UNIVERSITY of NORTH TEXAS

CSCE 3110
Data Structures
and Algorithm Analysis

Rada Mihalcea

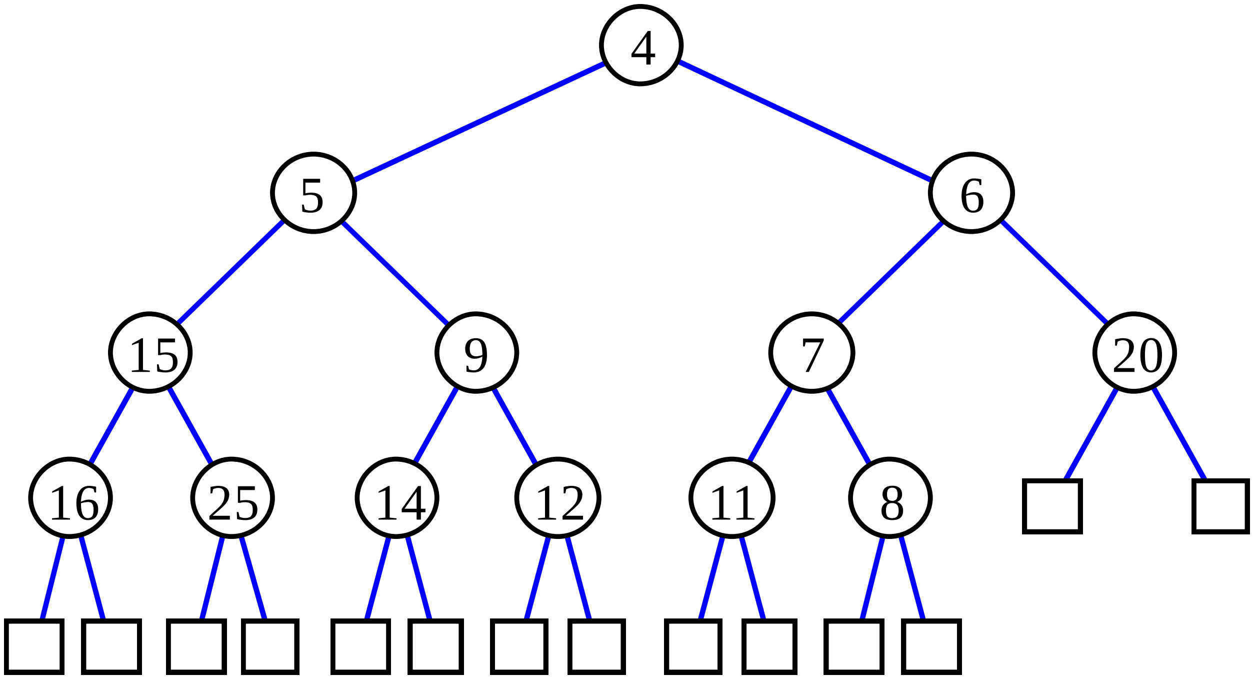
http://www.cs.unt.edu/~rada/CSCE3110

Heaps

Heaps

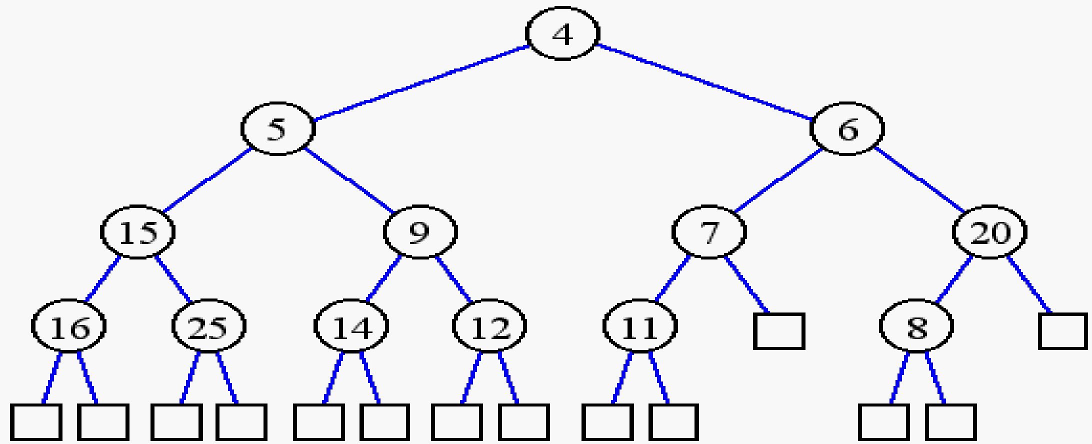
- A **<u>heap</u>** is a **binary** tree T that stores a key-element pairs at its nodes
- It satisfies two properties:
  - **MinHeap**: key(parent) <= key(child)
  - [OR MaxHeap: key(parent) >= key(child)]
  - all **levels** are **full**, except the **last** one, which is left-filled
- This way, it is **almost complete** tree, except right part of last level
- Complete binary tree is implemented using an array

