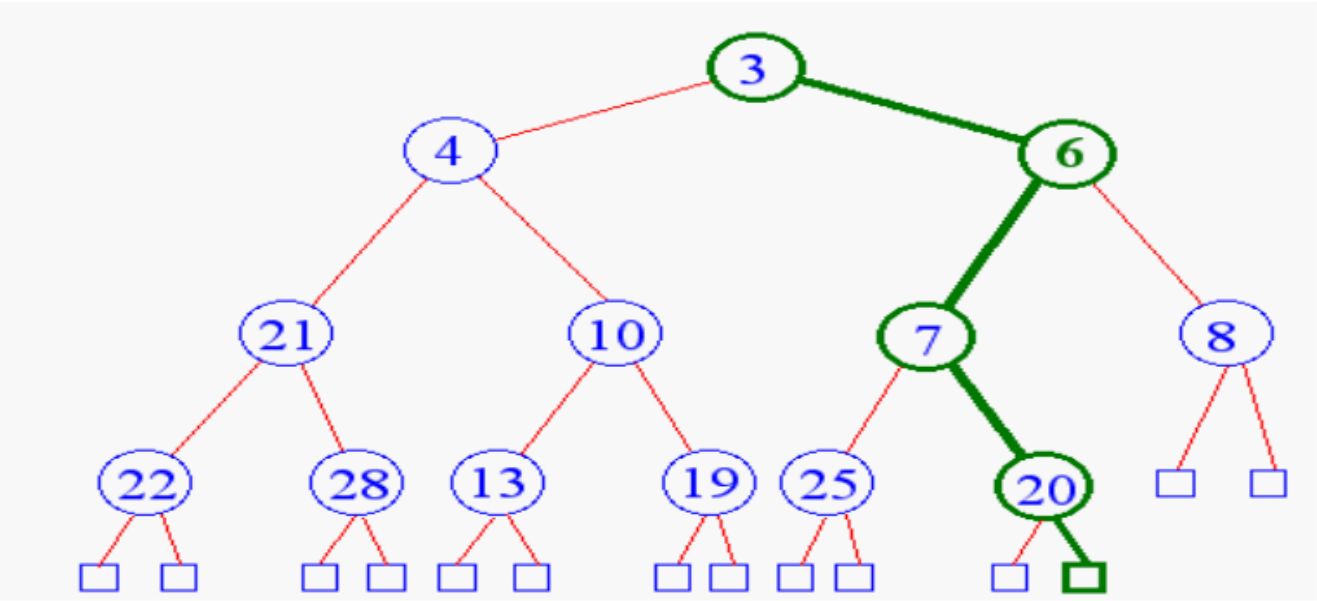


Heap Insertion



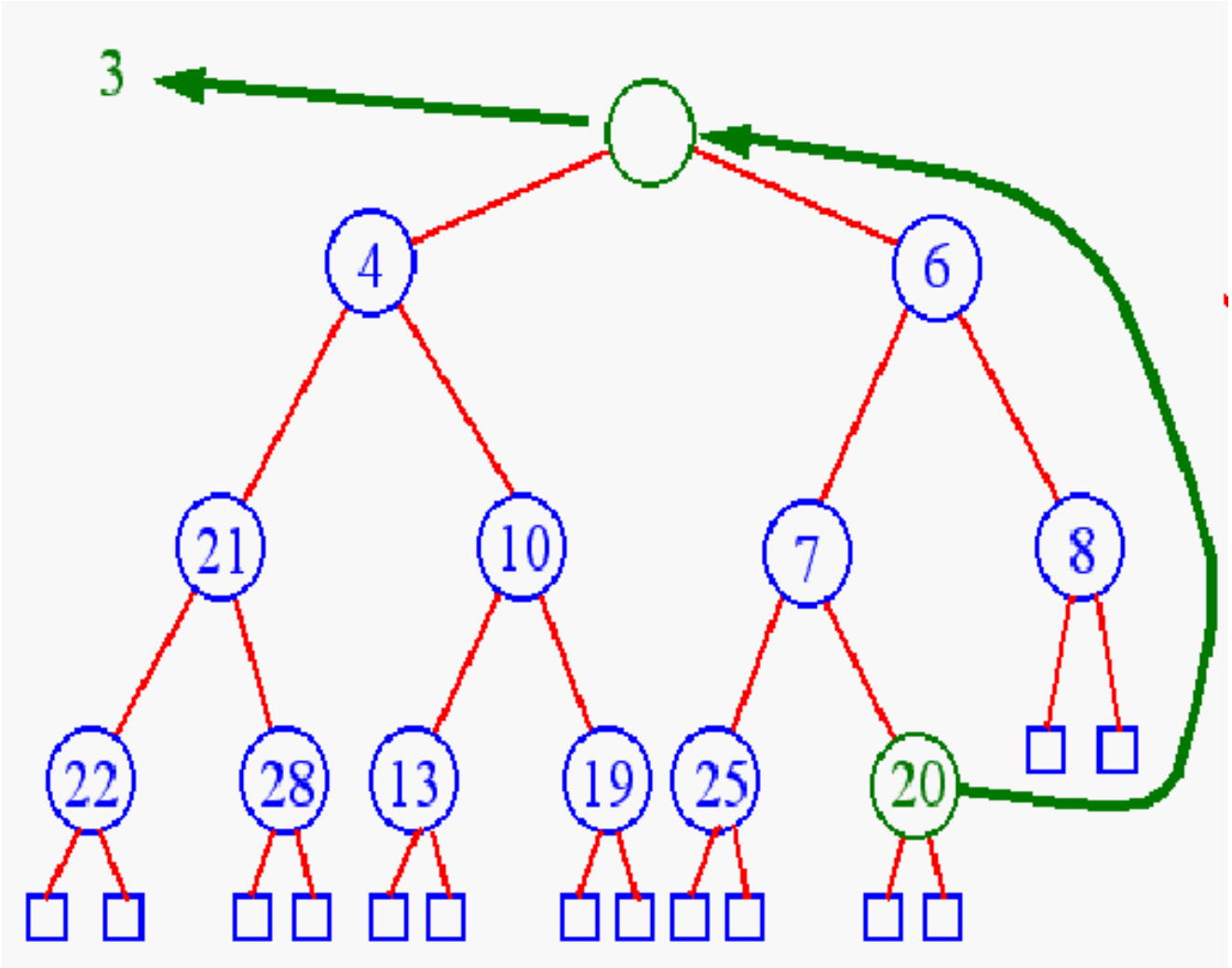
Heap Insertion

- Terminate unheap when
  - reach root
  - key child is greater than key parent



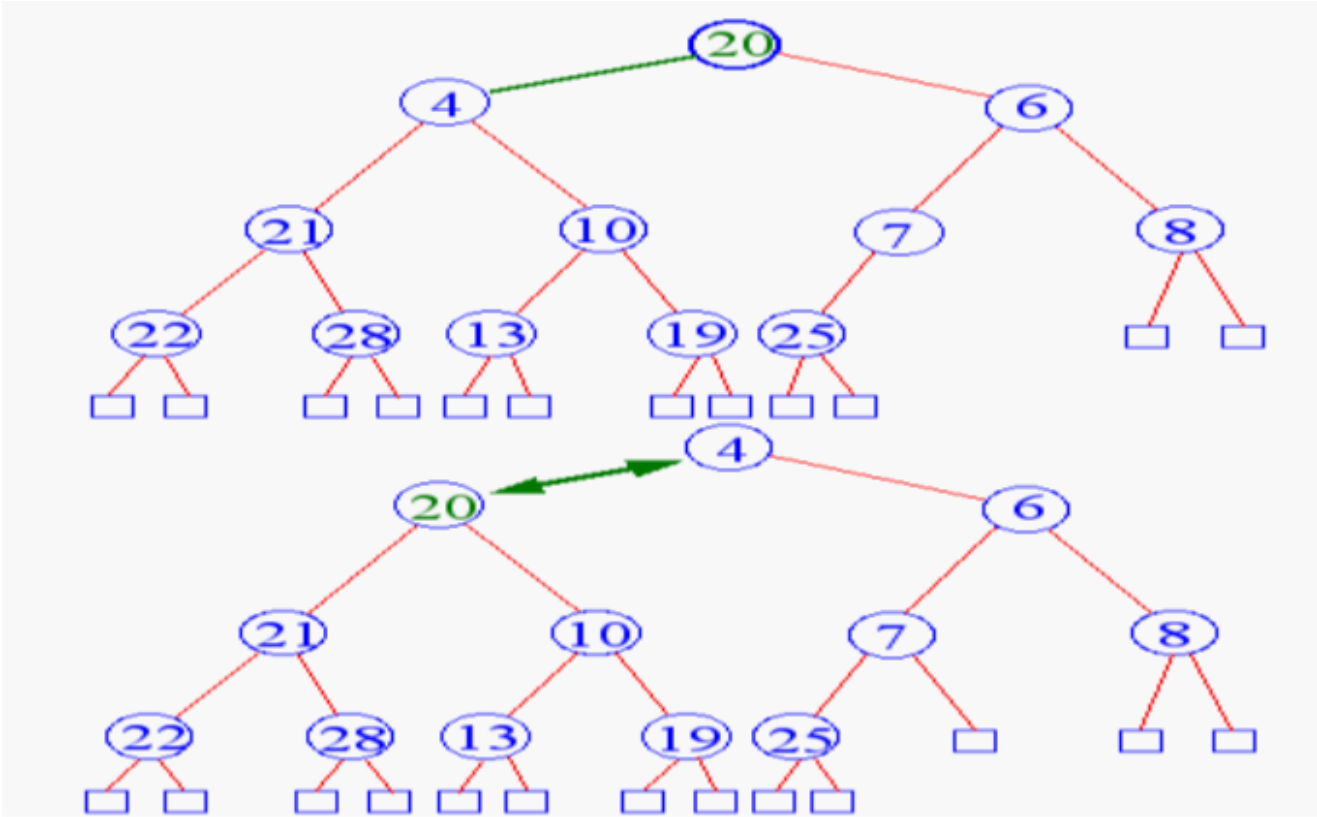
Heap Removal

- Remove element from priority queues? removeMin( )

