Data Structure
Chapter 9 Heap Structure

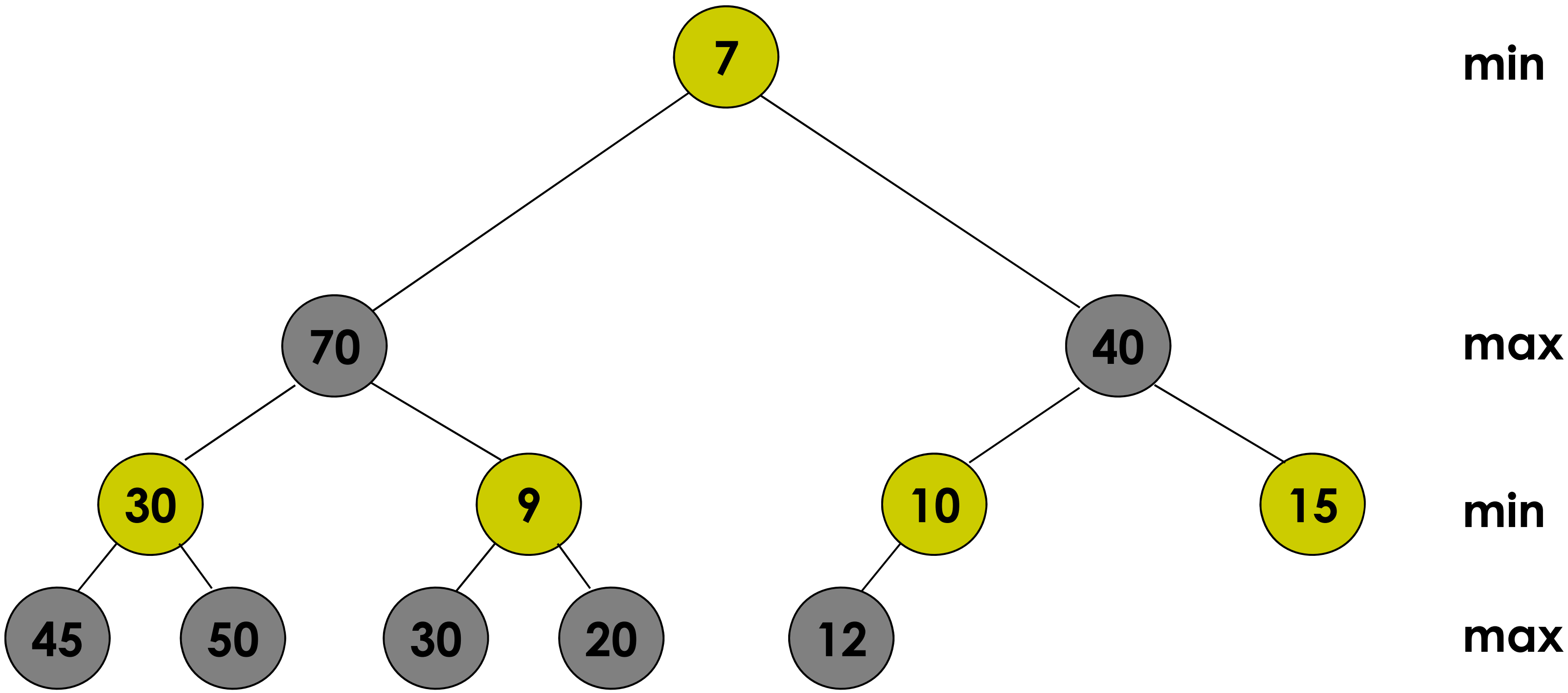
Angela Chih-Wei Tang
Department of Communication Engineering
National Central University
Jhongli, Taiwan