Heaps

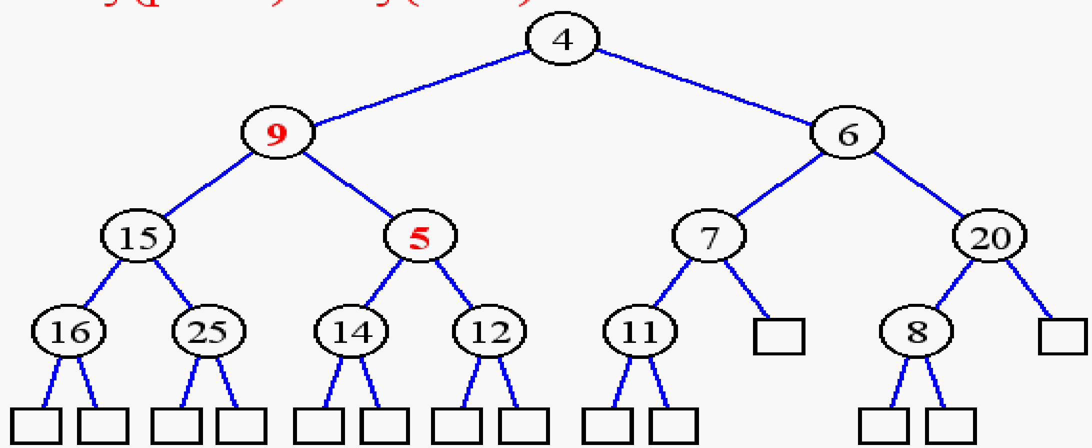


Not a Min Heap

- bottom level is not left-filled

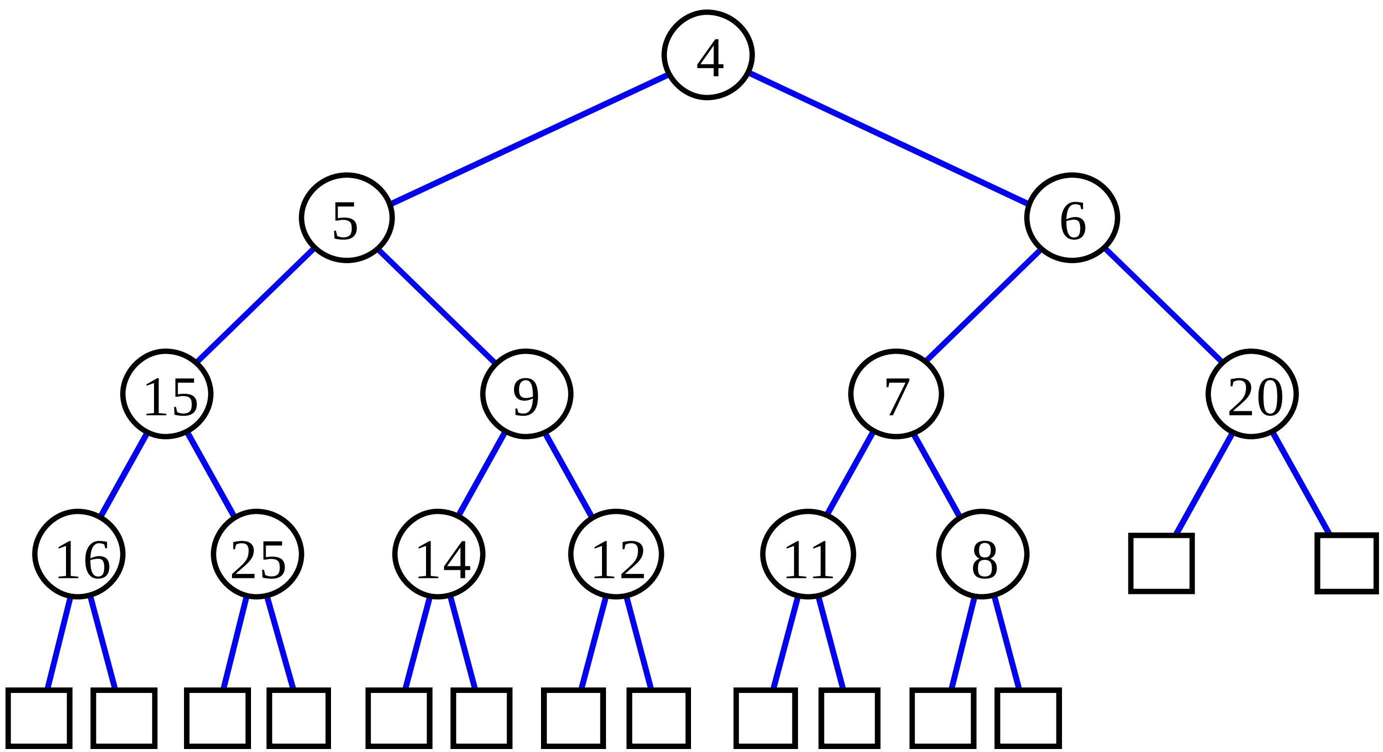


- key(parent)>key(child)



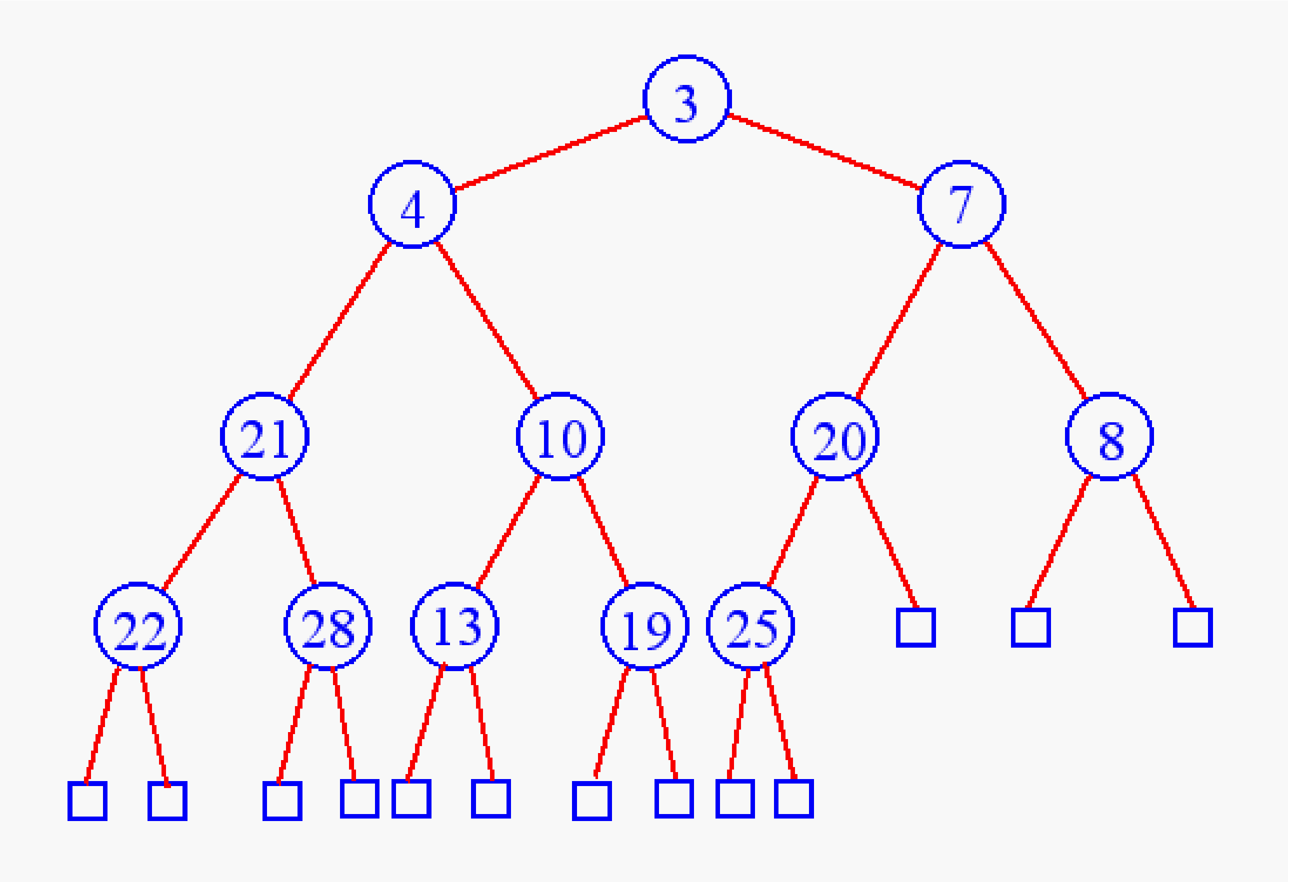
Heap Properties

- As heap is complete => height h = $\lfloor \log(n + 1) \rfloor$
- So insert/remove is log)
- Search is O(n) => not good to use it for that



