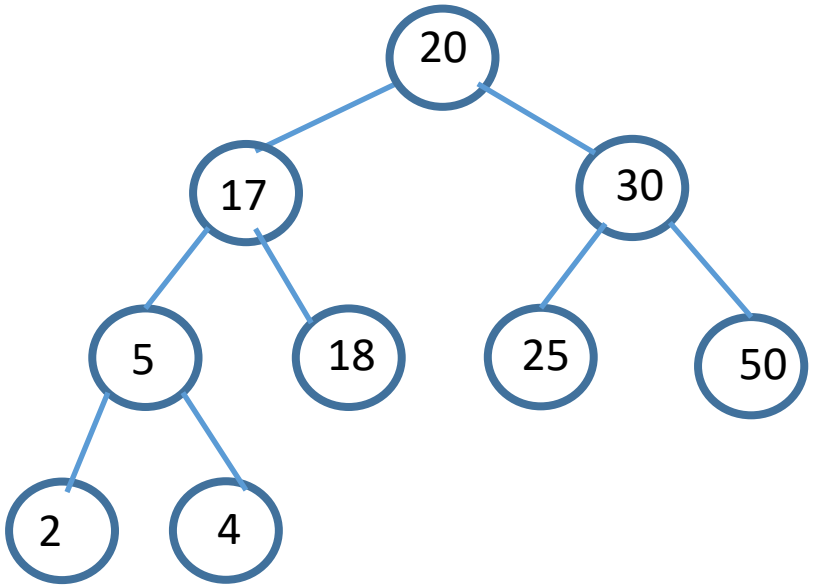
COMP 2611, DATA STRUCTURES
LECTURE 13

HEAPS: MAX-HEAPS

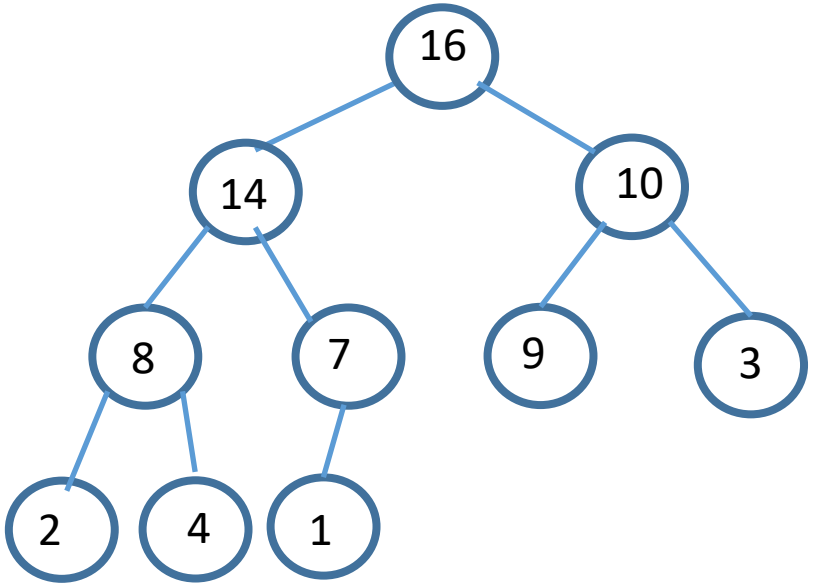
- **Maintaining the max-heap property**
- **Building a max-heap**
- **Finding the maximum value in a max-heap**
- **Deleting a value from a max-heap**

Almost Complete Binary Tree

- An *almost complete* binary tree is one in which:
  - ✓ All levels, except possibly the lowest, are completely filled.
  - ✓ The nodes at the lowest level (all leaves) are as far left as possible.



Heap: An Almost Complete Binary Tree



- Store the elements of the binary tree in an array:

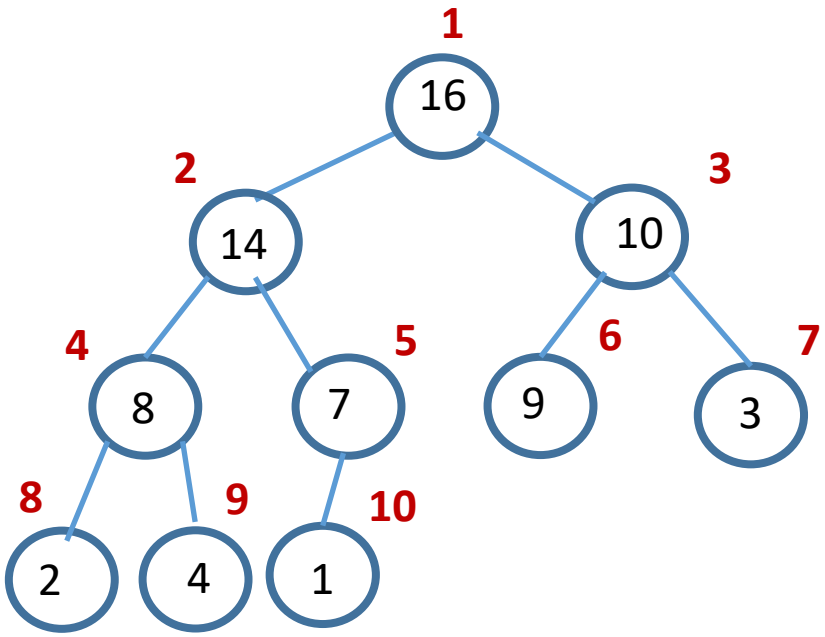
0	1	2	3	4	5	6	7	8	9	10
	16	14	10	8	7	9	3	2	4	1

A

Don't use location 0

Functions for a Heap

```
Parent (i):
    return floor (i/2)

Left (i):
    return 2*i

Right (i):
    return 2*i + 1
```



- *i* is the index of a node in the array:

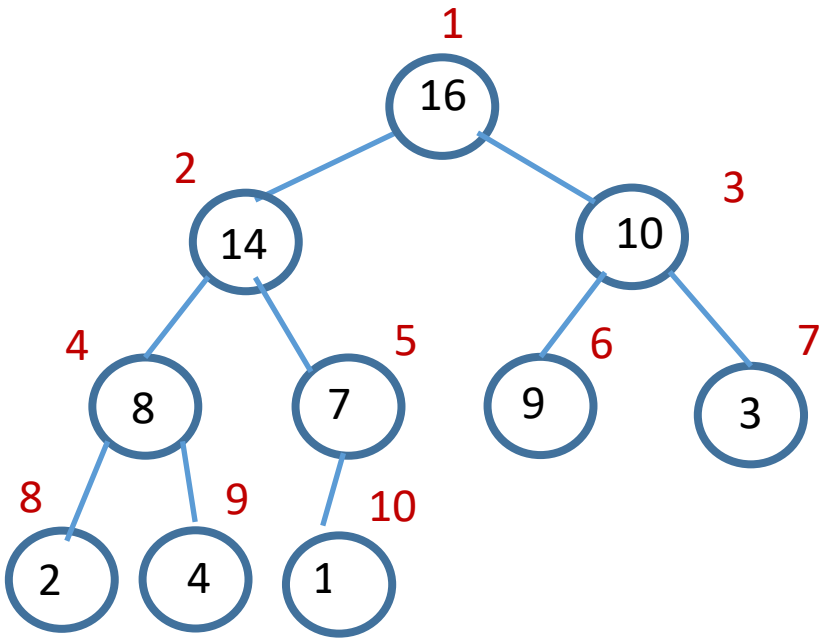
0	1	2	3	4	5	6	7	8	9	10
	16	14	10	8	7	9	3	2	4	1

location 0 is not used

A Max-Heap

A max-heap satisfies the max-heap property:

A [Parent(i)] ≥ A[i]

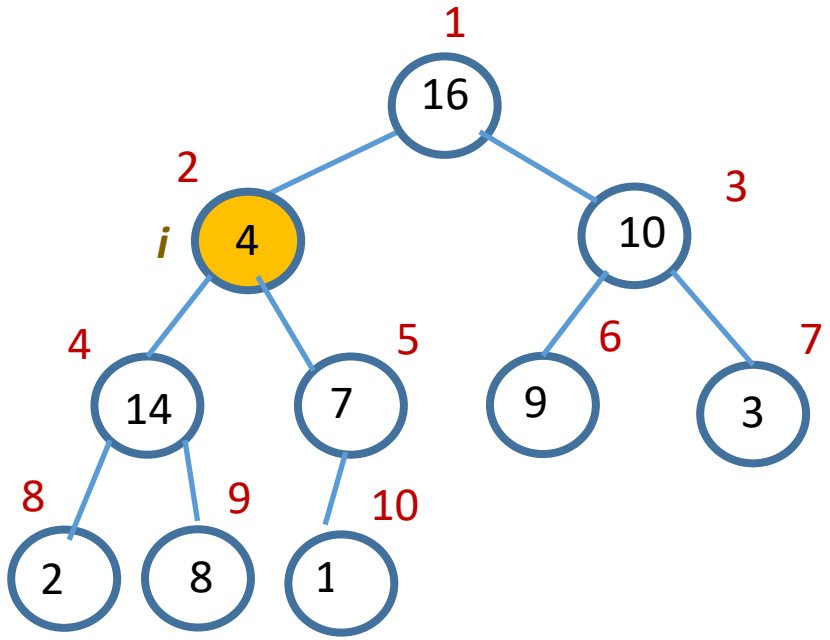


0	1	2	3	4	5	6	7	8	9	10
	16	14	10	8	7	9	3	2	4	1

A

Maintaining the Max-Heap Property

- Suppose we know that:
  - The binary trees rooted at Left (i) and Right (i) are max-heaps, but,
  - A[i] might be smaller than its children.