2010 Spring

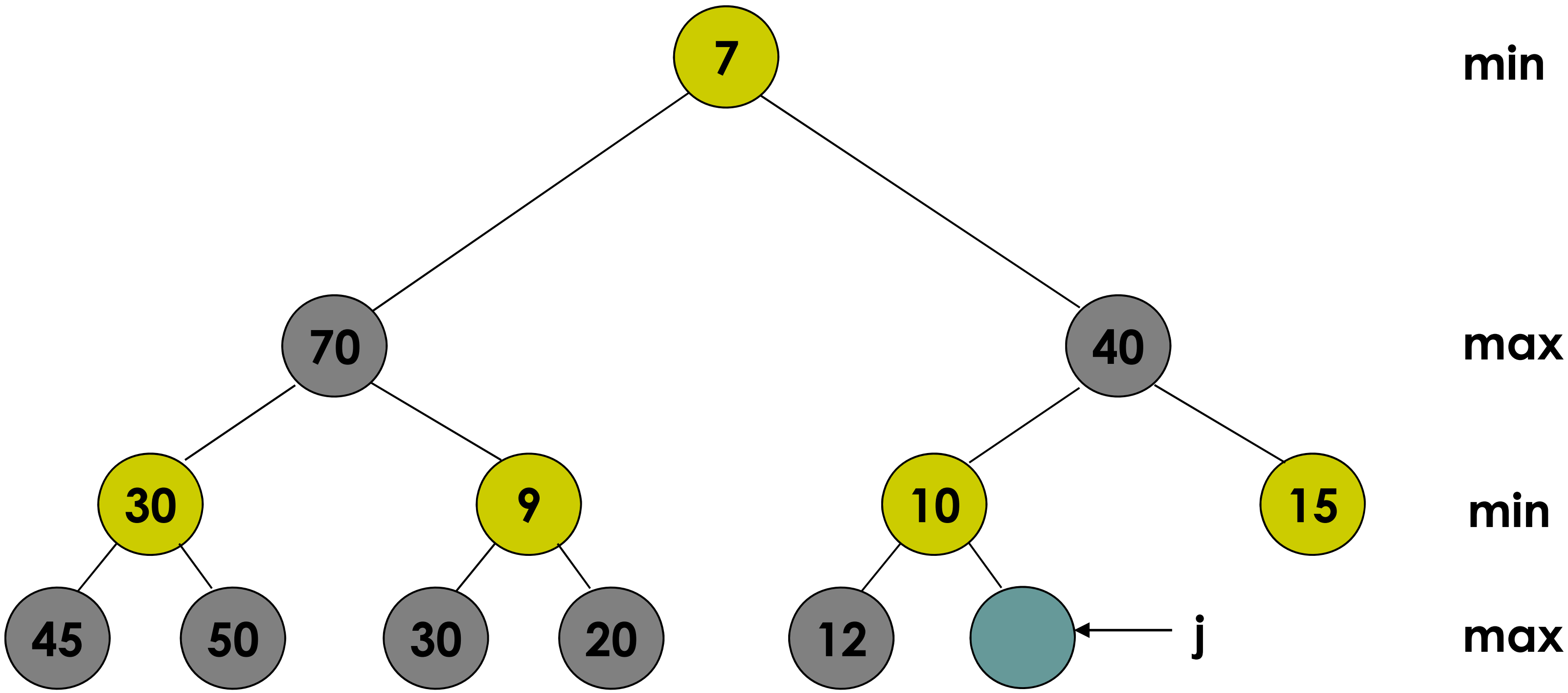
Min-Max Heaps

- A Min-Max Heap supports Double-ended priority queue!
  - Inserting an element with an arbitrary key
  - deleting an element with the largest key
  - deleting an element with the smallest key
- Definition:
  - A **min-max heap** is a **complete binary tree** such that if it is not empty, each element has a data member called **key**.
  - Alternating levels of this tree are min levels and max levels, respectively.
  - The root is on a min level. Let x be any node in a min-max heap. If x is on a min (max) level then the element in x has the minimum (maximum) key from among all elements in the subtree with root x. A node on a min (max) level is called a min (max) node.