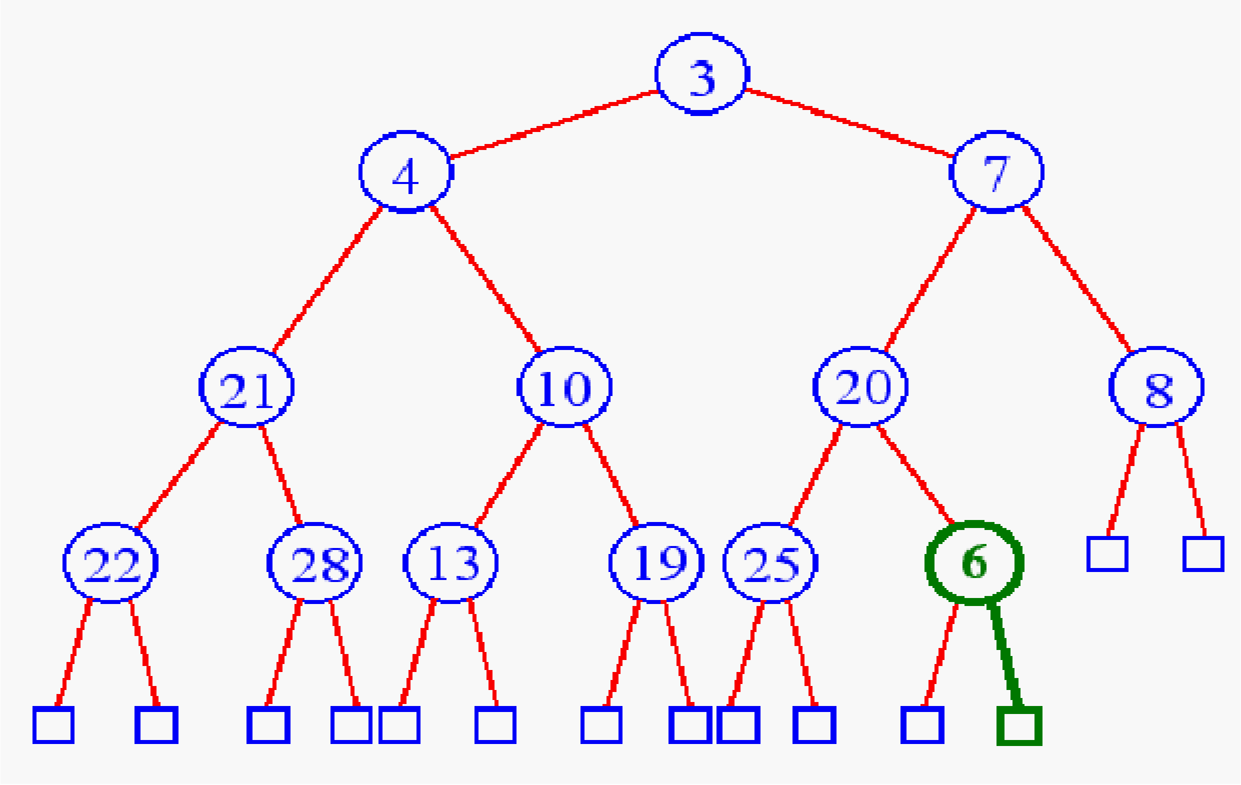
Min Heap Insertion

- Insert 6



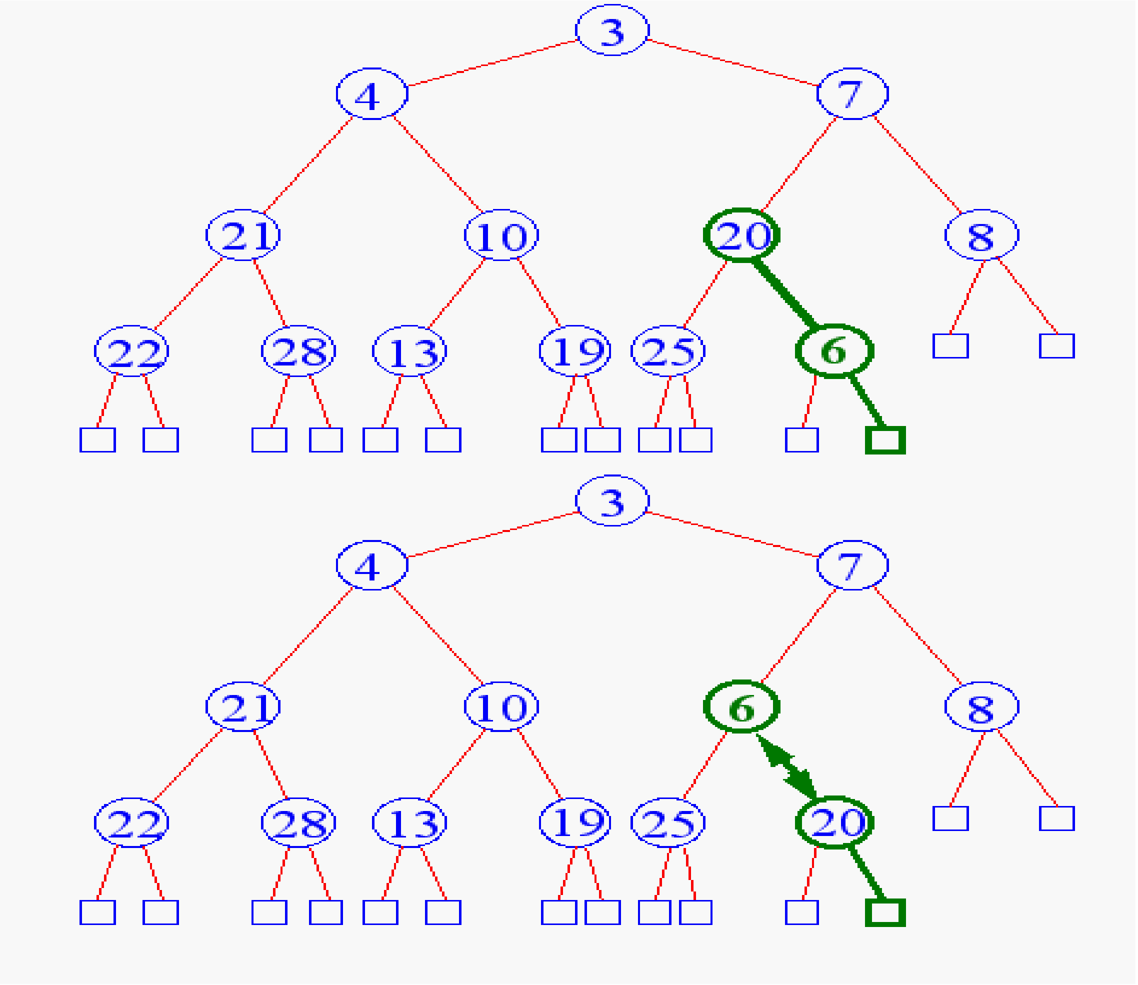
Heap Insertion

- Add key in next available position

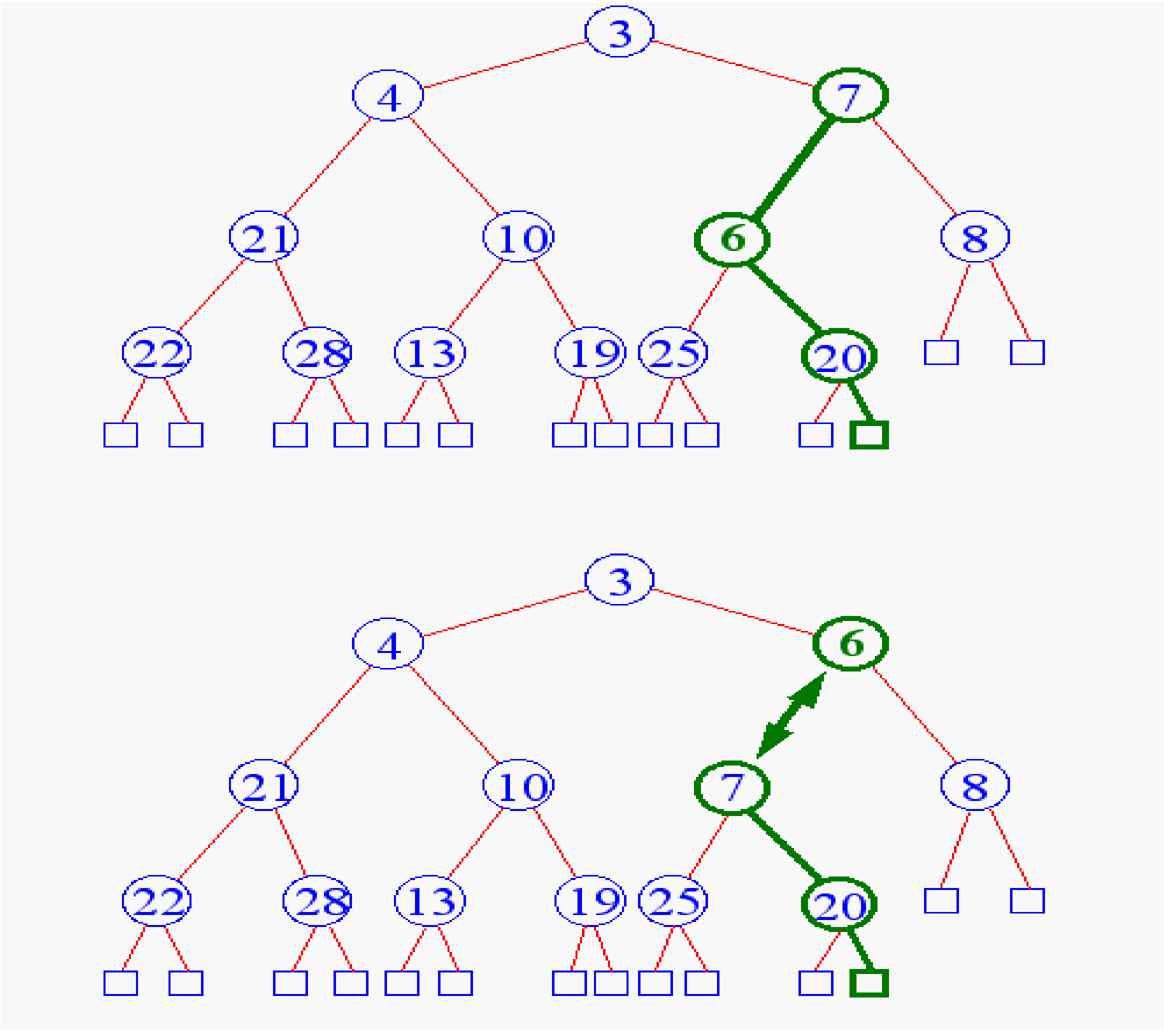


Heap Insertion