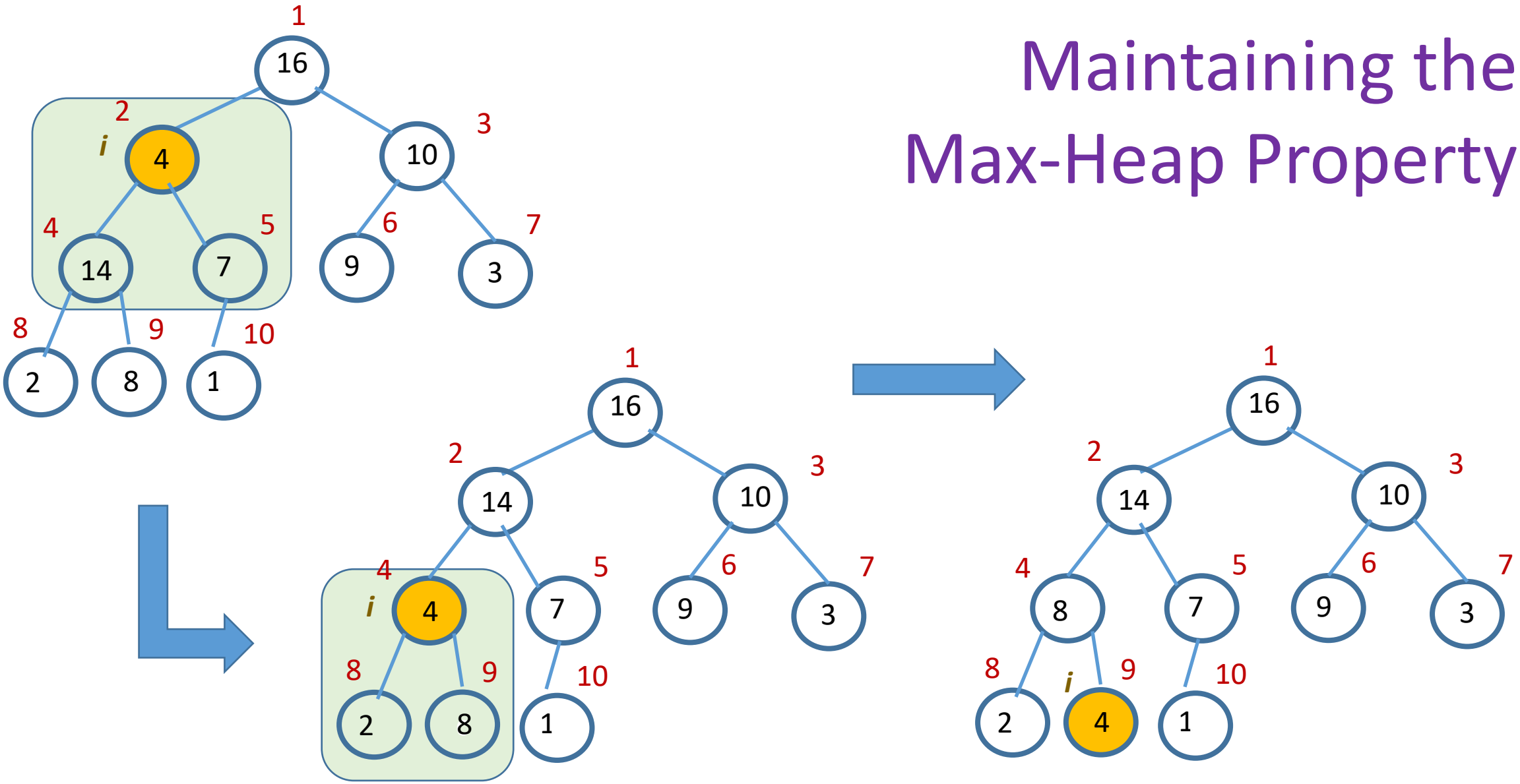


- How can we maintain the max-heap property?

0	1	2	3	4	5	6	7	8	9	10
	16	4	10	14	7	9	3	2	8	1

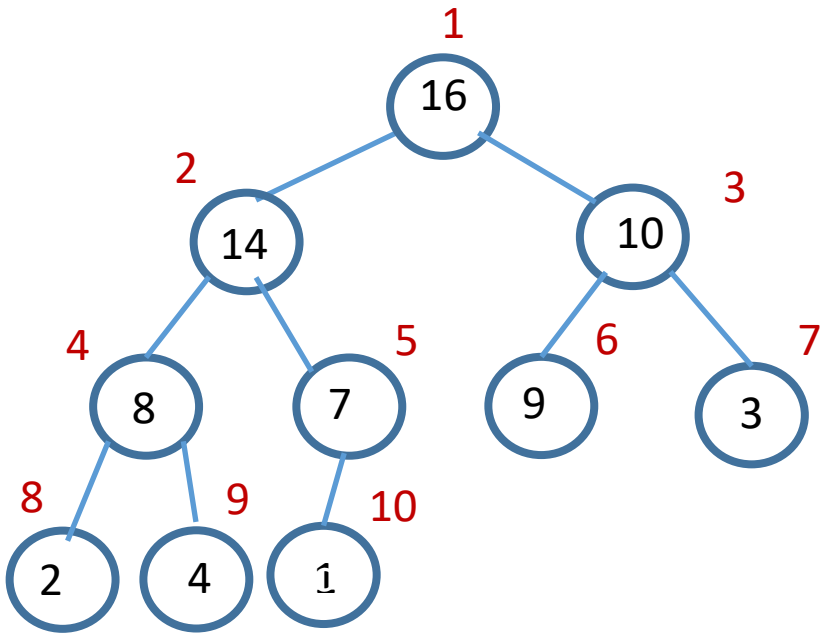
A

Maintaining the Max-Heap Property



We can write a function, **maxHeapify** (A, *i*) to restore the binary tree to a max-heap.

Declaration of a Max-Heap




```
struct MaxHeap {
   int A [1000];
   int size;
};

MaxHeap * heap;
```

- *i* is the index of a node in the array:

0	1	2	3	4	5	6	7	8	9	10
	16	14	10	8	7	9	3	2	4	1

location 0 is not used

The *maxHeapify* Function

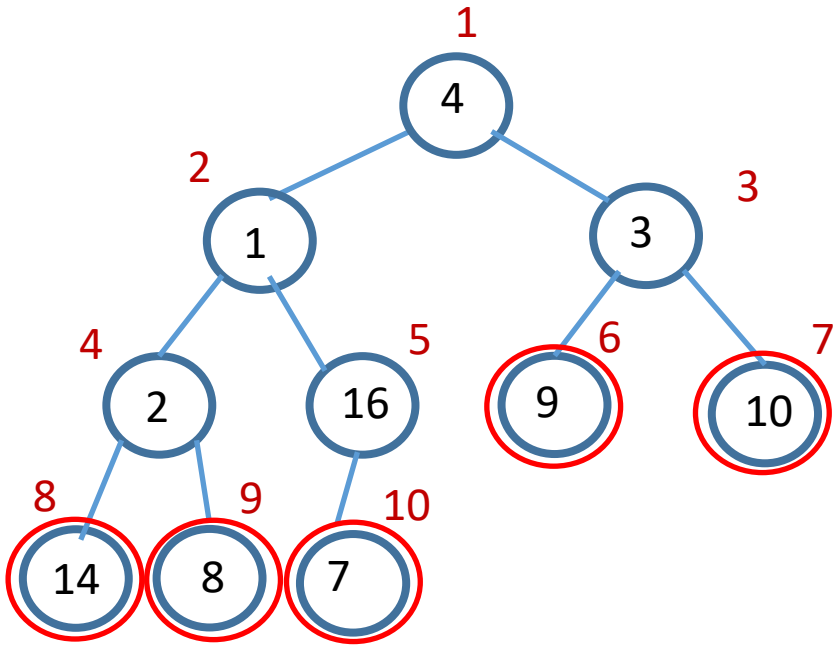
```
maxHeapify (MaxHeap * heap, int i) {

        left = i * 2;
        right = i * 2 + 1;

        largest = index of largest of:
                              heap->A[i],
                              heap->A[left],
                              heap->A[right]


        if (largest != i) {
                swap heap->A[largest] with heap->A[i];
                maxHeapify(heap, largest);
        }
}
```

Building a Max-Heap

- Draw the almost-complete binary tree obtained from the set of values in the following array, *A*:

0	1	2	3	4	5	6	7	8	9	10
	4	1	3	2	16	9	10	14	8	7

A



- Apply the *maxHeapify* function to all nodes, starting from 10 and going to 1.

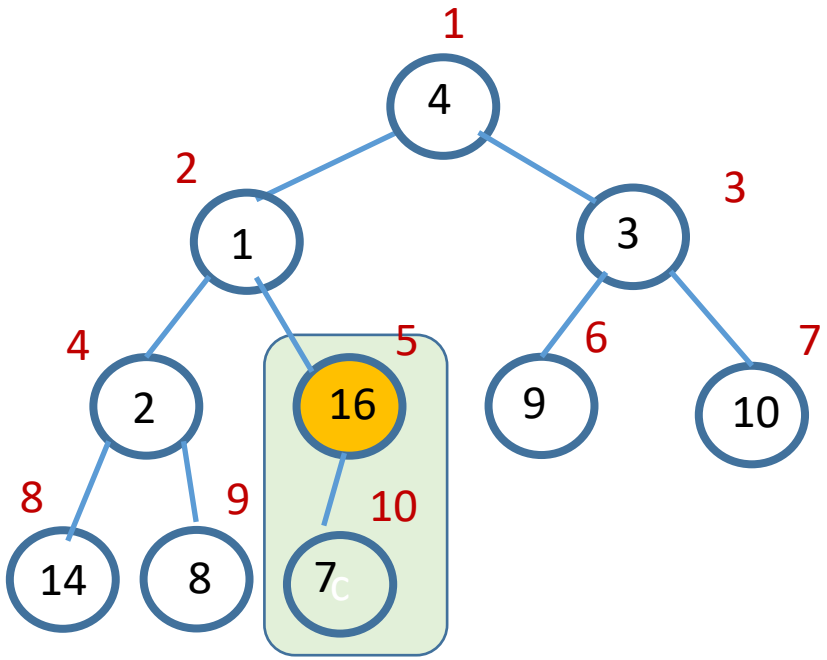
How to *maxHeapify* a leaf?