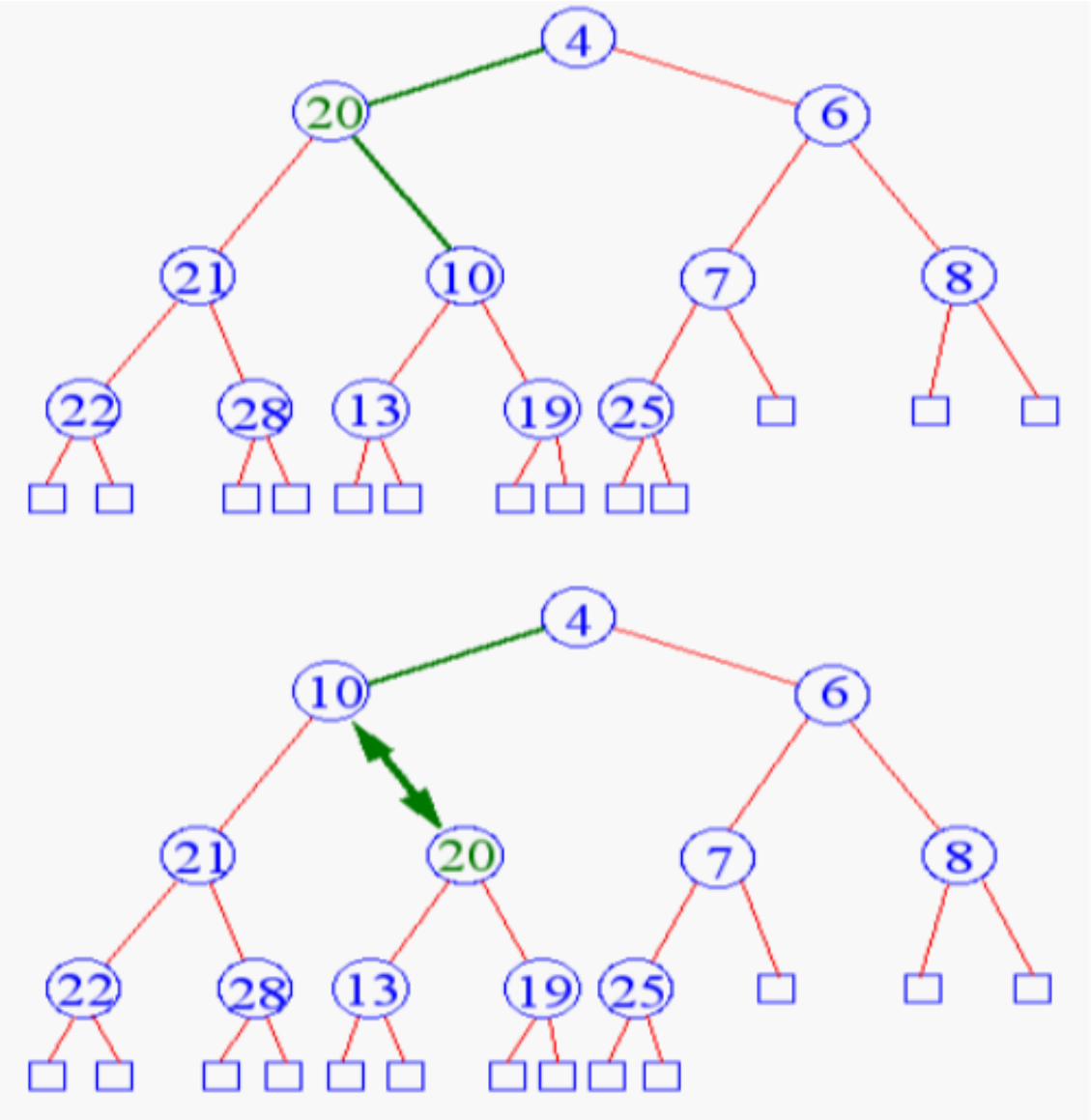


Heap Removal

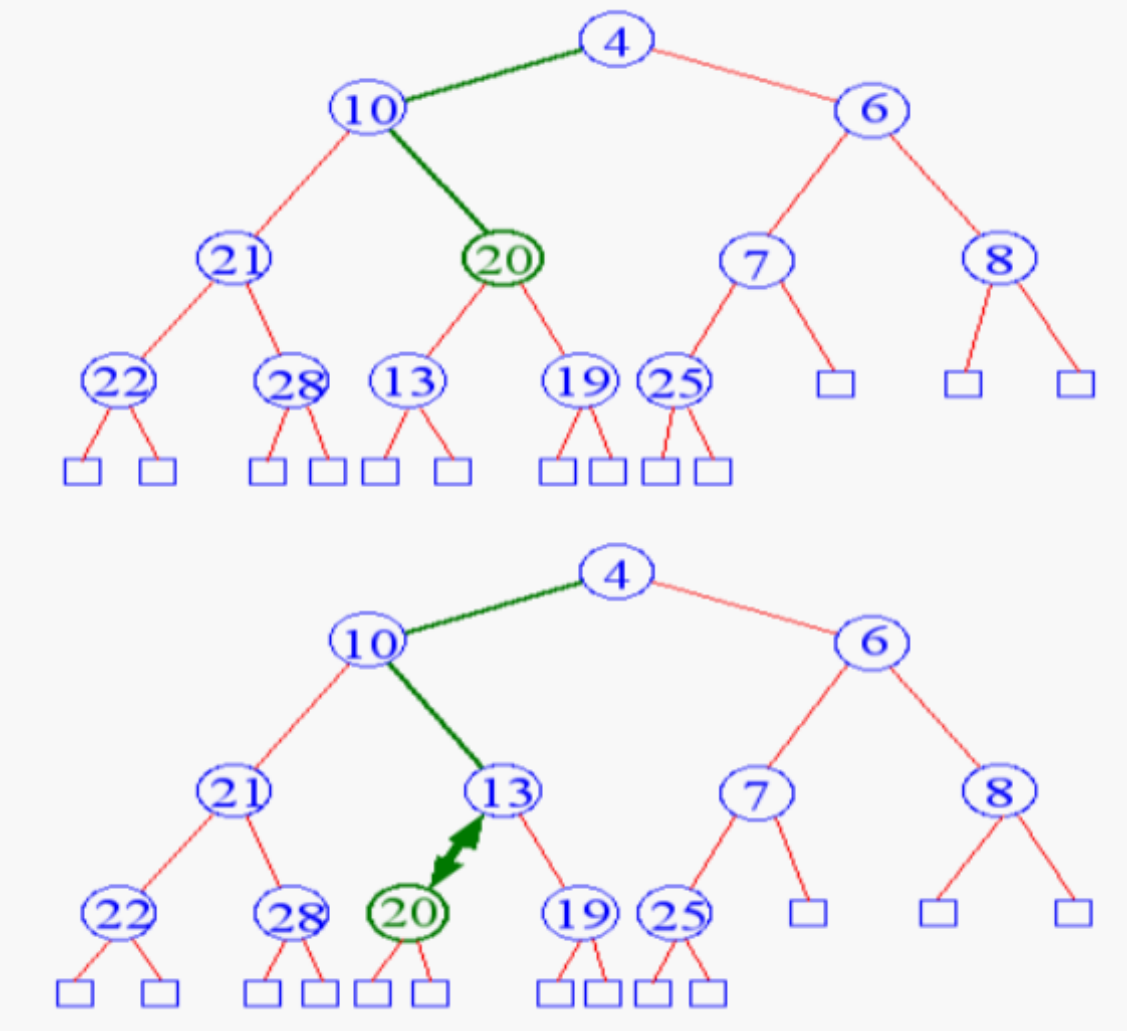
- Begin downheap



Heap Removal

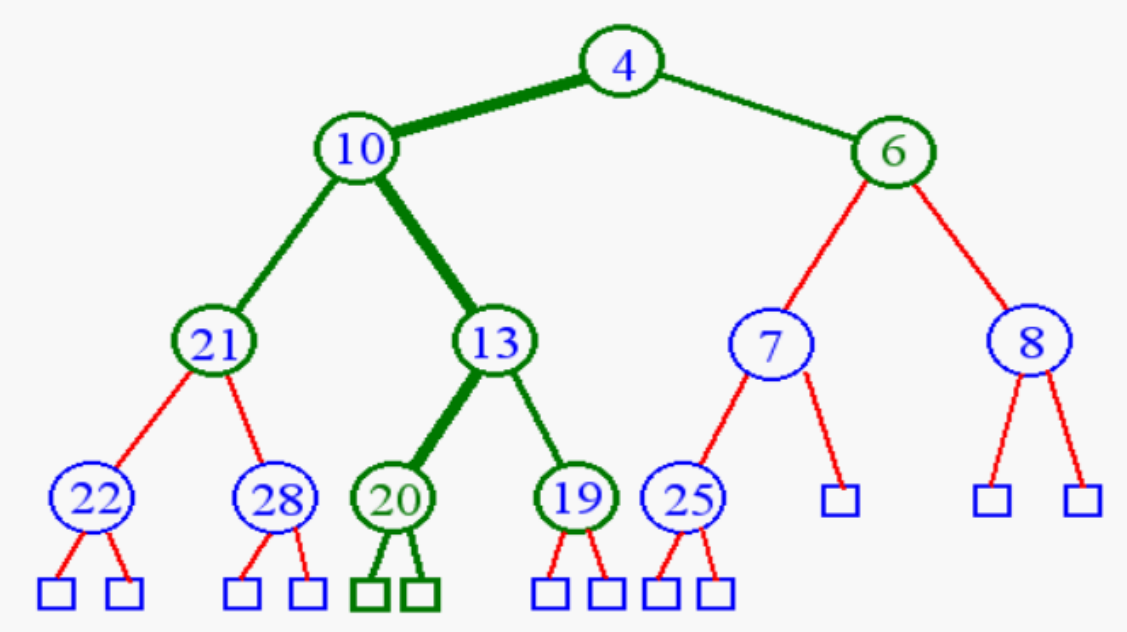


Heap Removal



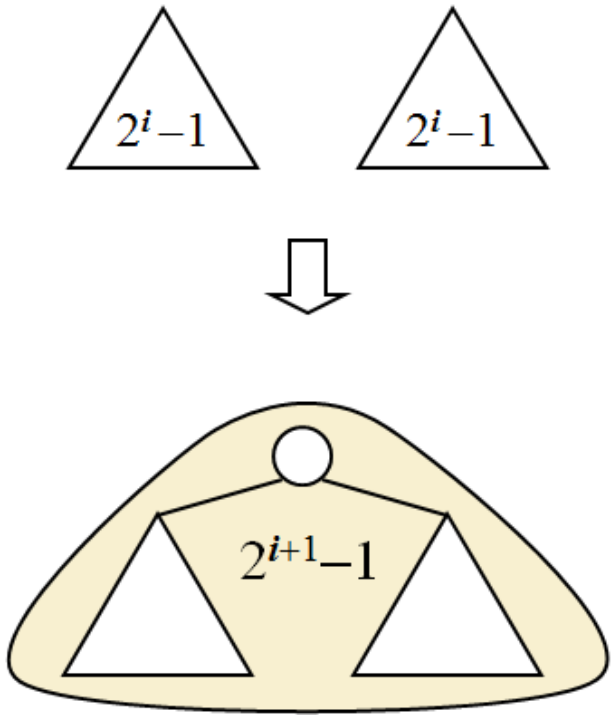
Heap Removal

- Terminate downheap when
  - reach leaf level