- Begin reheap up

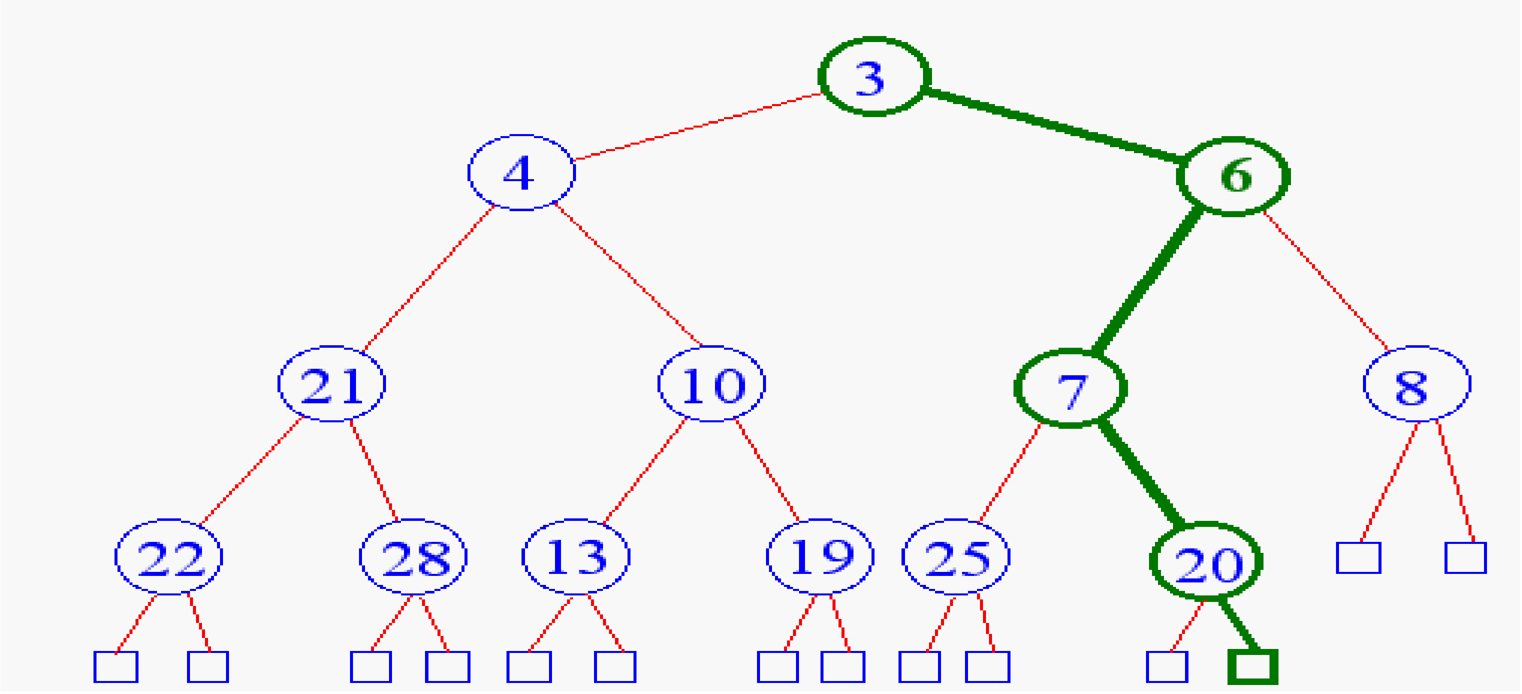


Heap Insertion



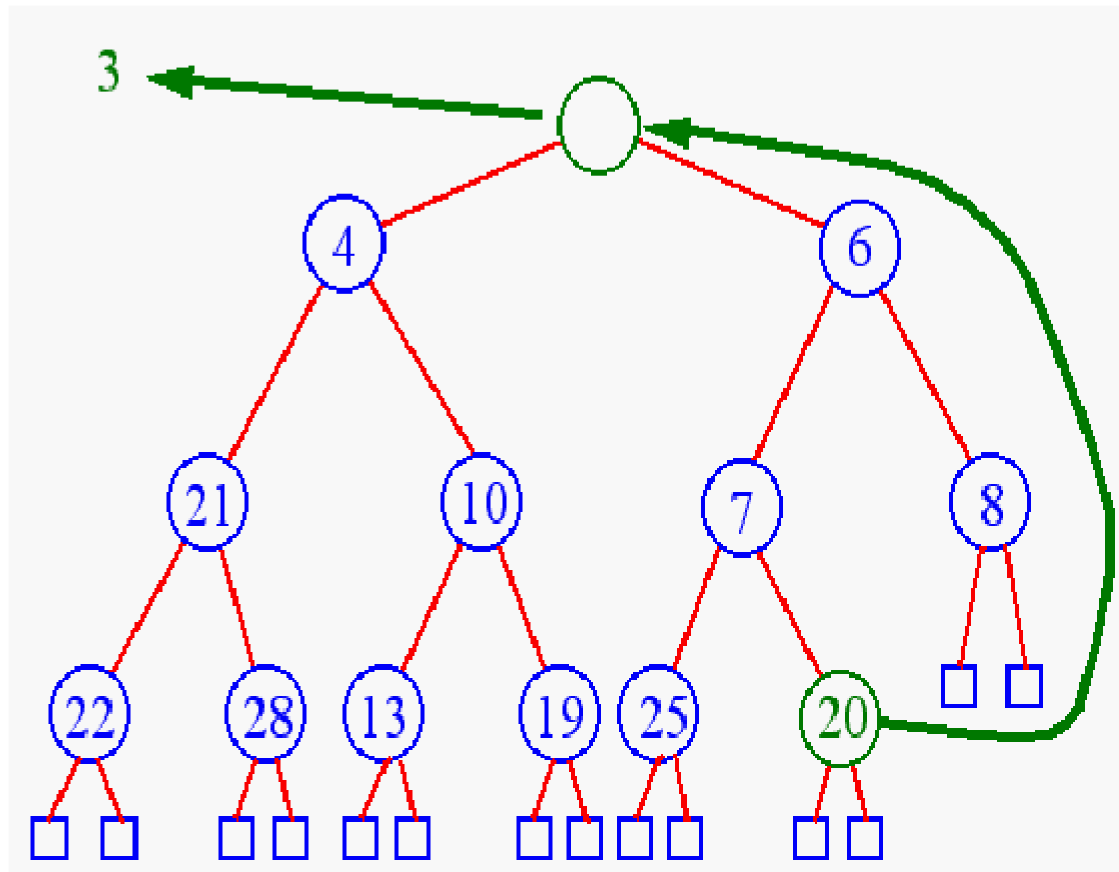
Heap Insertion

- Terminate reheap up when
  - reach root
  - key child is greater than key parent