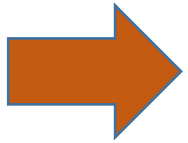
Building a Max-Heap

- Draw the binary tree obtained from the set of values in the following array, *A*:

0	1	2	3	4	5	6	7	8	9	10
	4	1	3	2	16	9	10	14	8	7

A





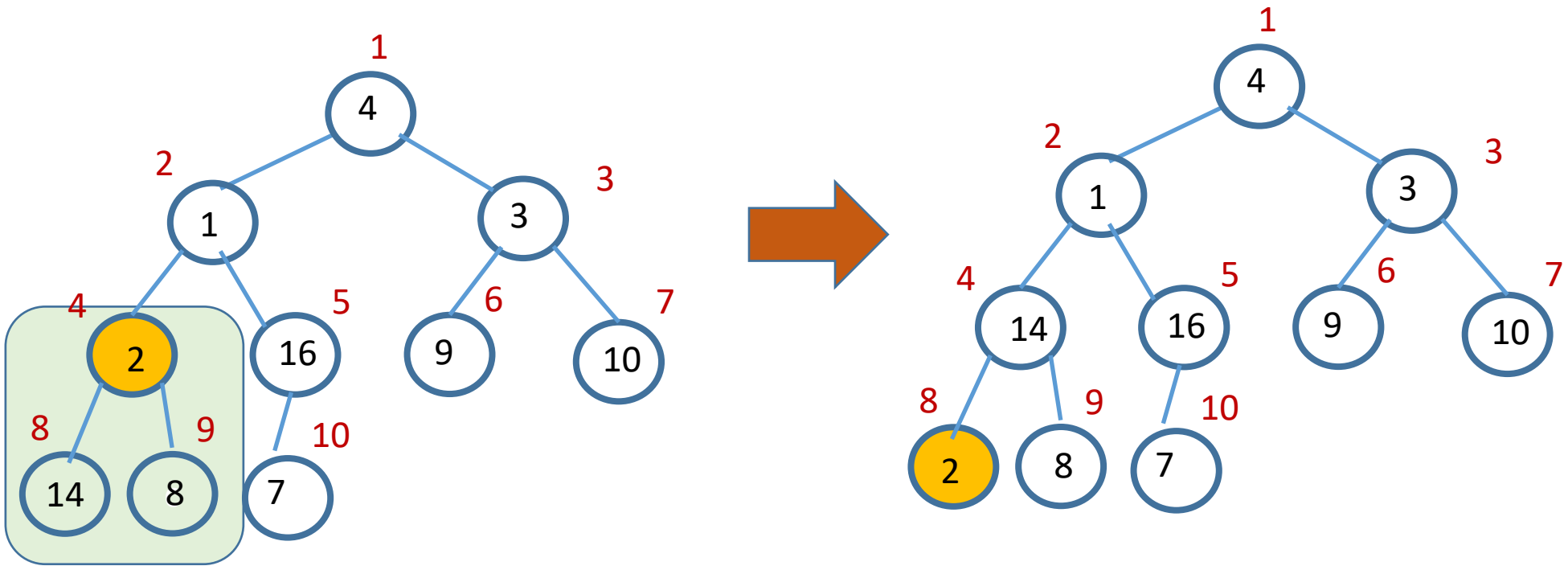
No change required

Building a Max-Heap

- Draw the binary tree obtained from the set of values in the following array, *A*:

0	1	2	3	4	5	6	7	8	9	10
	4	1	3	2	16	9	10	14	8	7

A

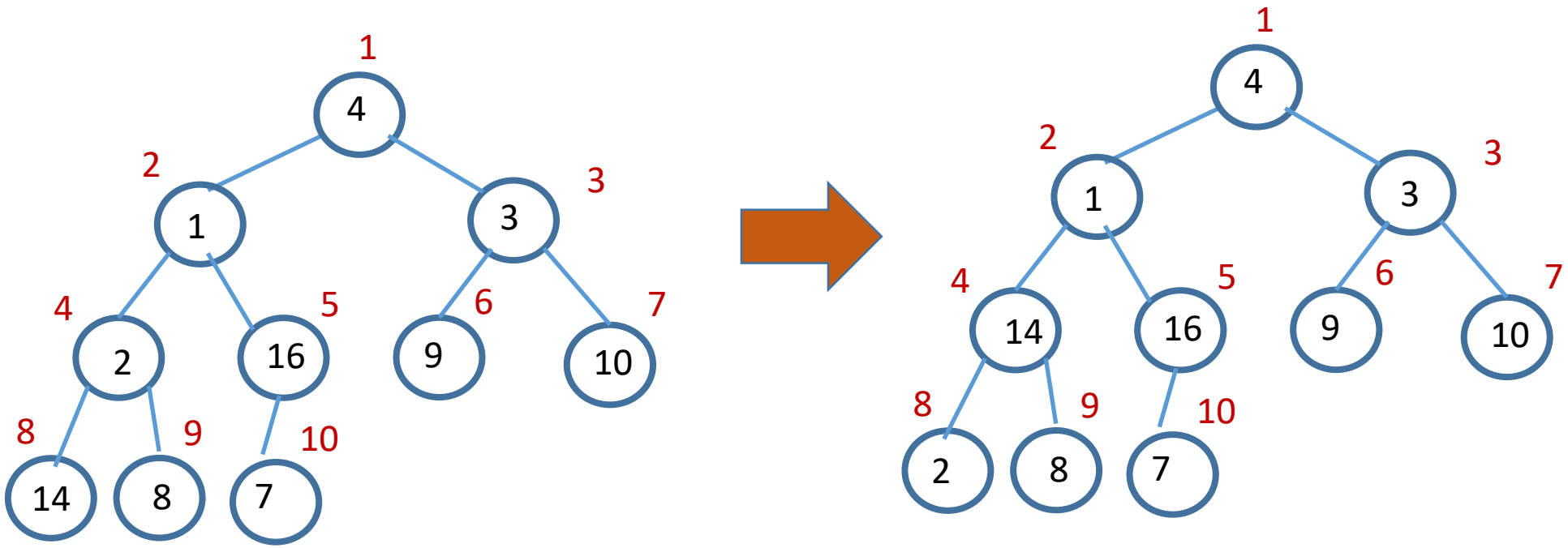


Building a Max-Heap

- Draw the binary tree obtained from the set of values in the following array, *A*:

0	1	2	3	4	5	6	7	8	9	10
	4	1	3	2	16	9	10	14	8	7

A

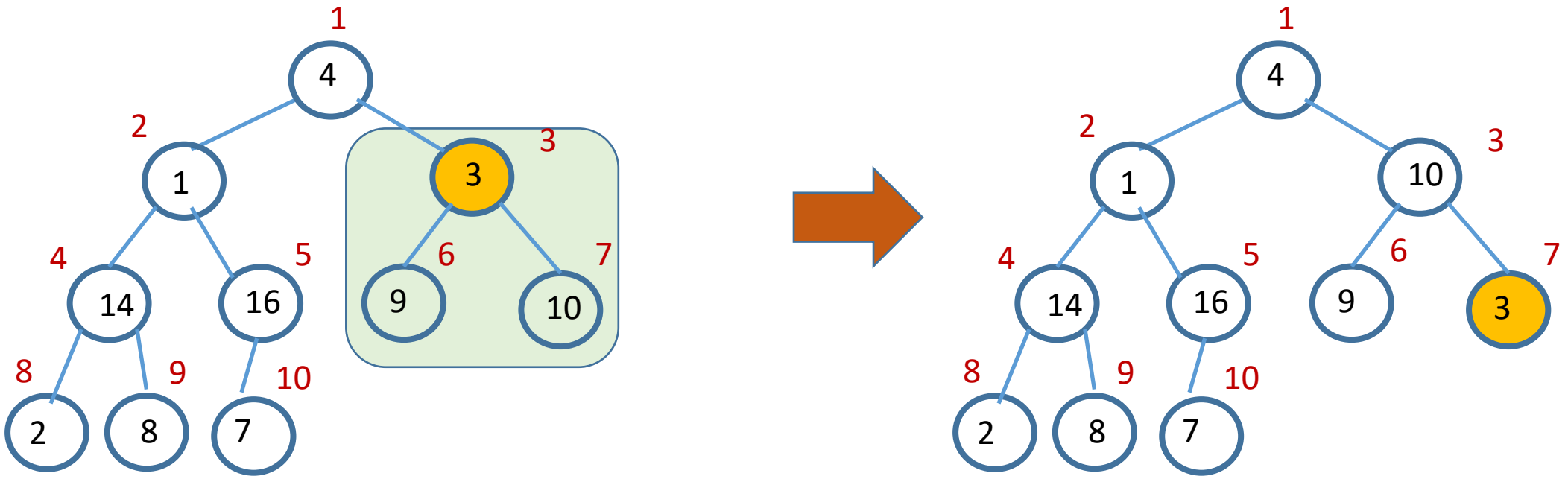


Building a Max-Heap

- Draw the binary tree obtained from the set of values in the following array, *A*:

0	1	2	3	4	5	6	7	8	9	10
	4	1	3	2	16	9	10	14	8	7

A

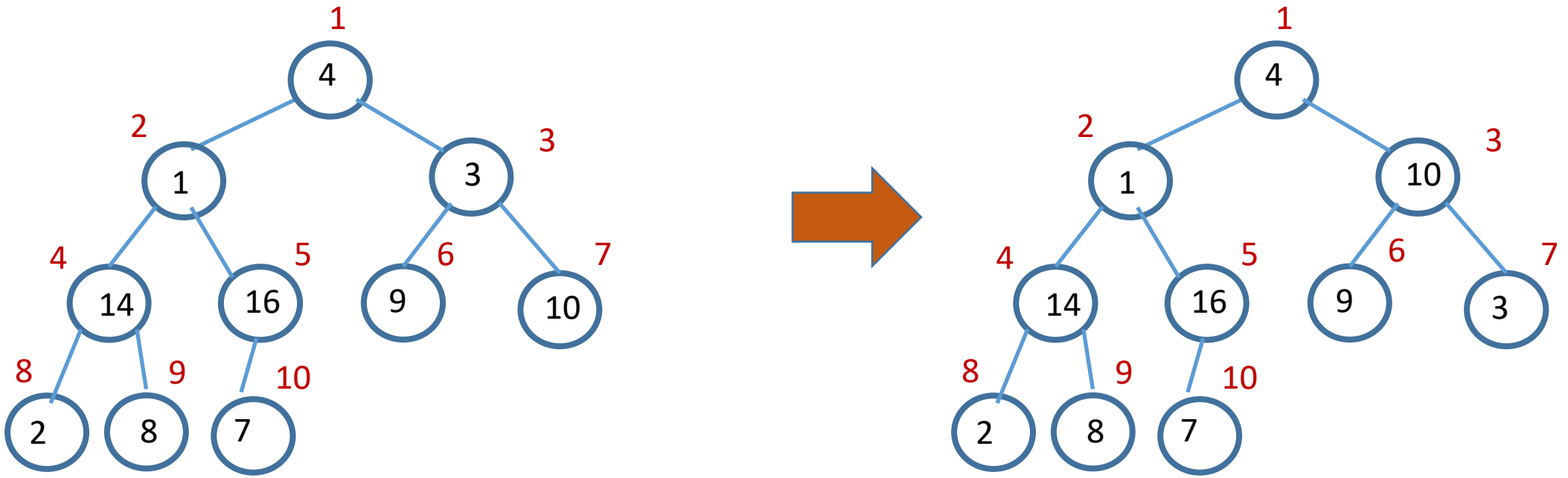


Building a Max-Heap

- Draw the binary tree obtained from the set of values in the following array, *A*:

0	1	2	3	4	5	6	7	8	9	10
	4	1	3	2	16	9	10	14	8	7

A

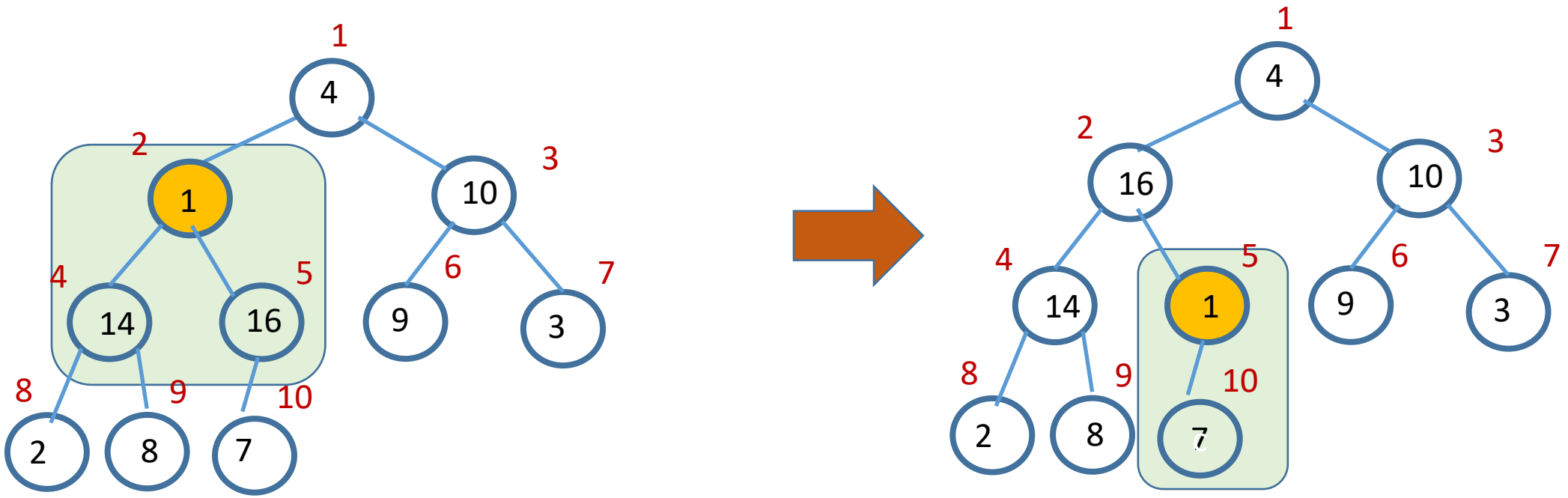


Building a Max-Heap

- Draw the binary tree obtained from the set of values in the following array, *A*:

0	1	2	3	4	5	6	7	8	9	10
	4	1	3	2	16	9	10	14	8	7

A

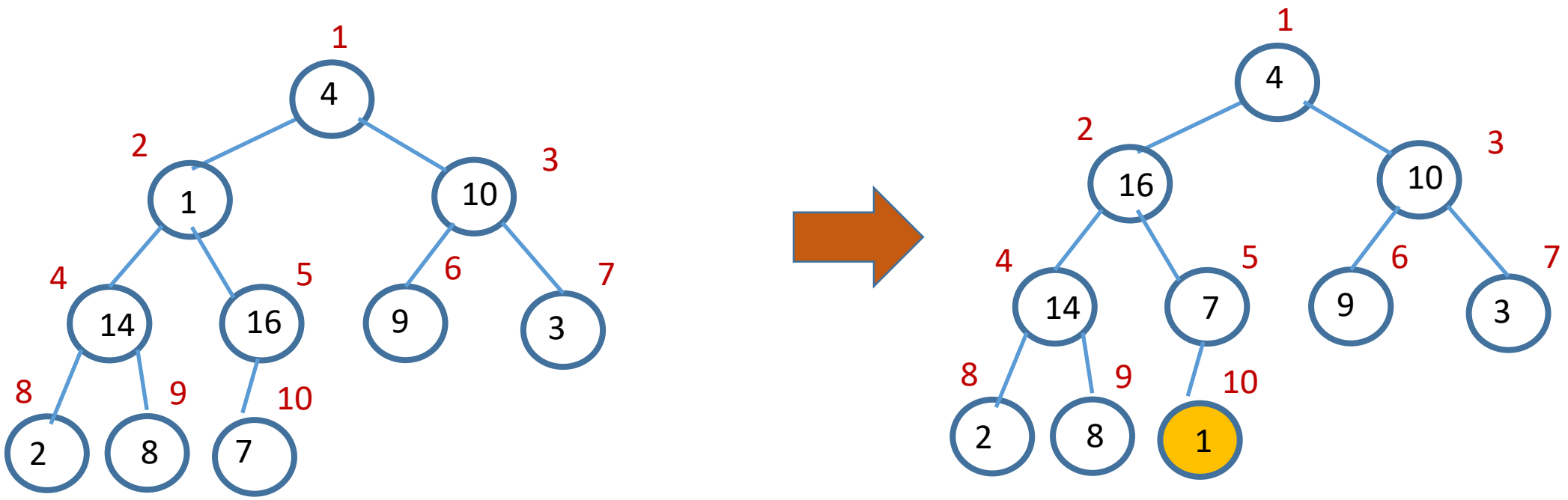


Building a Max-Heap

- Draw the binary tree obtained from the set of values in the following array, *A*:

0	1	2	3	4	5	6	7	8	9	10
	4	1	3	2	16	9	10	14	8	7

A

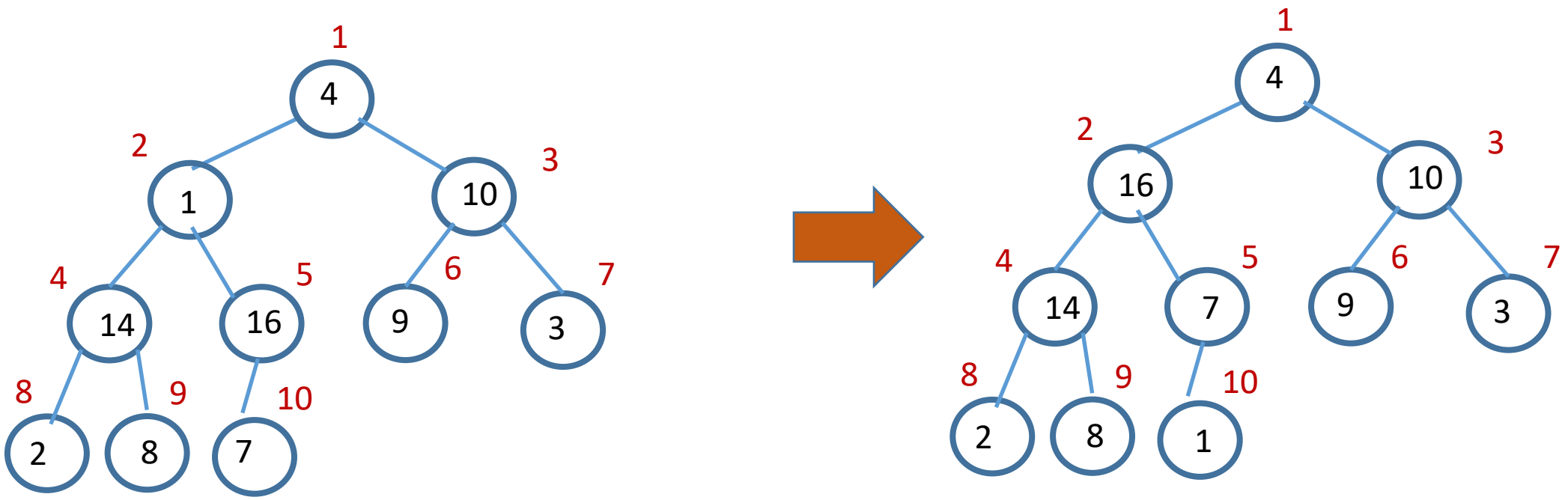


Building a Max-Heap

- Draw the binary tree obtained from the set of values in the following array, *A*:

0	1	2	3	4	5	6	7	8	9	10
	4	1	3	2	16	9	10	14	8	7

A

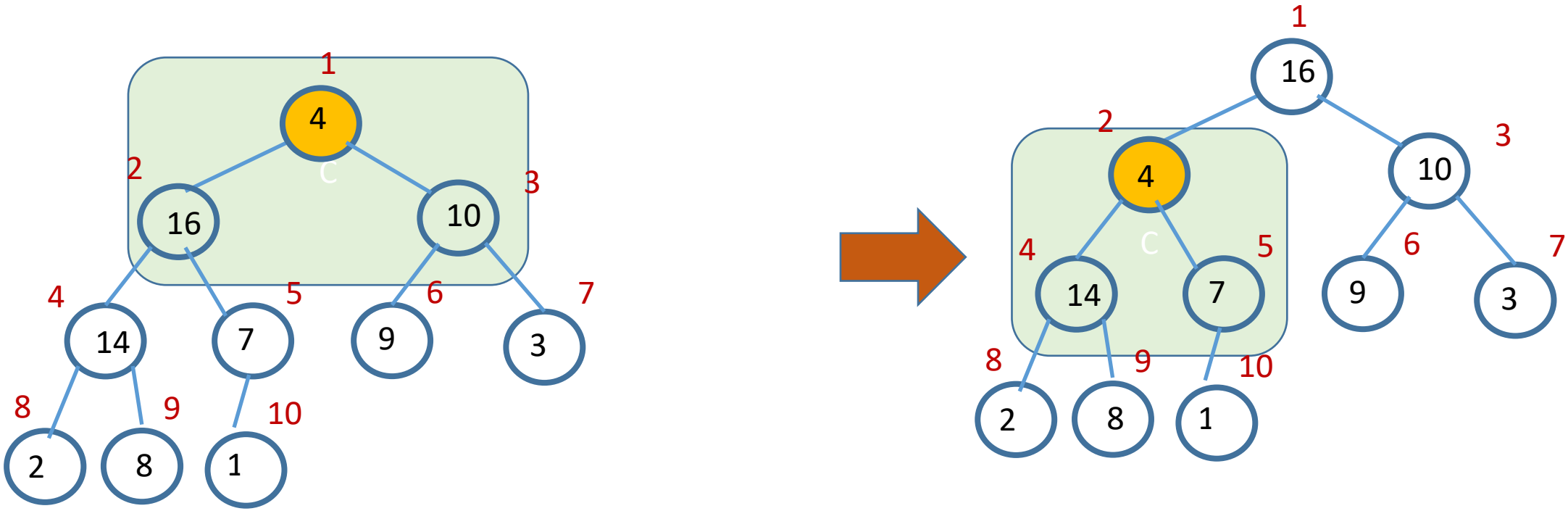


Building a Max-Heap

- Draw the binary tree obtained from the set of values in the following array, *A*:

0	1	2	3	4	5	6	7	8	9	10
	4	1	3	2	16	9	10	14	8	7

A

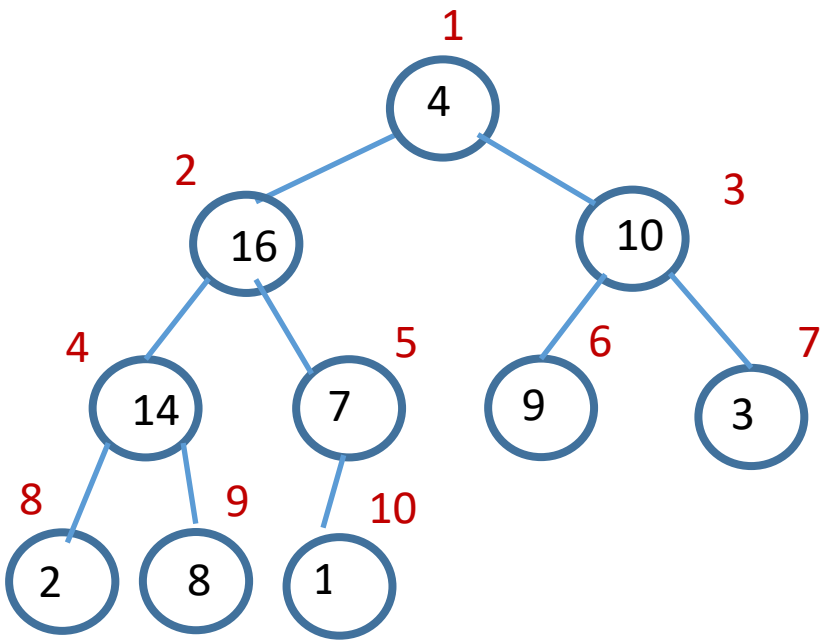


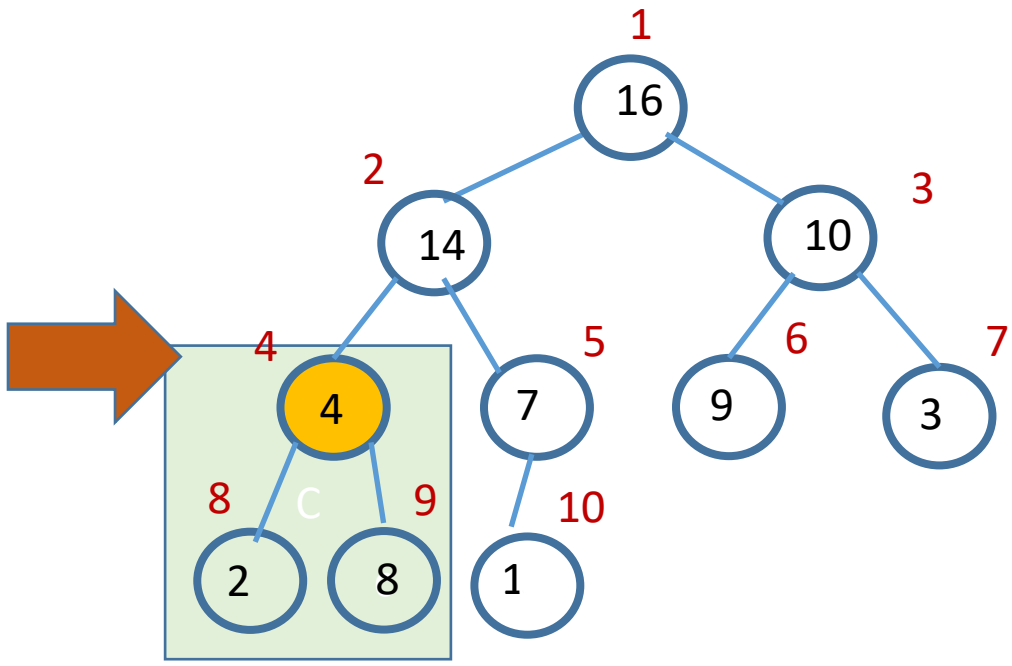
Building a Max-Heap

- Draw the binary tree obtained from the set of values in the following array, *A*:

0	1	2	3	4	5	6	7	8	9	10
	4	1	3	2	16	9	10	14	8	7

A



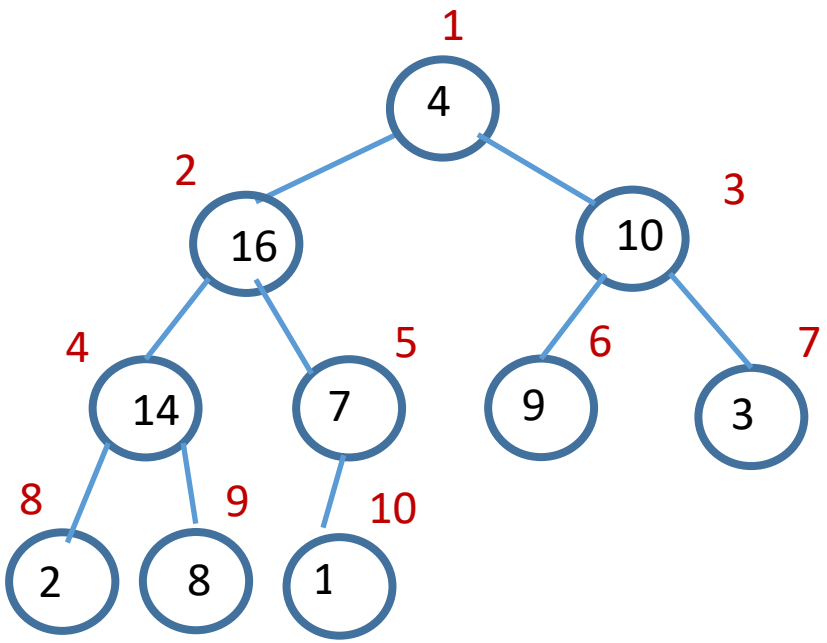


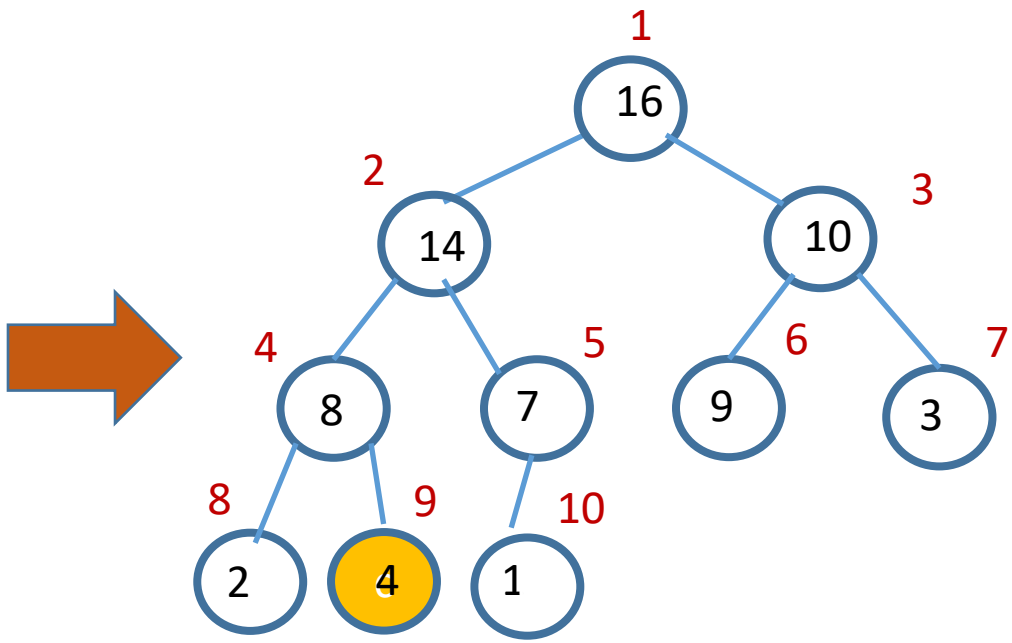
Building a Max-Heap

- Draw the binary tree obtained from the set of values in the following array, *A*:

0	1	2	3	4	5	6	7	8	9	10
	4	1	3	2	16	9	10	14	8	7

A



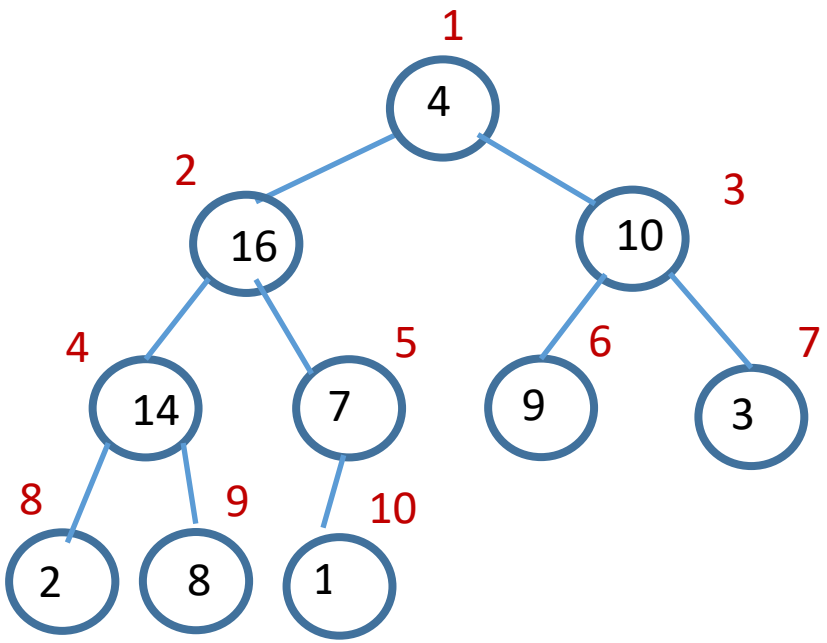


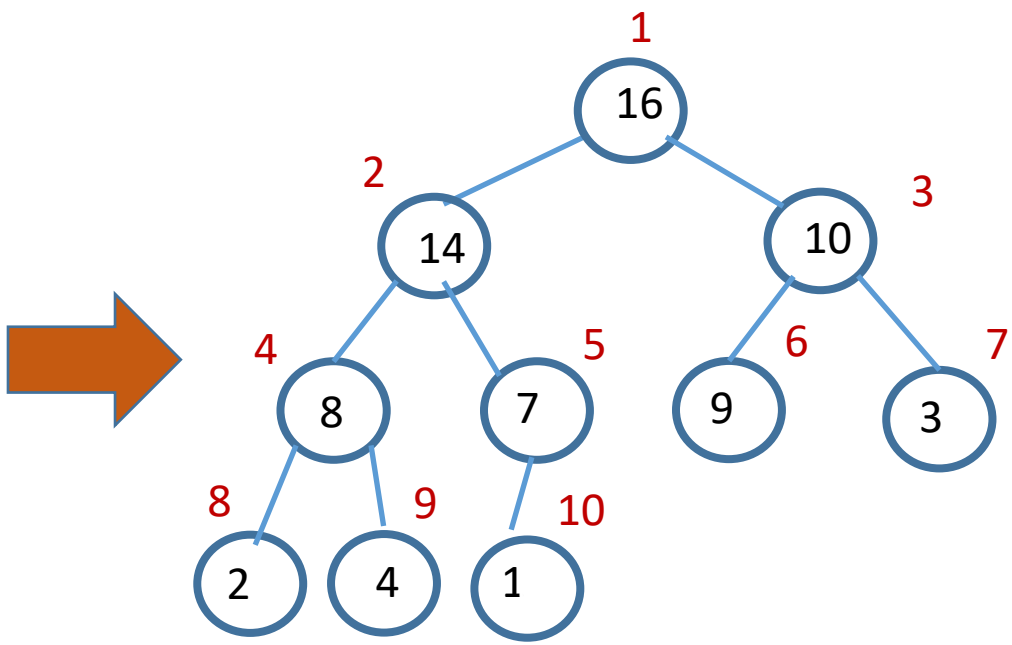
Building a Max-Heap

- Draw the binary tree obtained from the set of values in the following array, *A*:

0	1	2	3	4	5	6	7	8	9	10
	4	1	3	2	16	9	10	14	8	7

A





Function for Building a Max-Heap

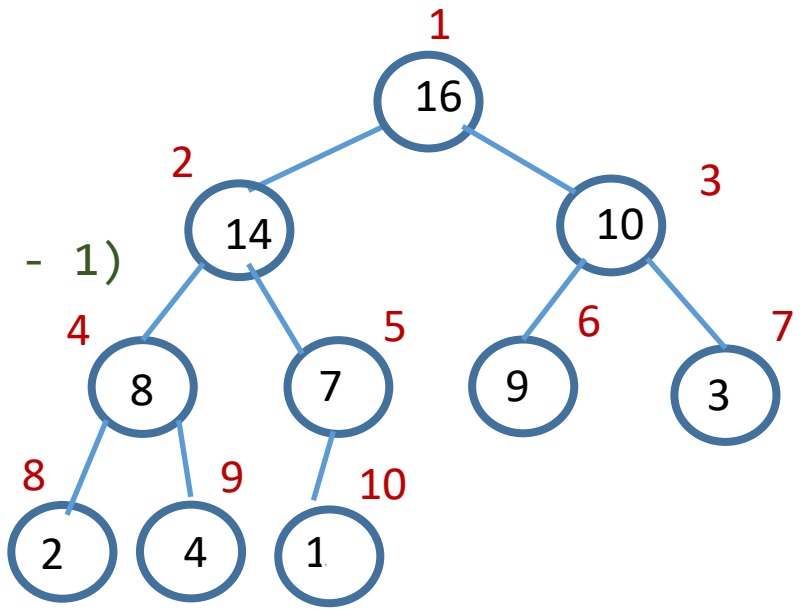
- Call *buildMaxHeap* (*A*, 10) where *A* contains the following values:

0	1	2	3	4	5	6	7	8	9	10
	4	1	3	2	16	9	10	14	8	7

A

- The following max-heap is produced:

```
void buildMaxHeap(MaxHeap * heap) {

    for (int i = (heap->size/2); i >= 1; i = i - 1)
            maxHeapify(heap, i);
}
```



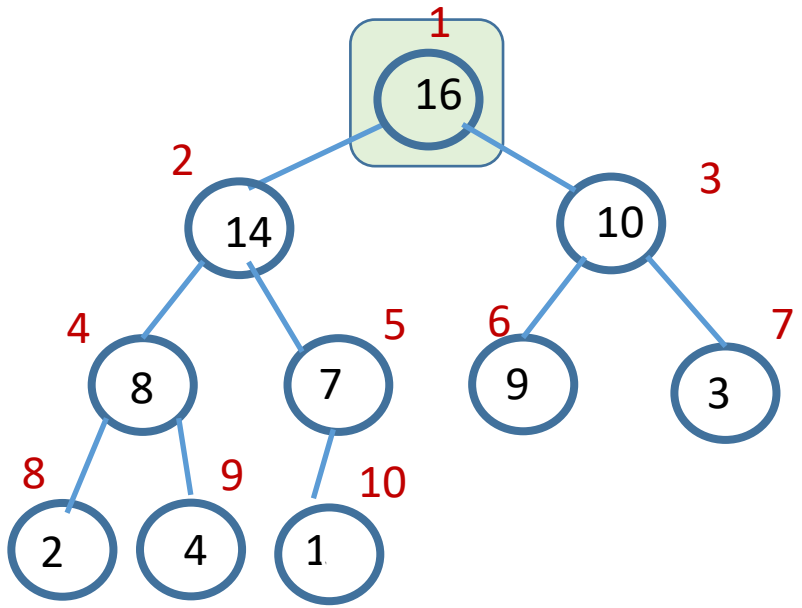
0	1	2	3	4	5	6	7	8	9	10
	16	14	10	8	7	9	3	2	4	1

Finding the Maximum Value in a Max-Heap

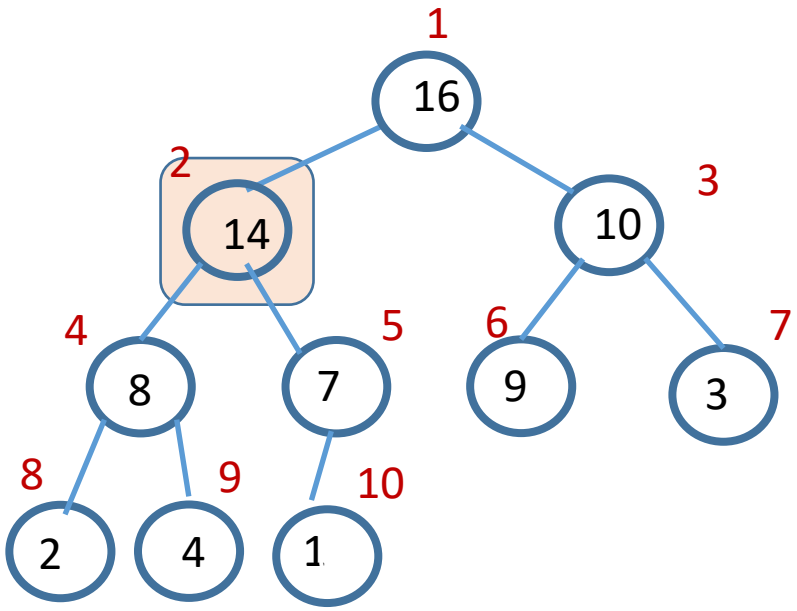
- Where is the maximum value stored in a max-heap?