  - key child is greater than key parent



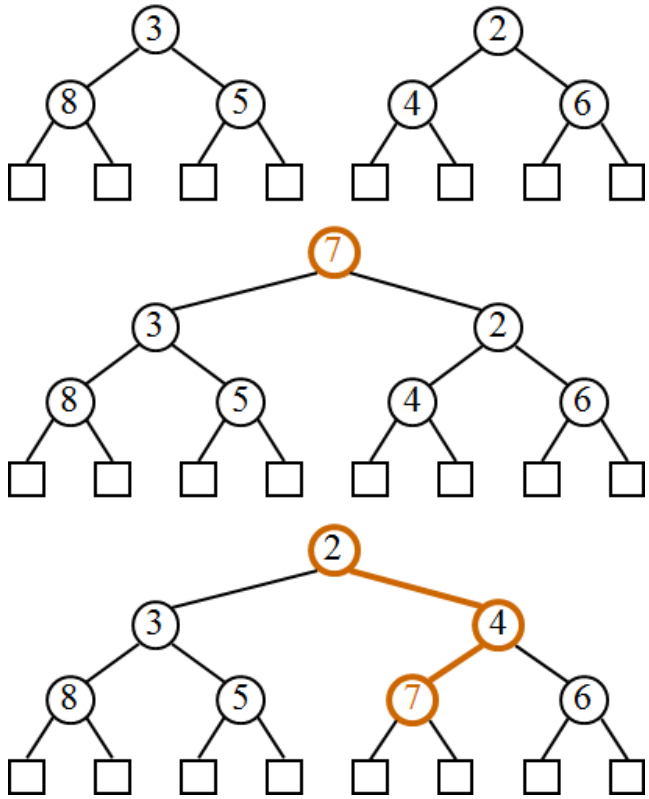
Bottom-up Heap Construction

- We can construct a heap storing n given keys using a bottom-up construction with $logn$ phases
- In phase i, pairs of heaps with $2^i - 1$ keys are merged into heaps with $2^{i+1} - 1$ keys