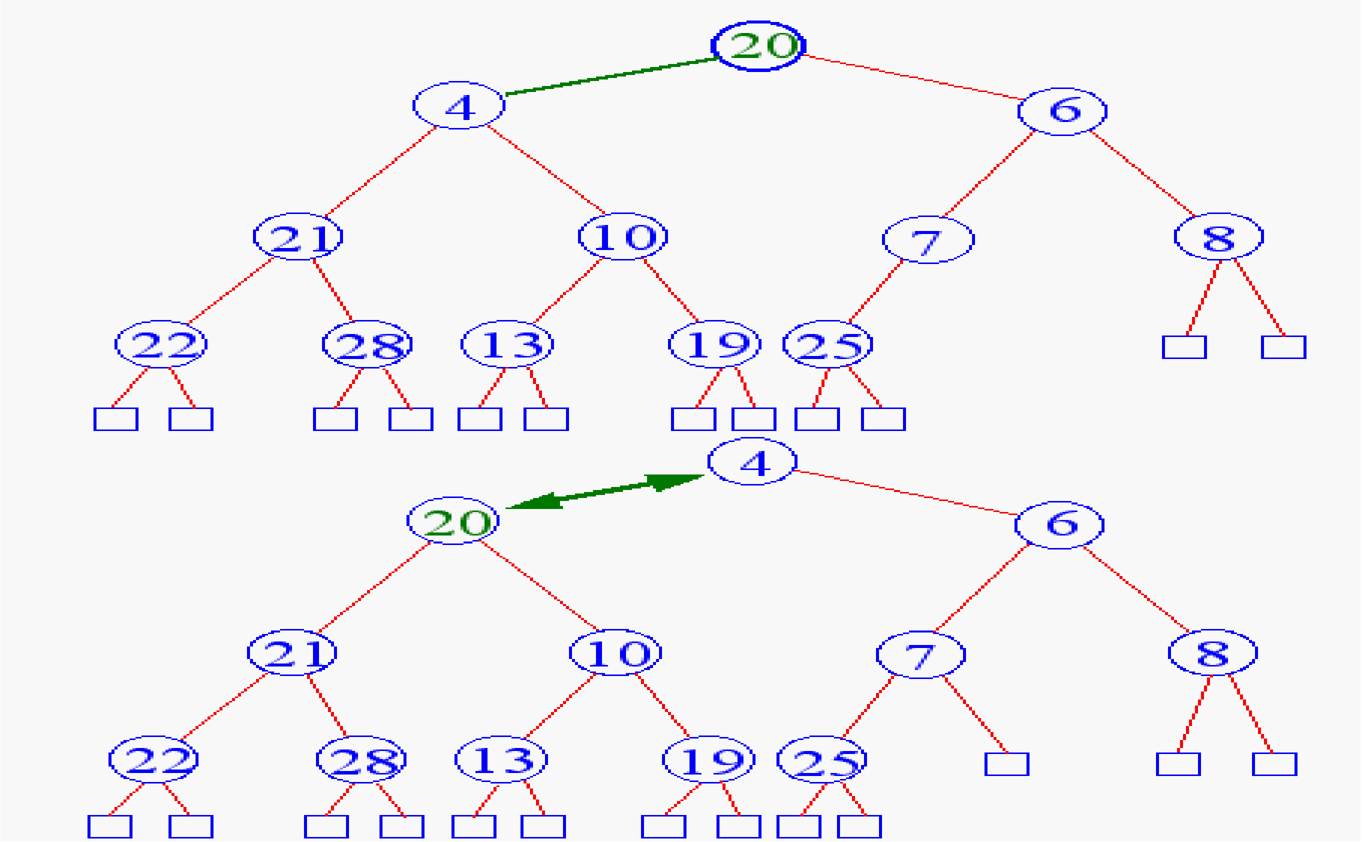
Min Heap Removal

- Remove min element from the heap
- Min Element on the root.