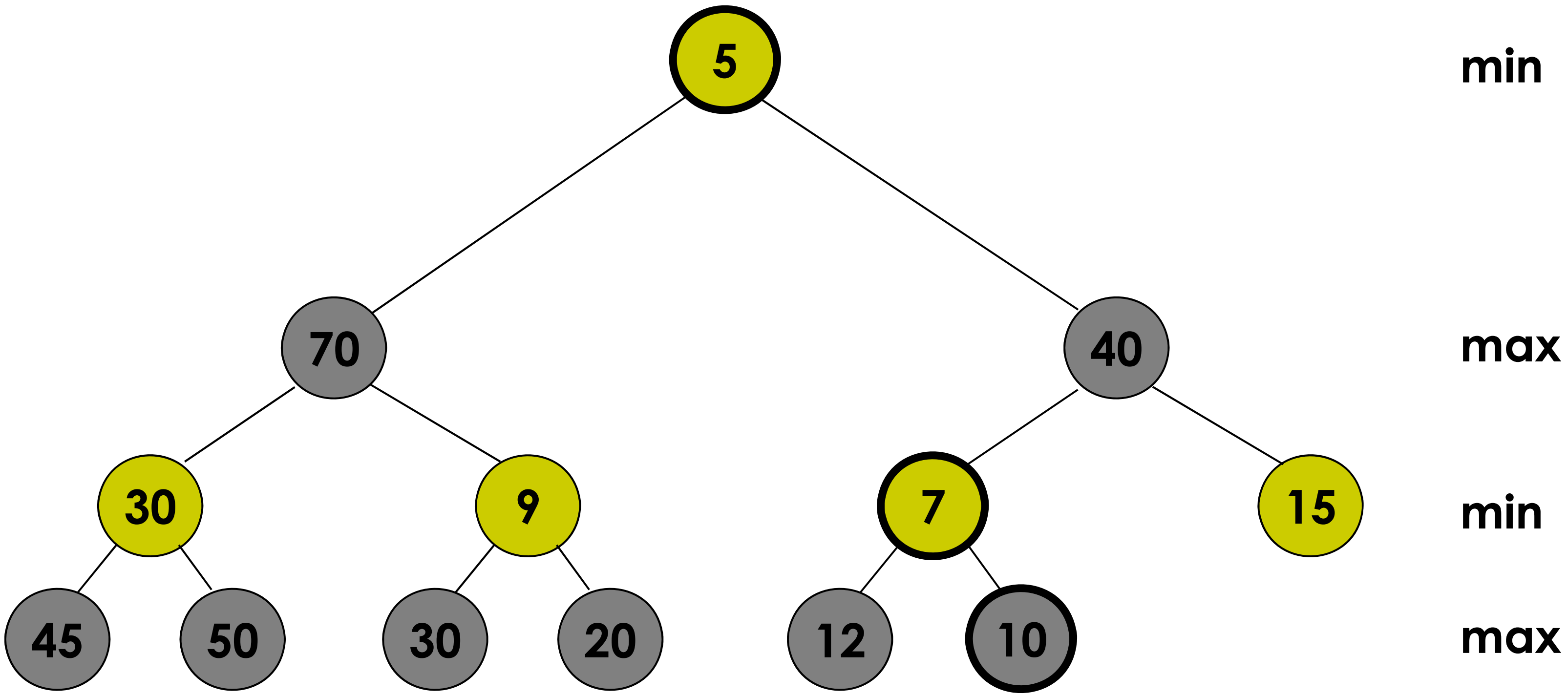
Figure 9.1: A 12-element Min-Max Heap



Insert to Min-Max Heap

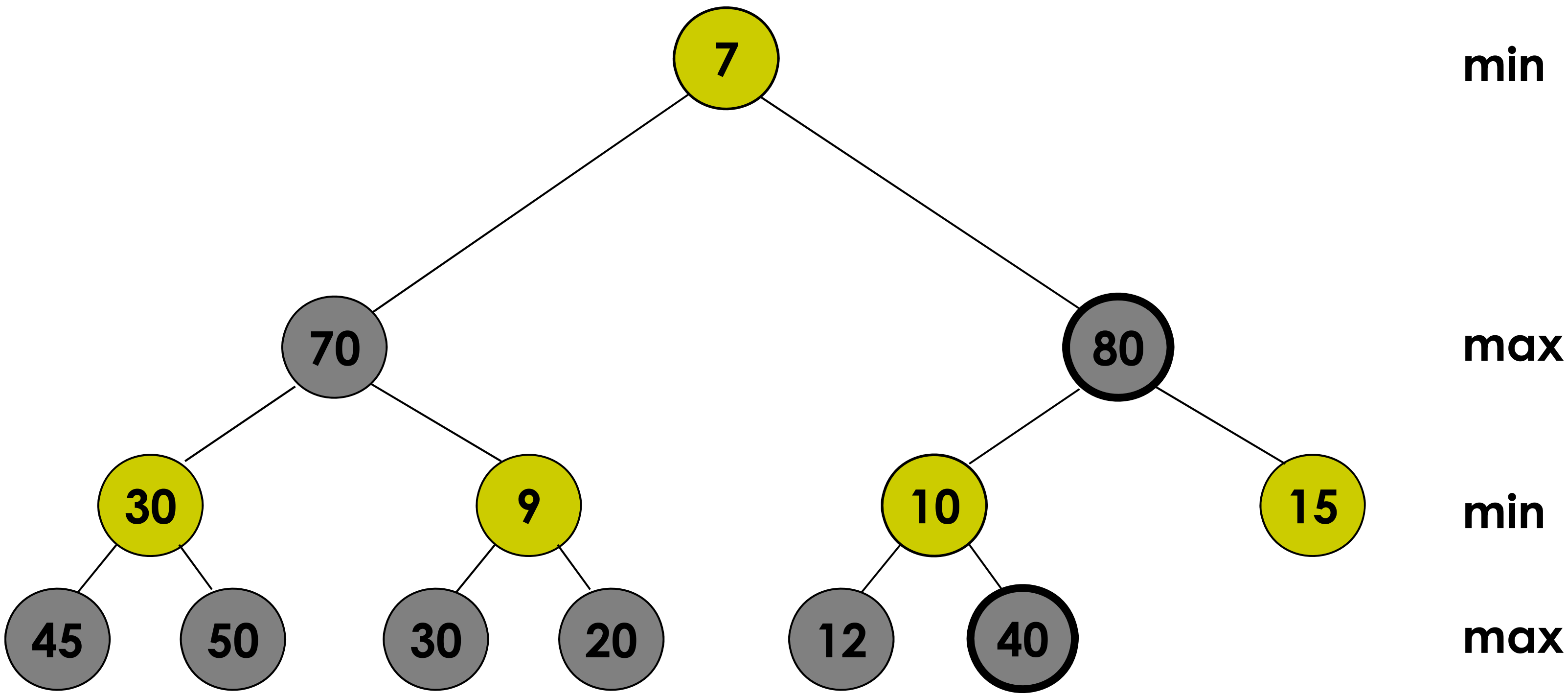


Min-Max Heap After Inserting Key 5



- 5 is guaranteed to be smaller than all keys in nodes that are both on max levels and on the path from j to root.
- Only need to check nodes on min levels.

Min-Max Heap After Inserting Key 80



- 80 is larger than all keys in the nodes that are both on min levels and on the path from j to the root.
- Only need to check nodes on max levels.

Program 9.3: Insertion Into A Min-Max Heap (1/2)

```
void min_max_insert(element heap[], int *n, element item)
{
 /* insert item into the min-max heap*/
 int parent;
 (*n)++;
 if(*n==MAX SIZE) {
   fprintf(stderr, "The heap is full\n");
   exit(1);
 }
 parent = (*n)/2;
 if (!parent)
 /* heap is empty, insert item into first position */
 heap[1] = item;
 else switch(level(parent)) {
```

Program 9.3: Insertion Into A Min-Max Heap (2/2)

```
    case FALSE: /* min level */
        if (item.key<heap[parent].key) {
                heap[*n]=heap[parent];
                verify_min(heap,parent,item);
        }
        else
                verify_max(heap,*n,item);
        break;
    case TRUE;    /*max level*/
        if (item.key > heap[parent].key) {
                heap[*n] = heap[parent];
                verify_max(heap, parent, item);
        }
        else
                verify_min(heap,*n,item);
    }

}
```