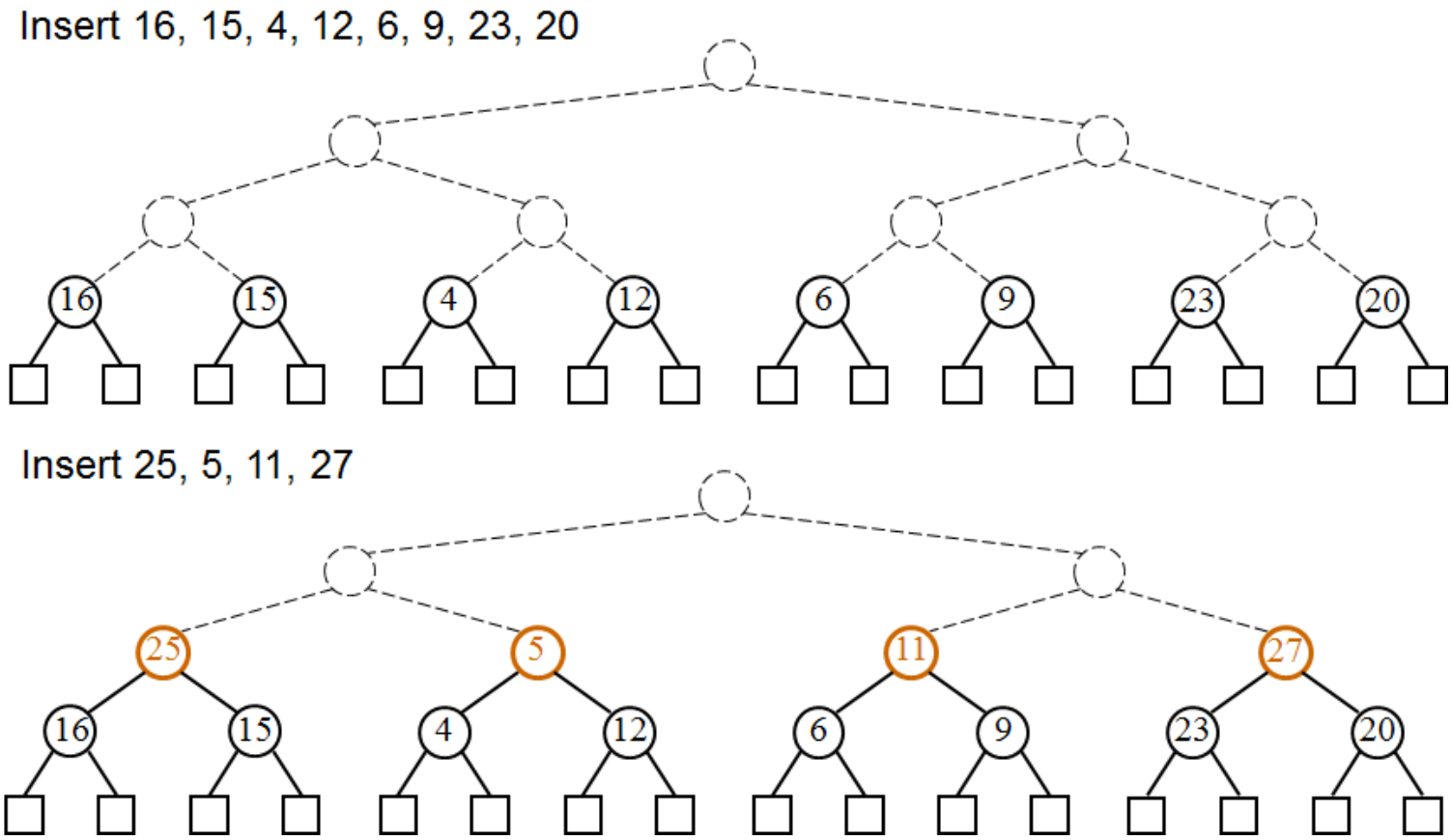


Merging Two Heaps

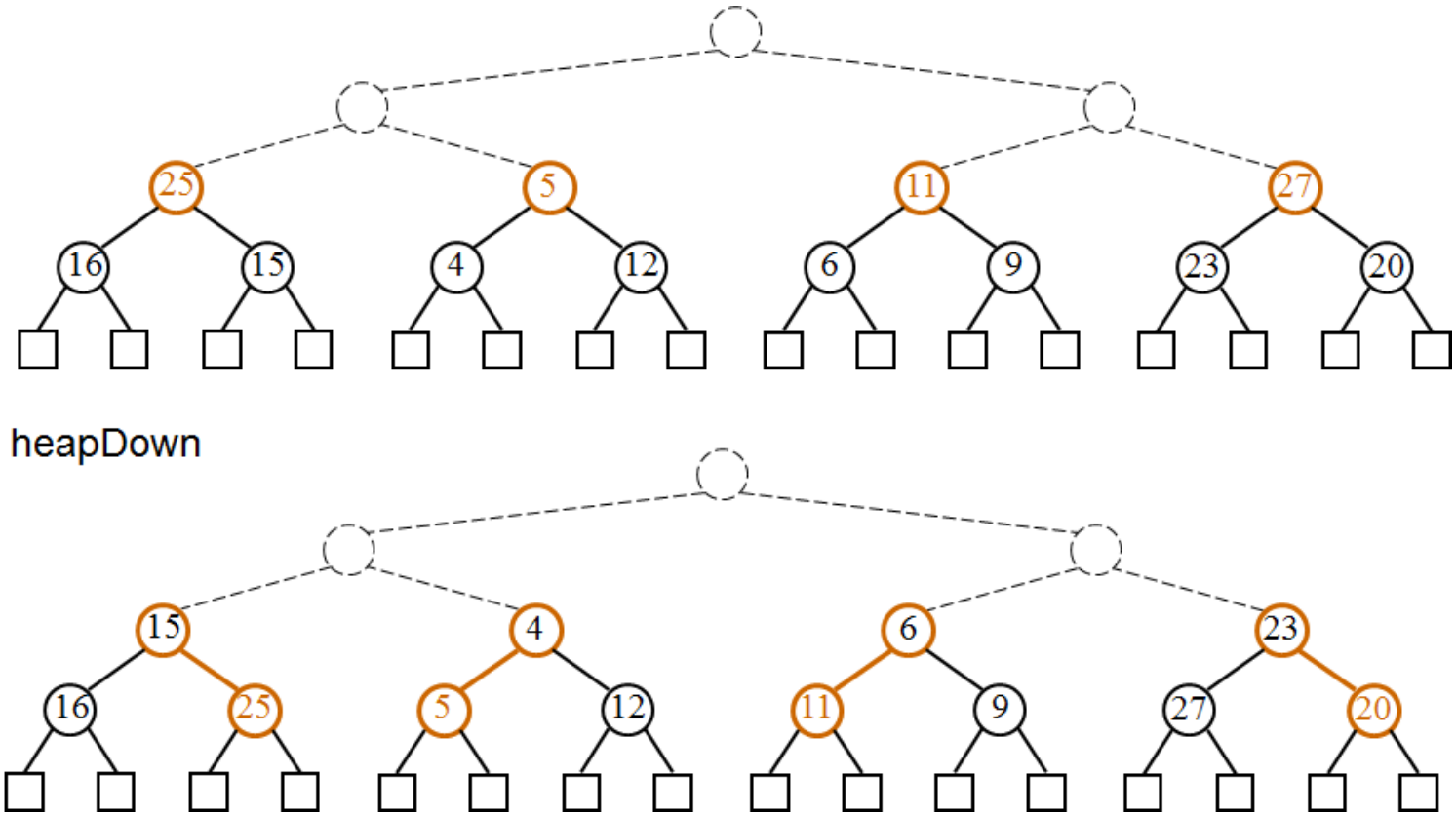
- We are given two two heaps and a key k
- We create a new heap with the root node storing k and with the two heaps as subtrees
- We perform heapDown to restore the heap-order property