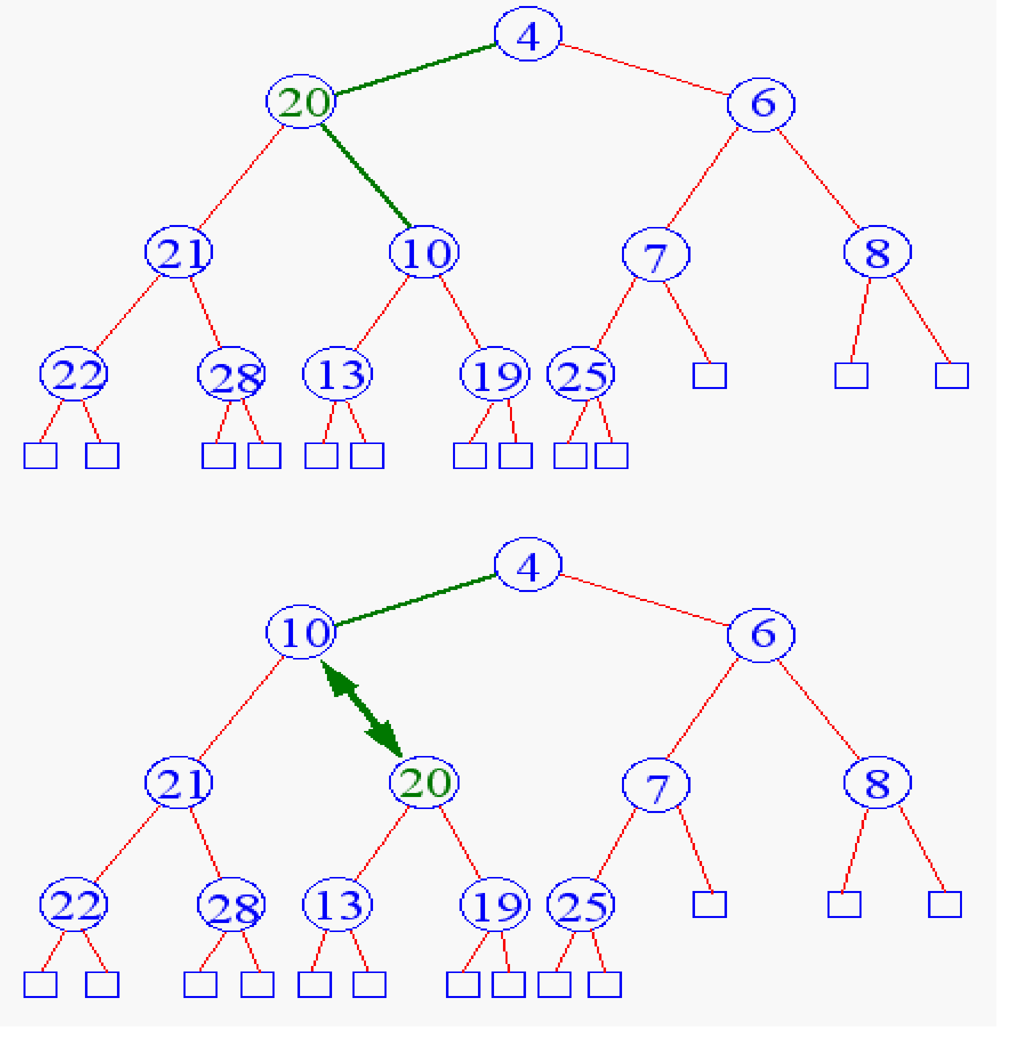


Heap Removal

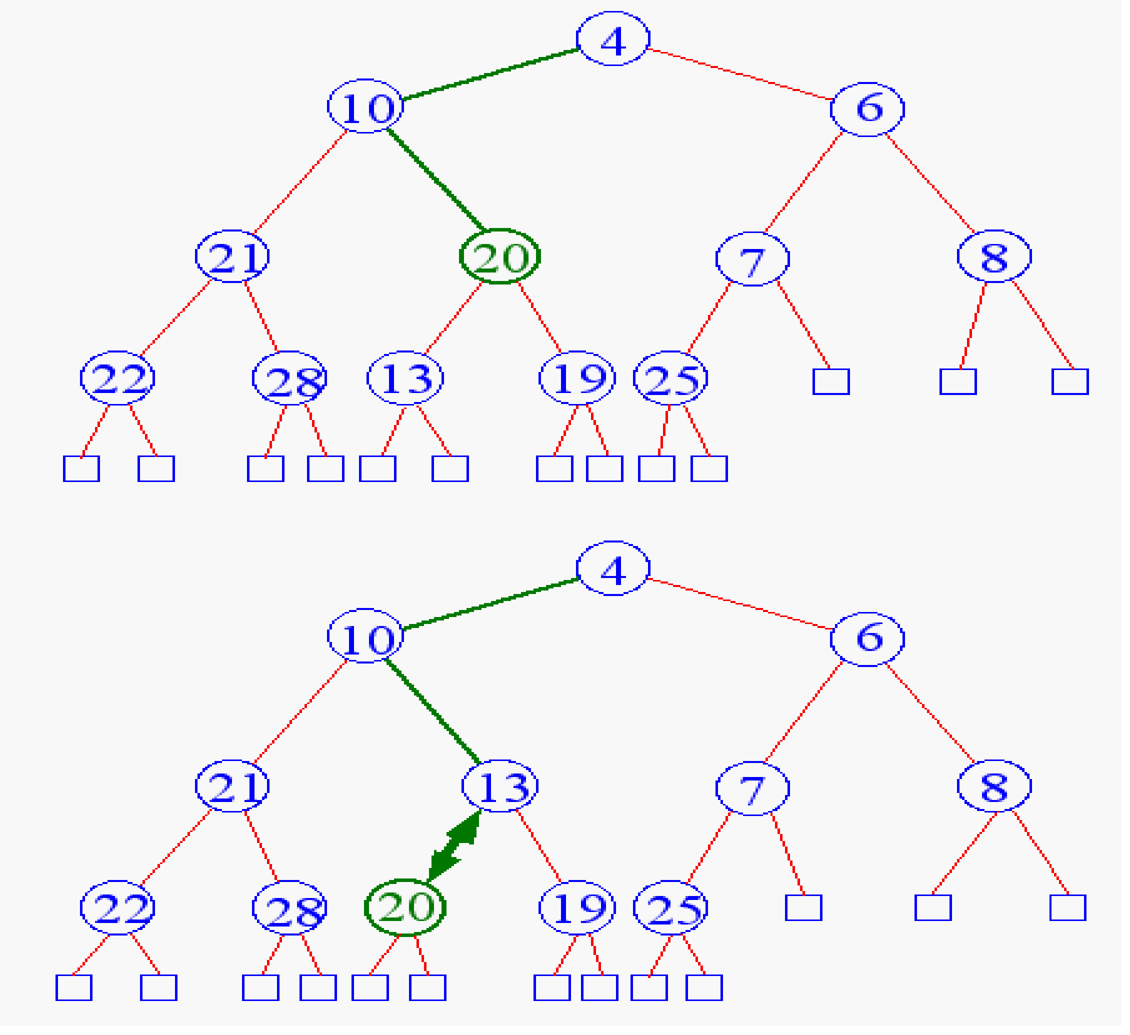
- Begin reheap down
  - Select the child with the minimum value