0	1	2	3	4	5	6	7	8	9	10
	16	14	10	8	7	9	3	2	4	1

```
int maximum (MaxHeap * heap) {
   return heap->A[1];
}
```

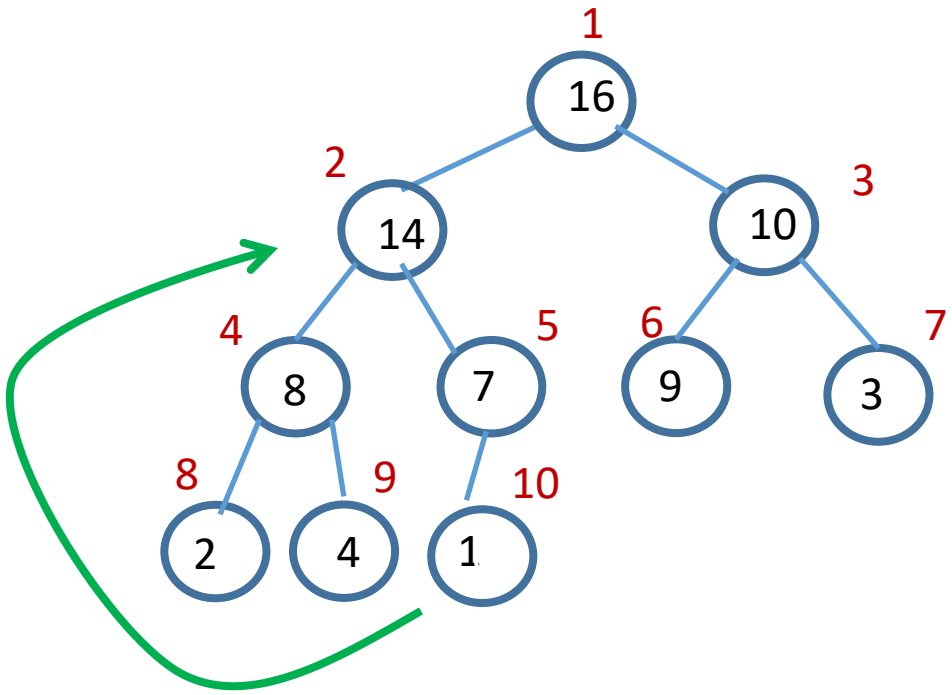


Deleting the Value at Index *i* in a Max-Heap



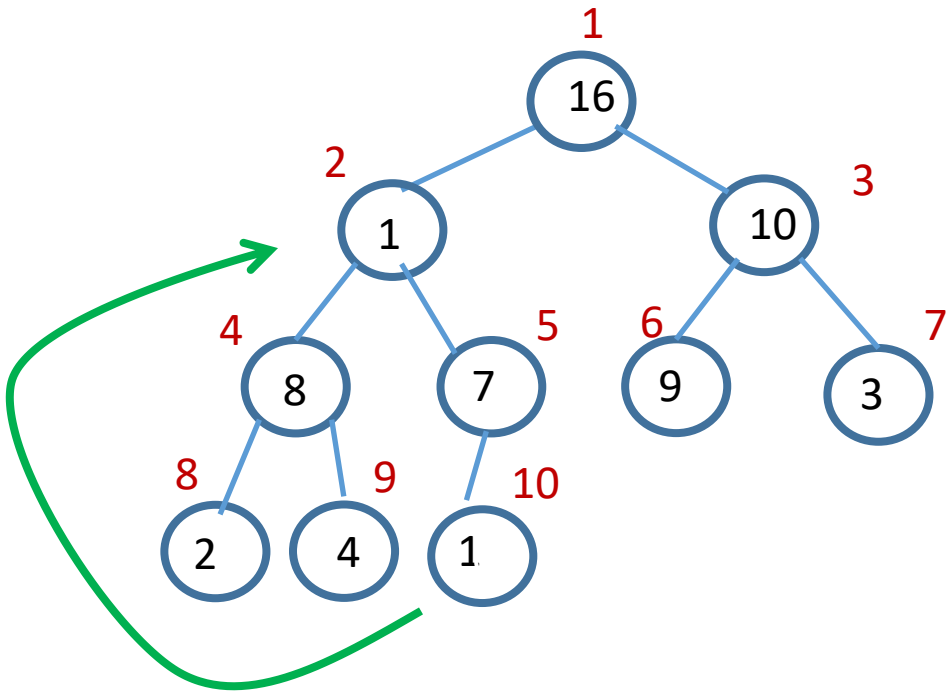
0	1	2	3	4	5	6	7	8	9	10
	16	14	10	8	7	9	3	2	4	1

Deleting the Value at Index *i* in a Max-Heap



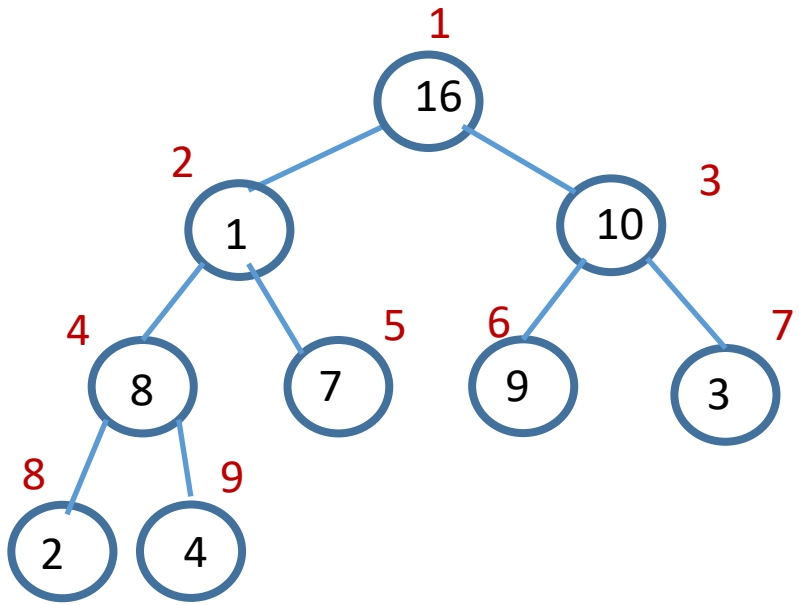
0	1	2	3	4	5	6	7	8	9	10
	16	14	10	8	7	9	3	2	4	1

Deleting the Value at Index *i* in a Max-Heap



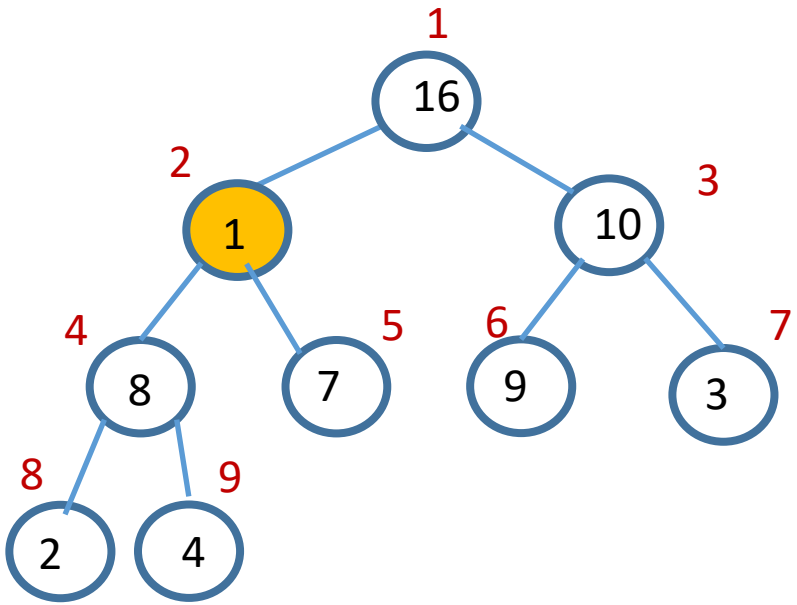
0	1	2	3	4	5	6	7	8	9	10
	16	1	10	8	7	9	3	2	4	1

Deleting the Value at Index *i* in a Max-Heap



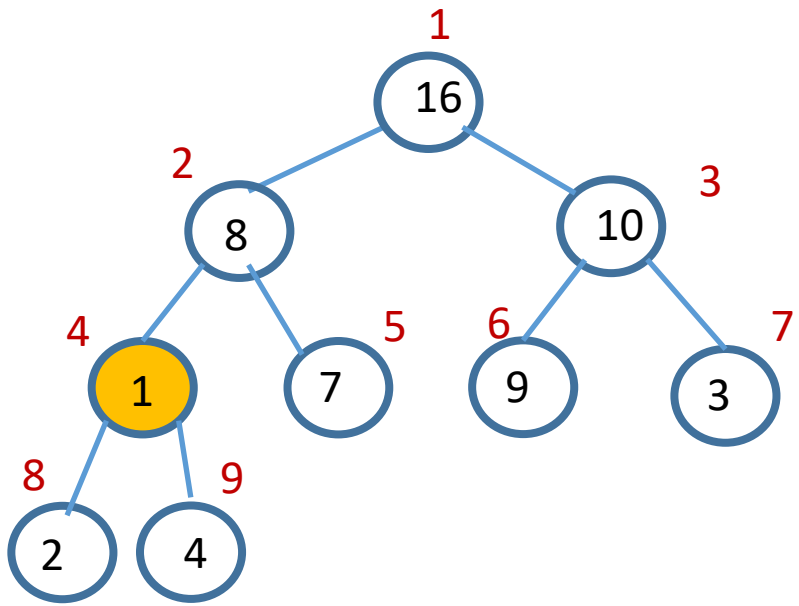
0	1	2	3	4	5	6	7	8	9
	16	1	10	8	7	9	3	2	4