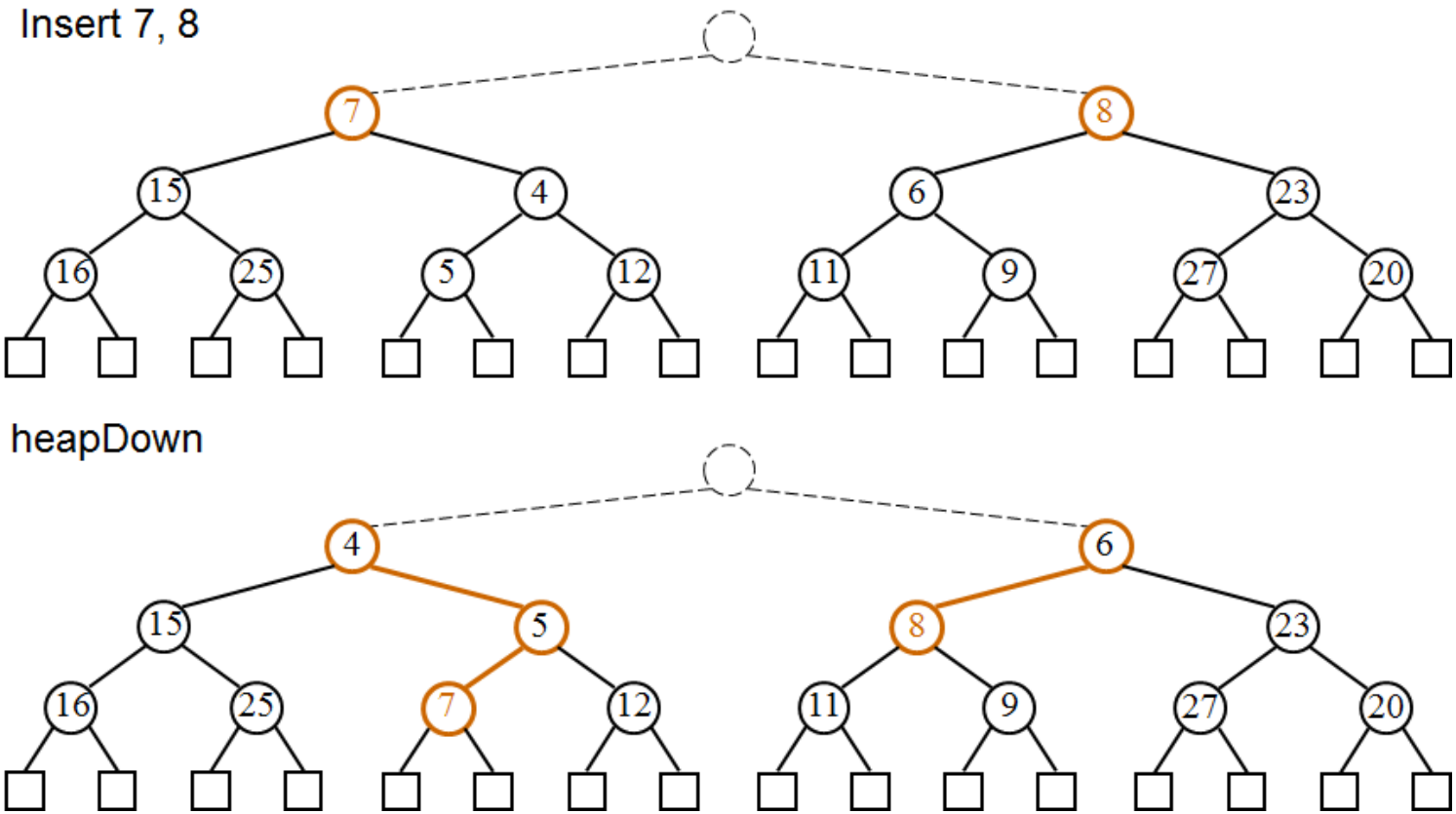
Merging Example



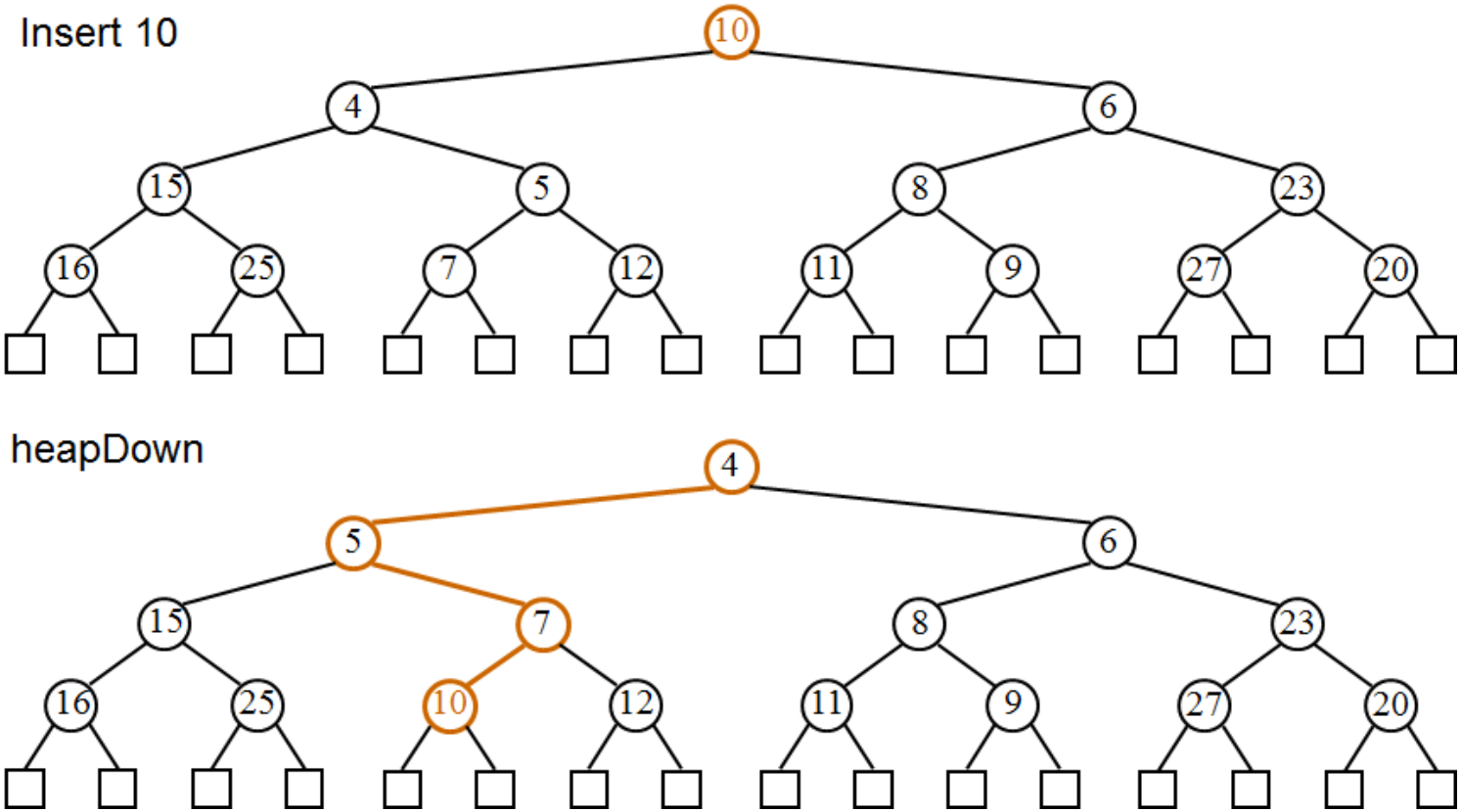
Example (contd.)



Example (contd.)