Heap Removal

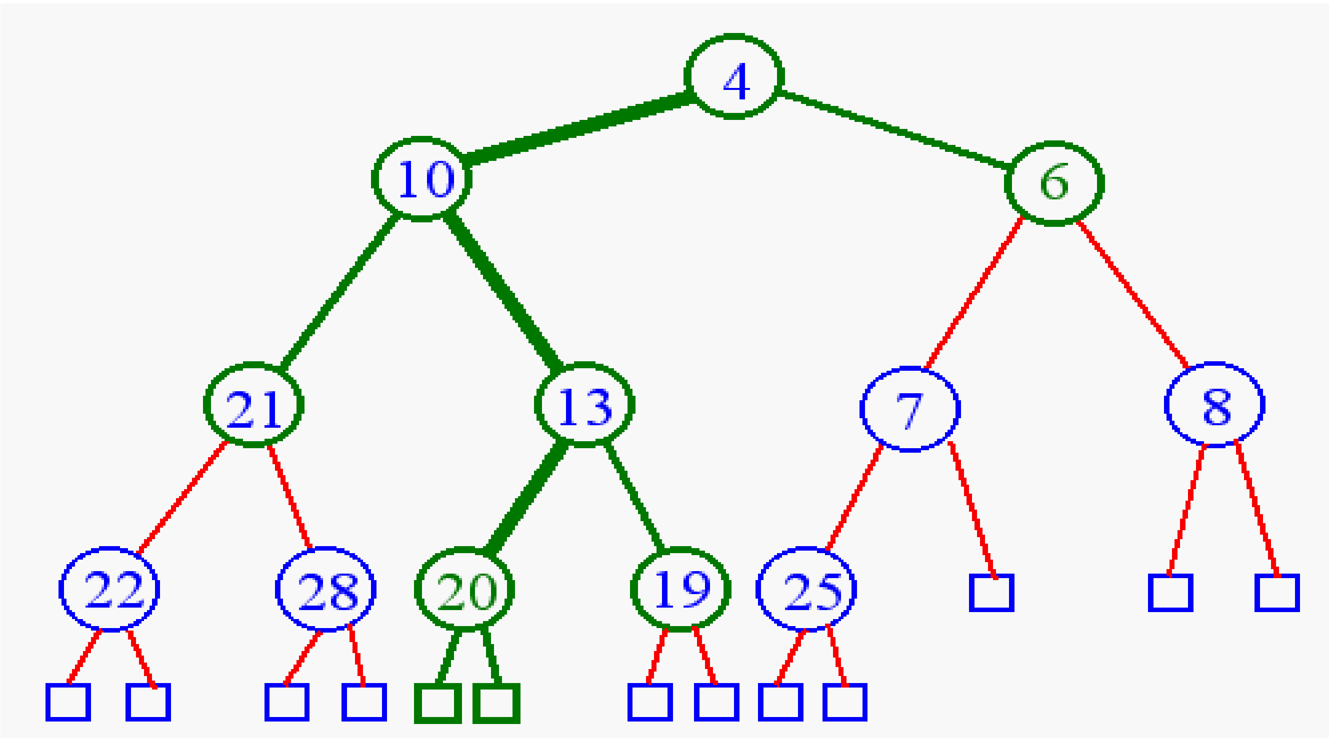


Heap Removal



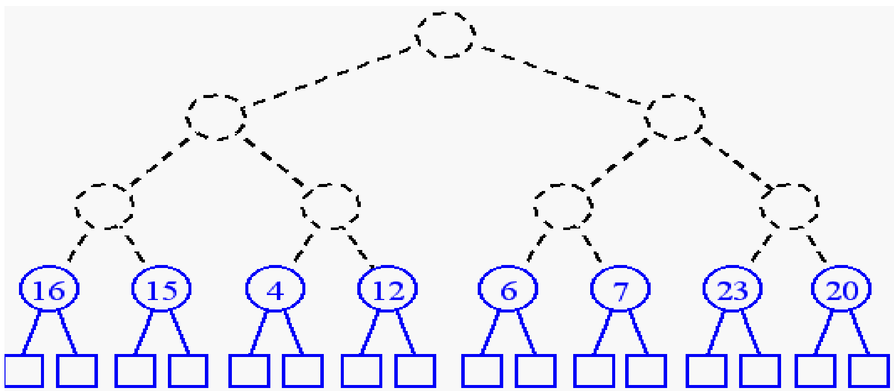
Heap Removal

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

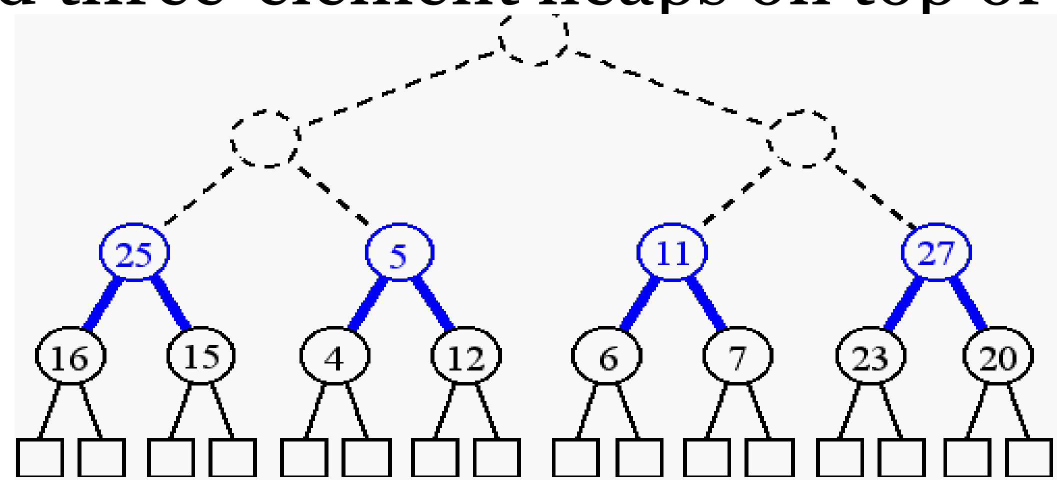


Building a Heap

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

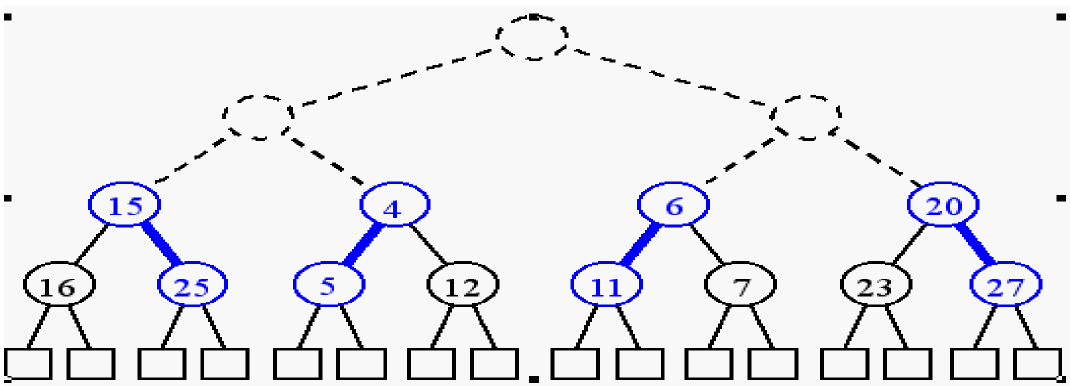


- build three-element heaps on top of them

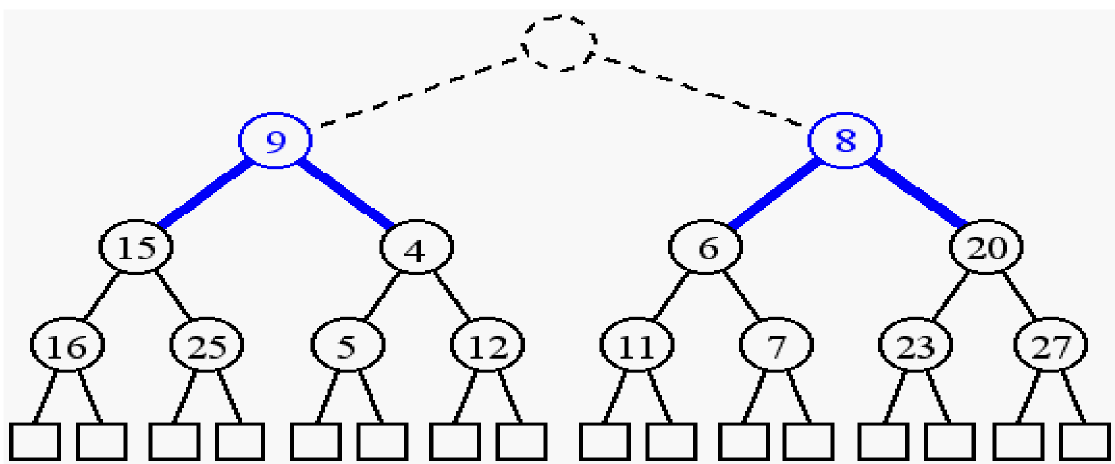


Building a Heap

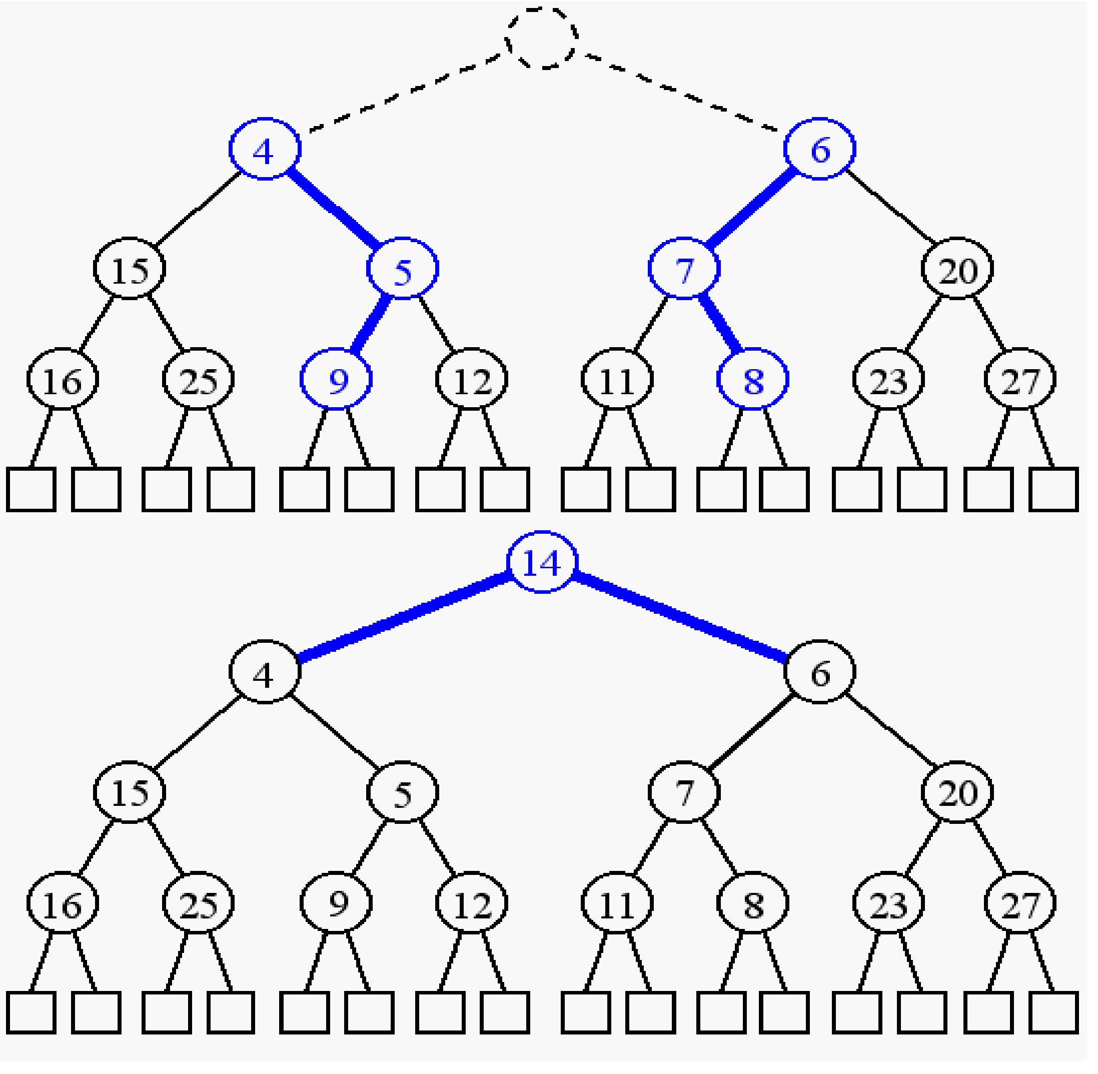
- *downheap* to preserve the order property