Program 9.2: verify_max: function

```
void  verify_max(element heap[], int i, element item)
{
/* follow the nodes from the max node i to the root and insert item into
    its proper place */
    int grandparent = i/4;
    while (grandparent)
            if(item.key>heap[gradparent].key) {
                        heap[i]=heap[grandparent];
                        i=grandparent;
                        grandparent/=4;
            }
            else
                        break;
    heap[i]=item;
}
```

Figure 9.1:
A 12-element Min-Max Heap

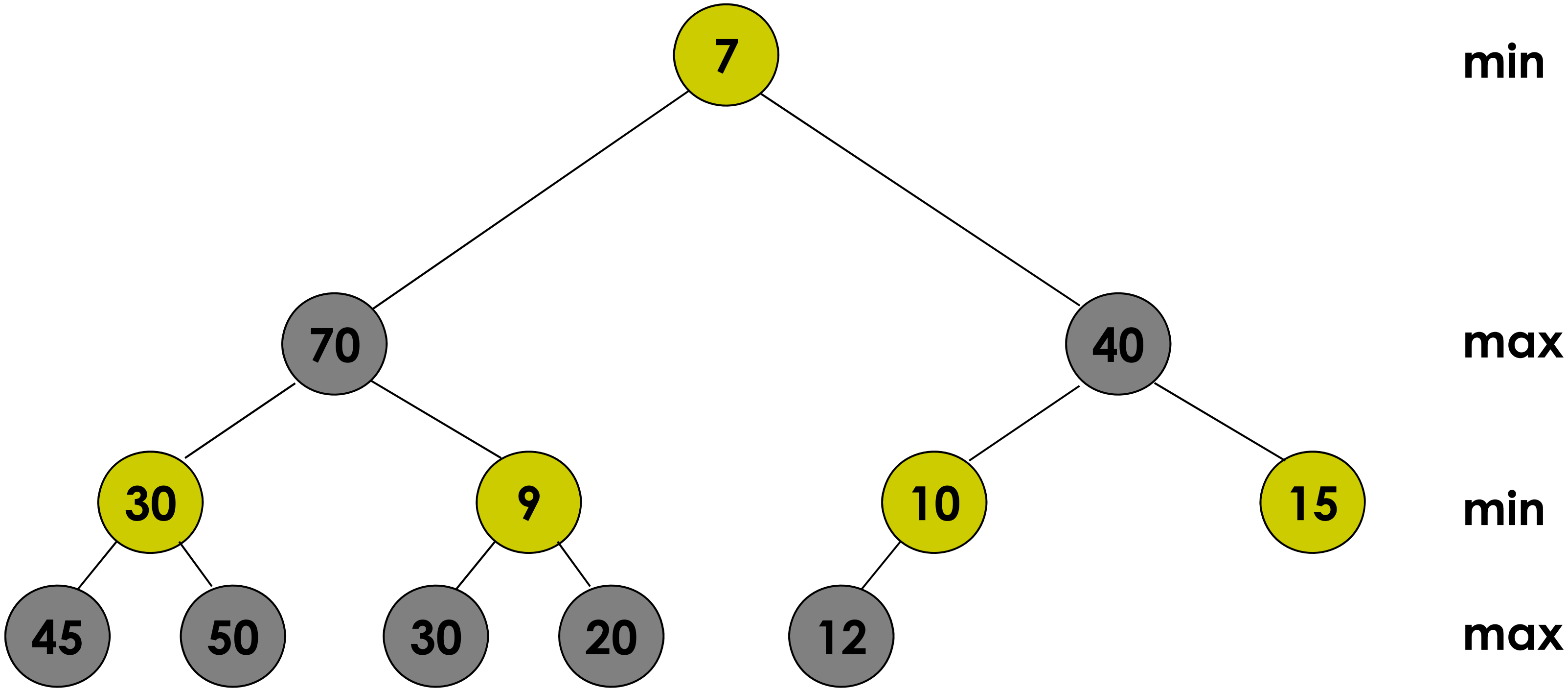