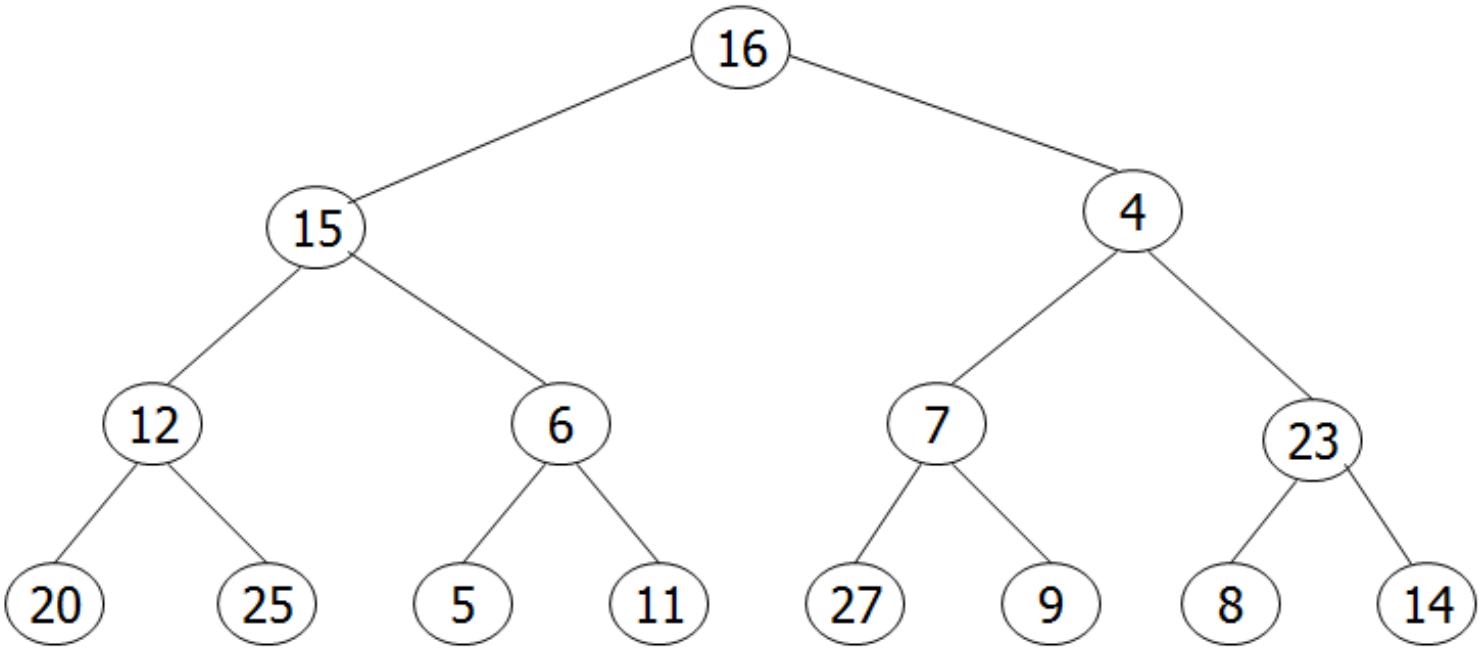


Example (contd.)