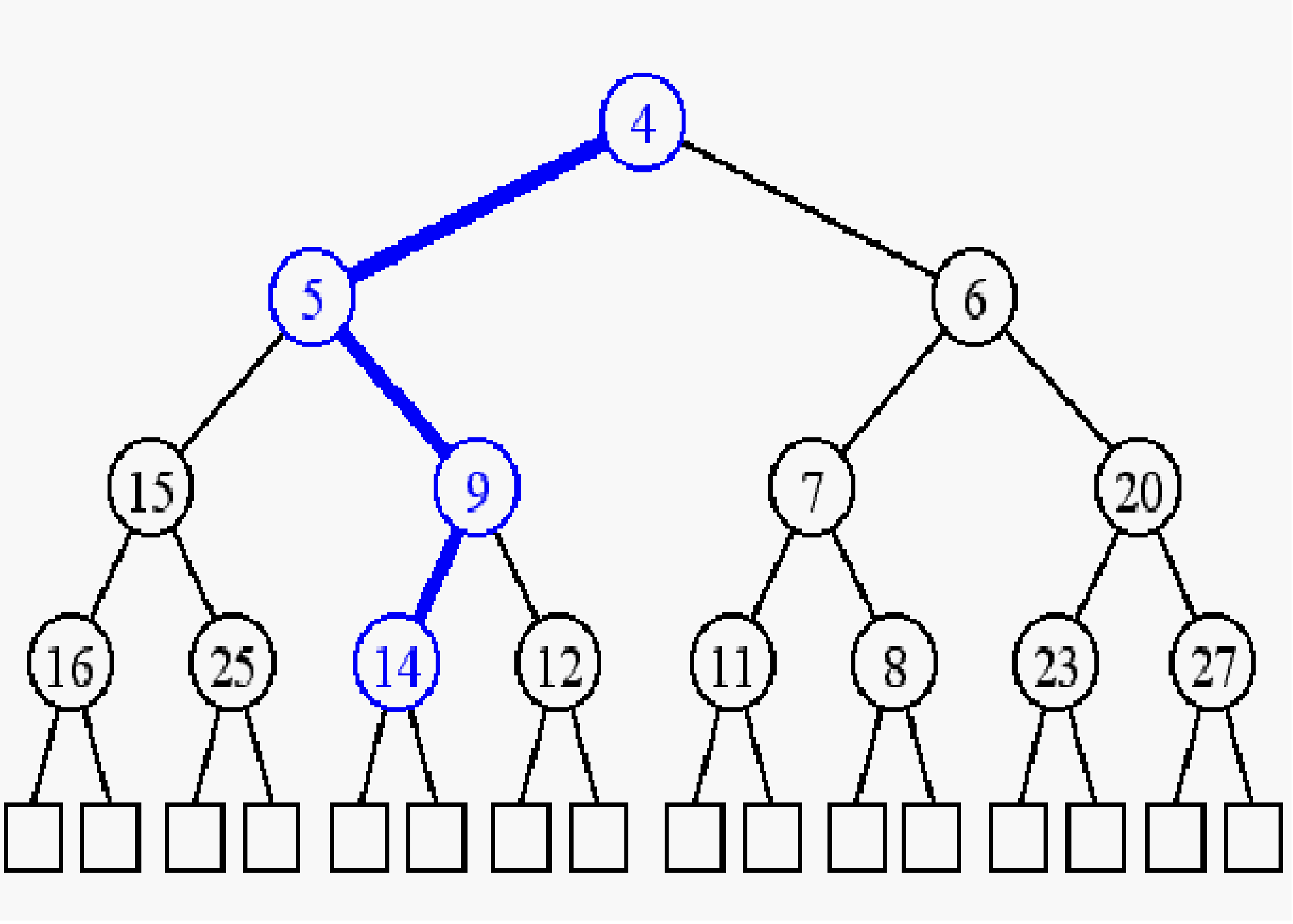
- now form seven-element heaps



Building a Heap



Building a Heap

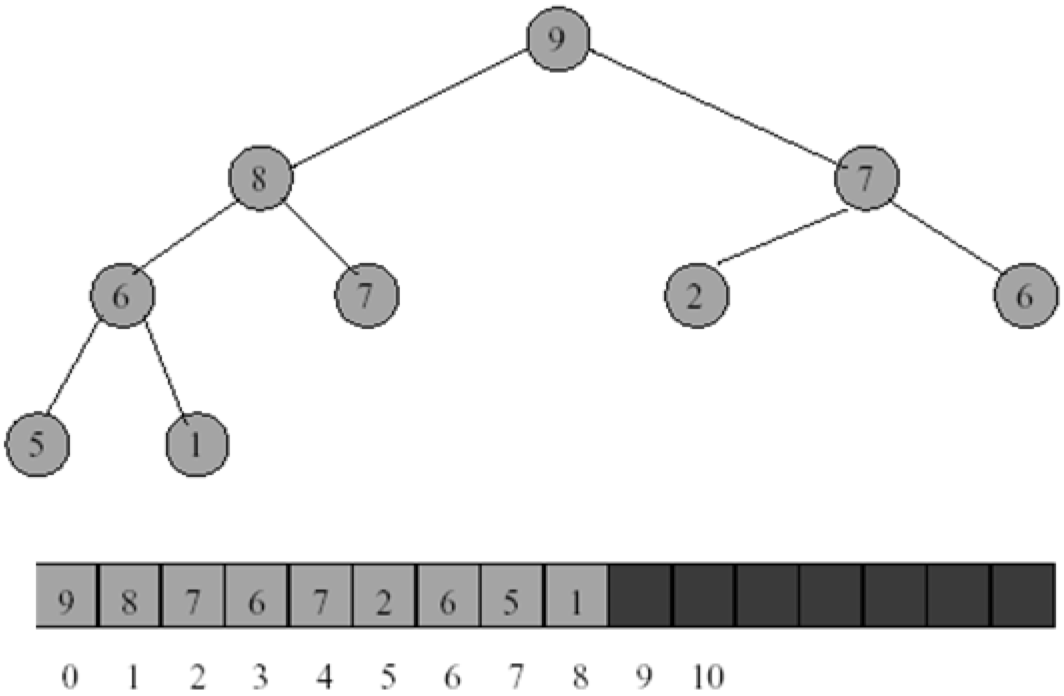


INITIALIZE

```
template<class T>
void maxHeap<T>::initialize(T *theHeap, int theSize)
{// Initialize max heap to element array theHeap[1:theSize].
  delete [] heap;
  heap = theHeap;
  heapSize = theSize;

  // heapify
  for (int root = heapSize / 2; root >= 1; root--)
  {
    T rootElement = heap[root];

    // find place to put rootElement
    int child = 2 * root; // parent of child is target
                          // location for rootElement
    while (child <= heapSize)
    {
      // heap[child] should be larger sibling
      if (child < heapSize && heap[child] < heap[child + 1])
        child++;

      // can we put rootElement in heap[child/2]?
      if (rootElement >= heap[child])
        break;  // yes

      // no
      heap[child / 2] = heap[child]; // move child up
      child *= 2;                    // move down a level
    }
    heap[child / 2] = rootElement;
  }
}
```

Array Representation of Heap

- A heap is efficiently represented as an array.



What are Heaps Useful for?

- **Heap Sort**: Insert all elements in the heap, then extract one by one
  - O(nlogn)
  - Other way with same order, is to initialize in O(n) not n insertions in O(nlogn)
- To implement **priority queues**, which has many applications
- 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