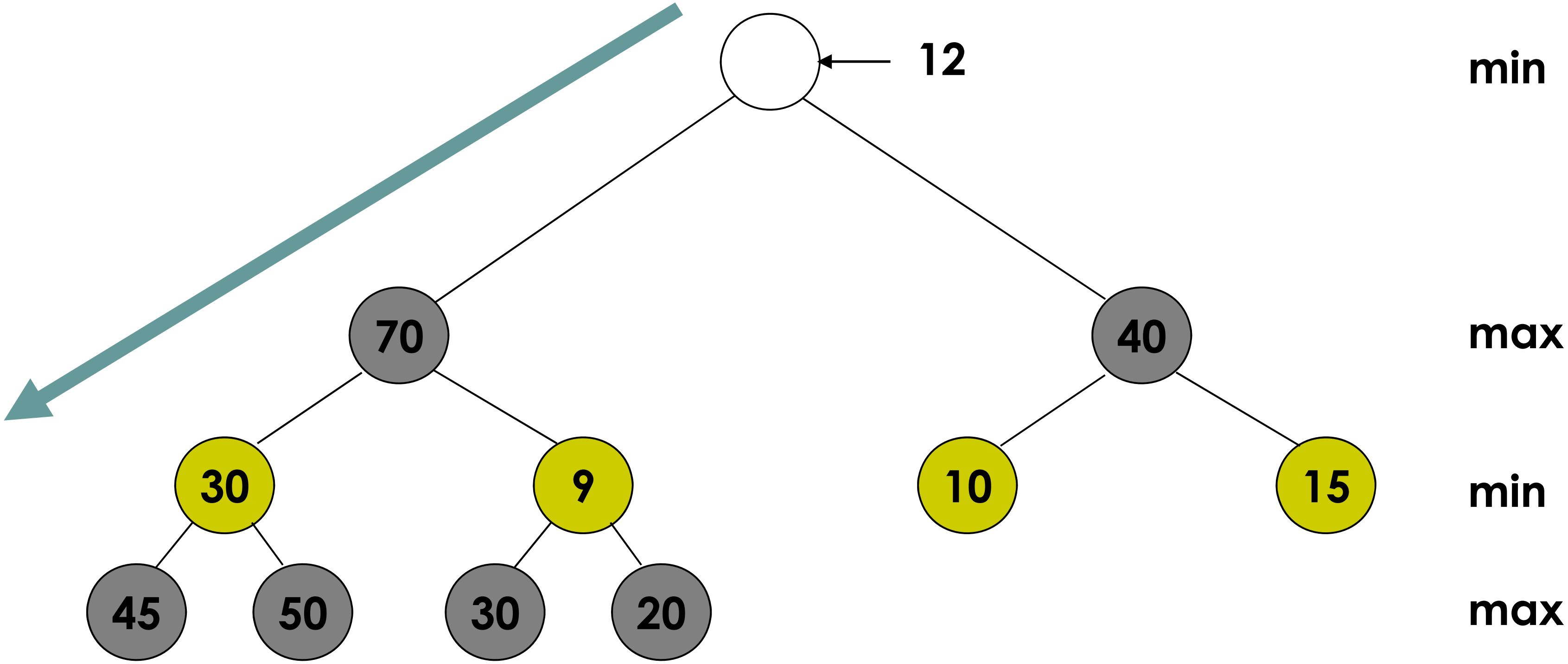
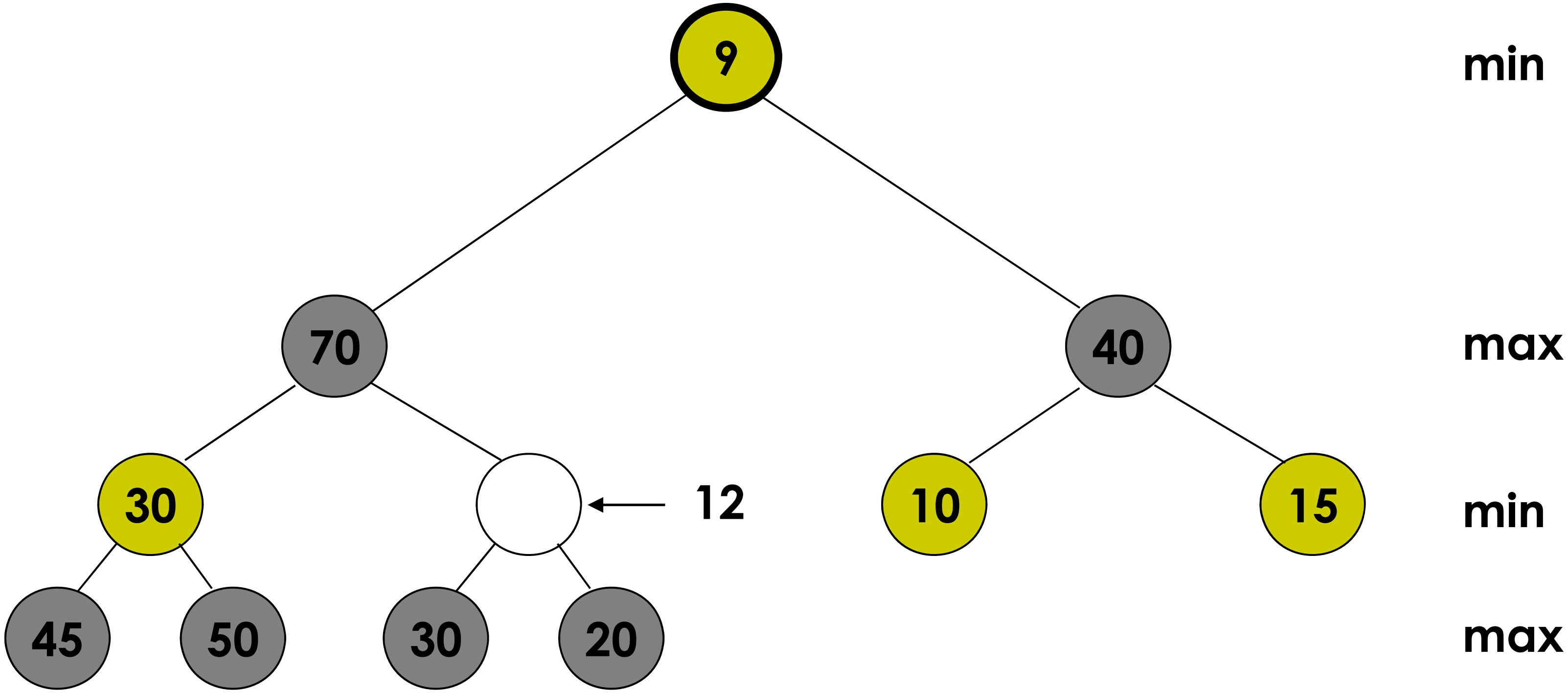


Figure 9.4: Deletion of the Min Element



The reinsertion is done by examining the nodes from the root down towards the leaves!

Min-Max Heap After Deleting Min Element



How to delete the element with the maximum key?

Deletion of the Min Element

- **Delete the root!**
- The last element is deleted from the min-max heap and then reinsert into the min-max heap.
  - The root has no children: insert x into the root.
  - The root has at least one child.
    - **x.key ≤ h[k].key:** x may be inserted into the root.
    - **x.key >h[k].key and k is a child of the root**.
      - Since k is a max node, it has not descendents with key larger than h[k].key.
      - So, node k has no descendents with key larger than x.key. The element h[k] may be moved to the root, and x can be inserted into node k.
    - **x.key> h[k].key and k is a grandchild of the root**.:
      - h[k] is moved to the root. Let p the parent of k.
      - If x.key > h[p].key, then h[p] and x are to be interchanged.

Appendix. Plot of Average Times of Different Sorting Algorithms