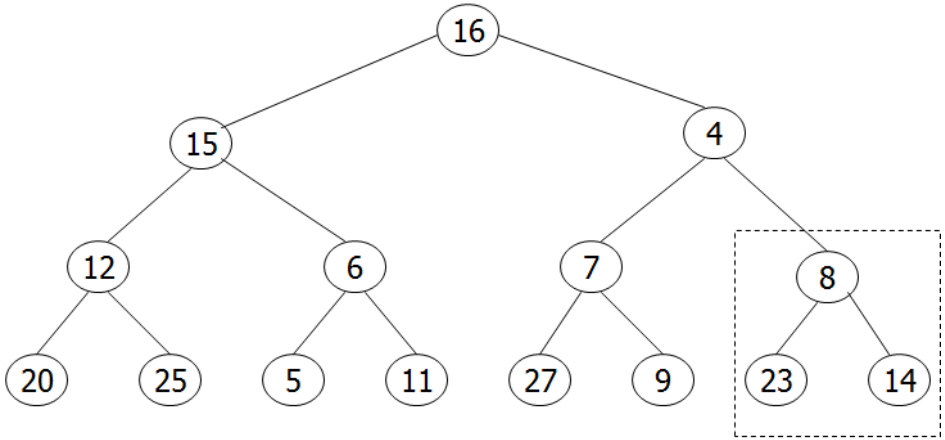


Bottom up Heap Construction

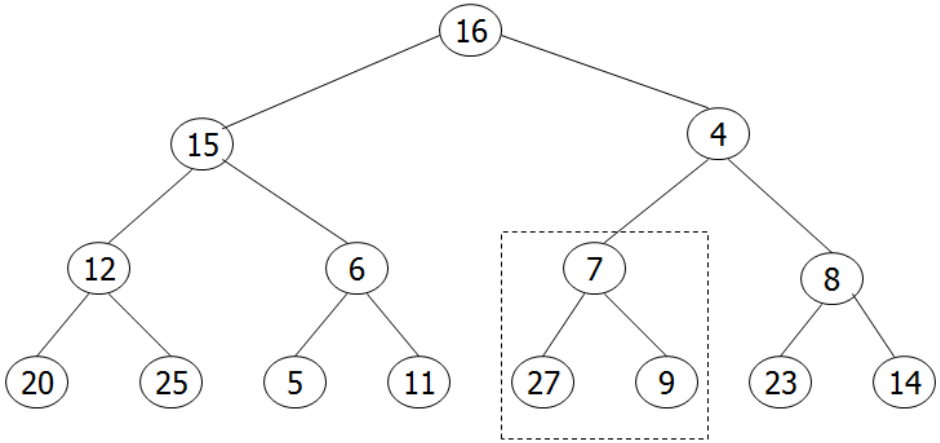
16	15	4	12	6	7	23	20	25	5	11	27	9	8	14



Bottom up Heap Construction

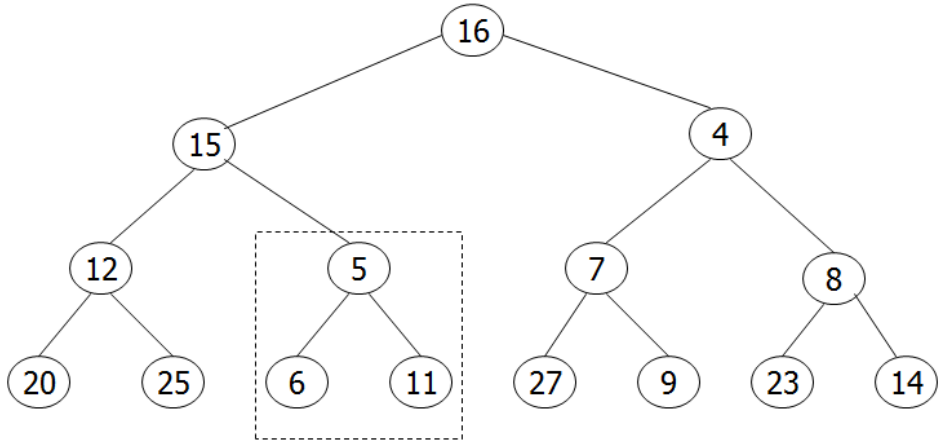


16	15	4	12	6	7	8	20	25	5	11	27	9	23	14

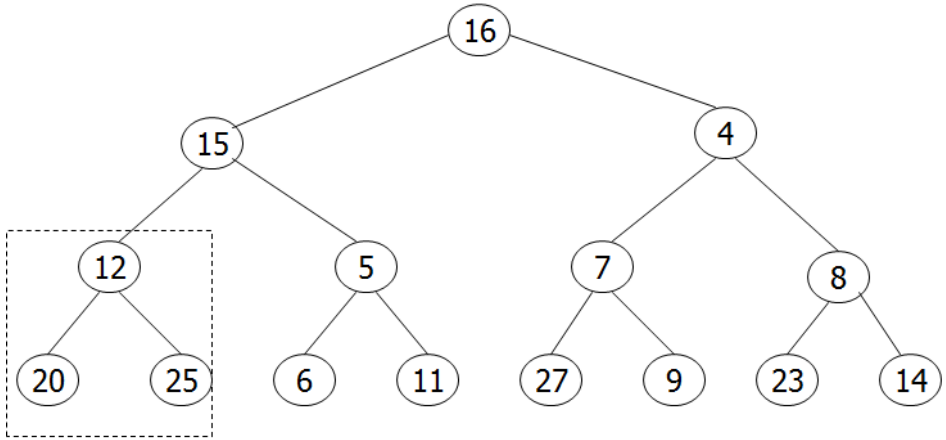


16	15	4	12	6	7	8	20	25	5	11	27	9	23	14

Bottom up Heap Construction

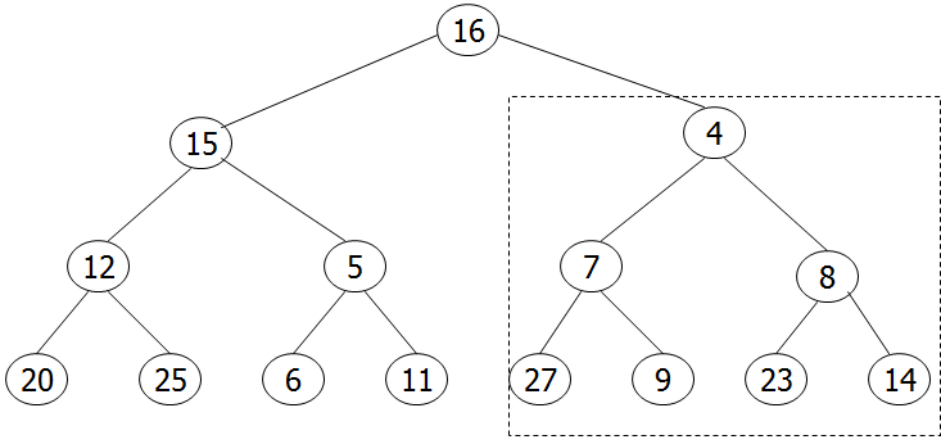