Deleting the Value at Index *i* in a Max-Heap

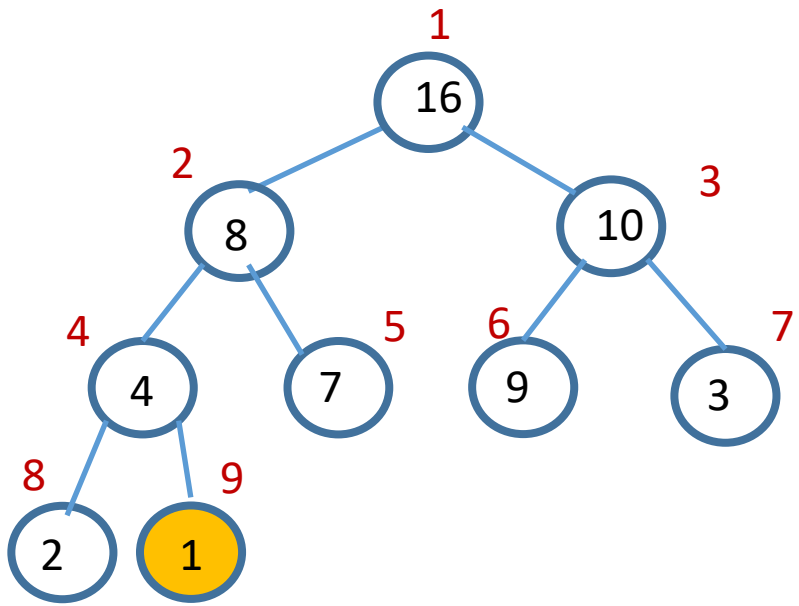


0	1	2	3	4	5	6	7	8	9
	16	1	10	8	7	9	3	2	4

Deleting the Value at Index *i* in a Max-Heap

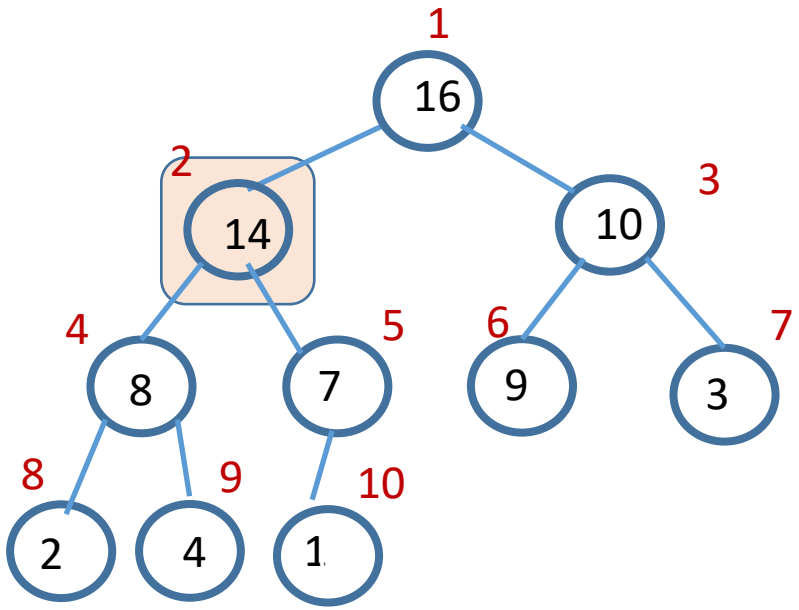


Deleting the Value at Index *i* in a Max-Heap



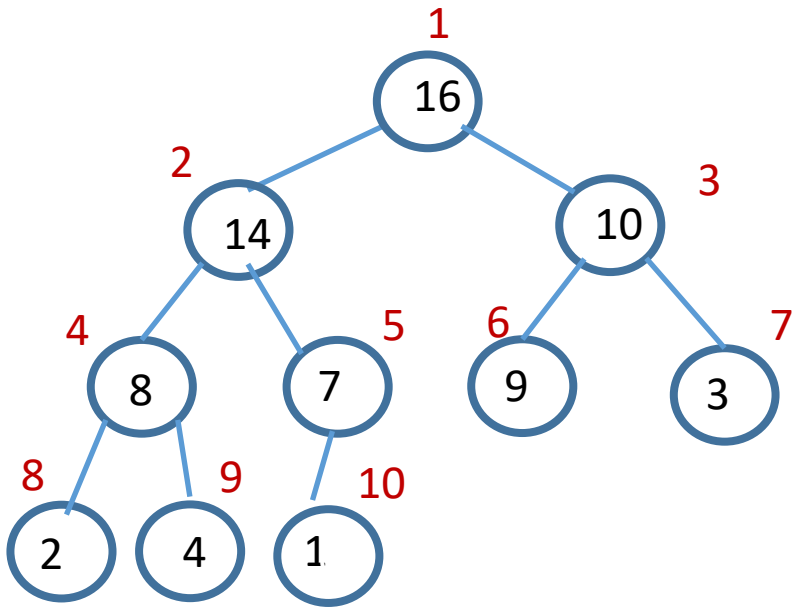
Deleting the Value at Index *i* in a Max-Heap

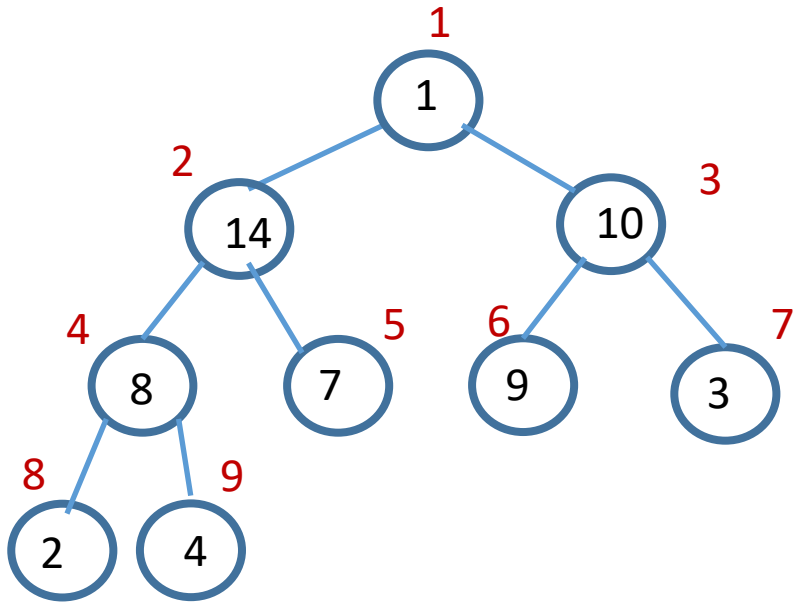
```
int deleteMaxHeap (MaxHeap * heap, int i) {

   int toDelete = heap->A[i];

   heap->A[i] = heap->A[heap->size];
   heap->size = heap->size - 1;
   maxHeapify (heap, i);

   return toDelete;

}
```



Deleting the Value at Index *1* in a Max-Heap

The max-heap below has 10 elements. Draw the max-heap after the biggest element (16) is deleted.



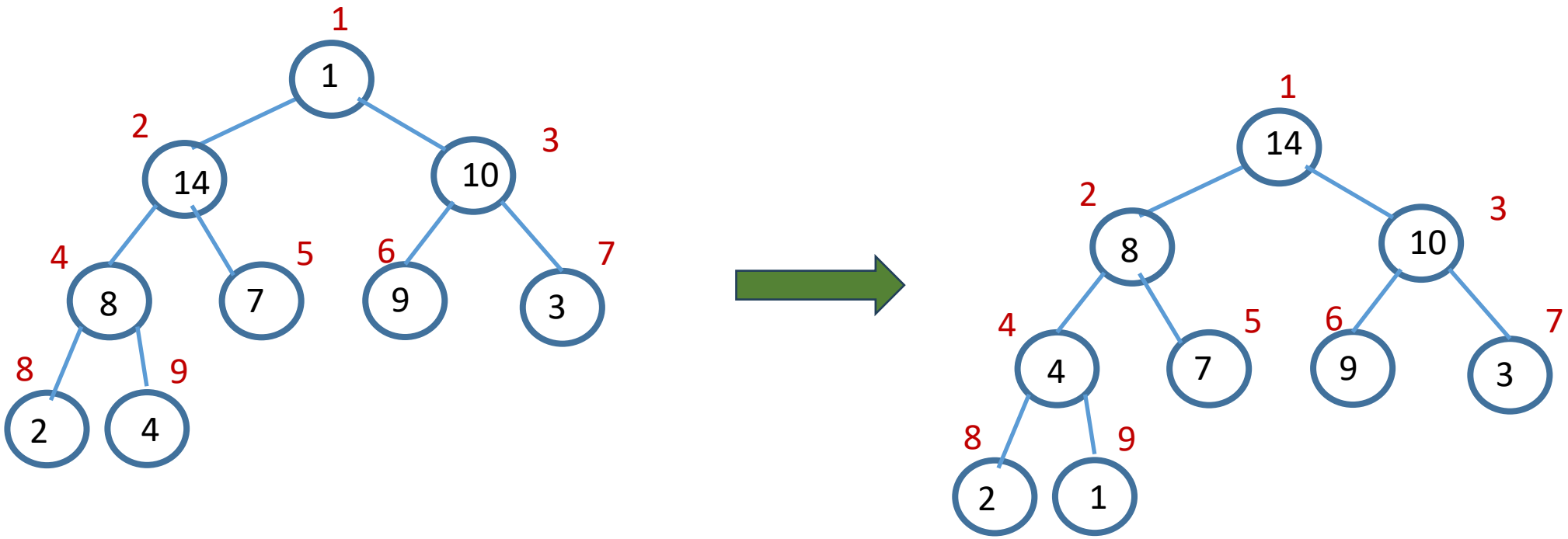


Copy the last element to location 1.

maxHeapify, starting from location 1.

Deleting the Value at Index *1* in a Max-Heap

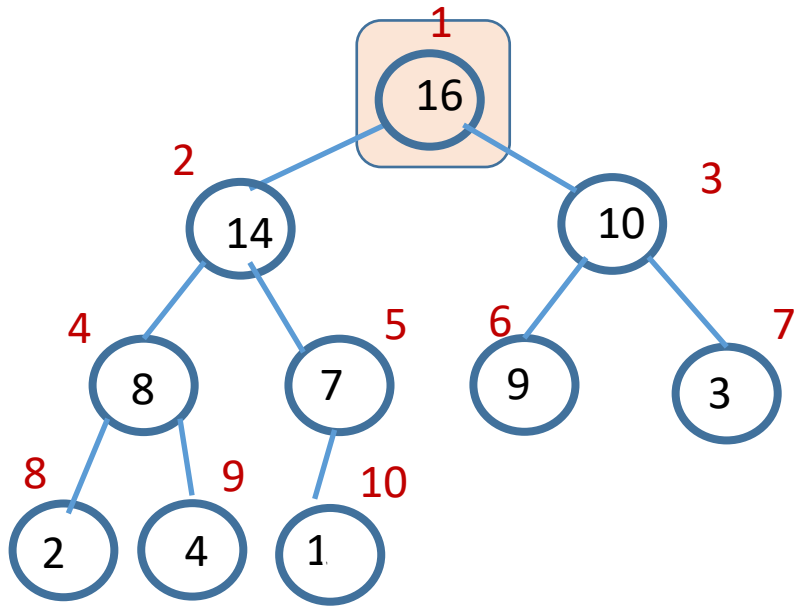
maxHeapify, starting from location 1:



What happens if we delete the next value at index 1?

Deleting the Values at Index *1* in a Max-Heap

```
void deleteAllMaxHeap (MaxHeap * heap) {

   int deleted;

   for (int i=1; i<=heap->size; i++) {
        deleted = deleteMaxHeap (heap, 1);
        cout << deleted << endl;
   }
}
```



What is the output produced if *deleteAllMaxHeap* is called with the above max-heap?