16	15	4	12	5	7	8	20	25	6	11	27	9	23	14

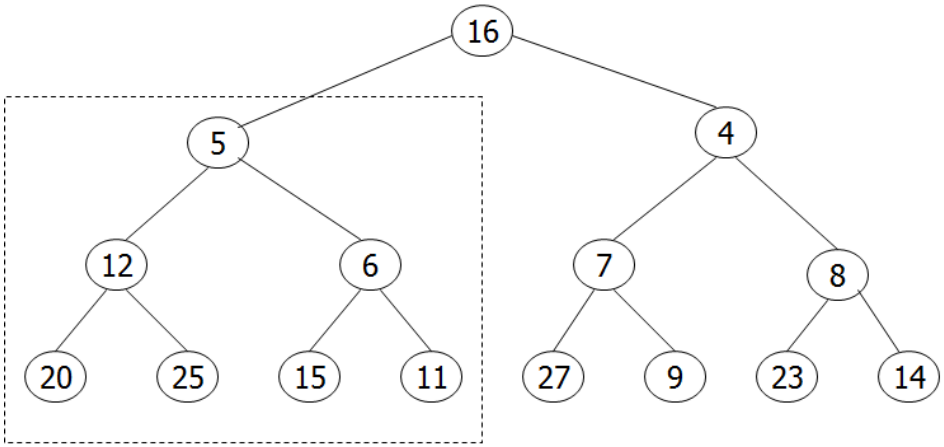


16	15	4	12	5	7	8	20	25	6	11	27	9	23	14

Bottom up Heap Construction

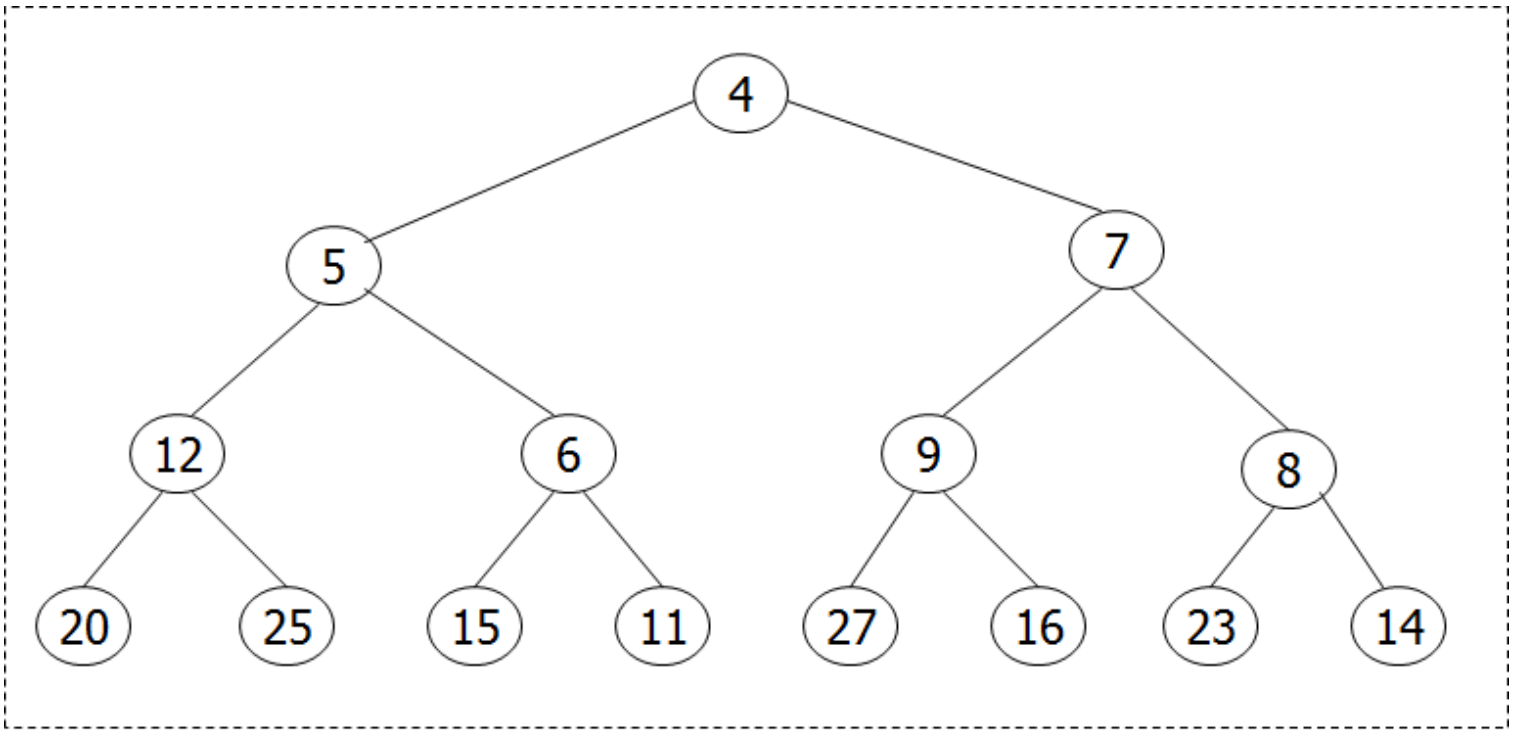


16	15	4	12	5	7	8	20	25	6	11	27	9	23	14



16	5	4	12	6	7	8	20	25	15	11	27	9	23	14

Bottom up Heap Construction



16	5	4	12	6	7	8	20	25	15	11	27	9	23	14

Heap Sorting

- Step 1: Build a heap
- Step 2: removeMin( )
- Running time?