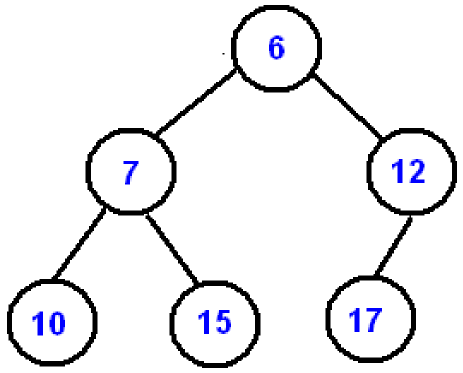
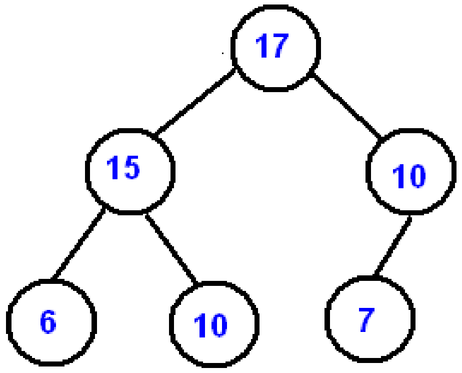
Binary Heap Tree

Binary Heap Tree properties

- A Binary Heap is a Binary Tree with following properties:
- 1) It’s a complete tree (All levels are completely filled except possibly the last level and the last level has all keys as left as possible). This property of Binary Heap makes them suitable to be stored in an array.
- 2) A Binary Heap is either Min Heap or Max Heap. In a Min Binary Heap, the key at root must be minimum among all keys present in Binary Heap. The same property must be recursively true for all nodes in Binary Tree. In a Max Binary Heap, the key at root must be maximum among all keys present in Binary Heap.





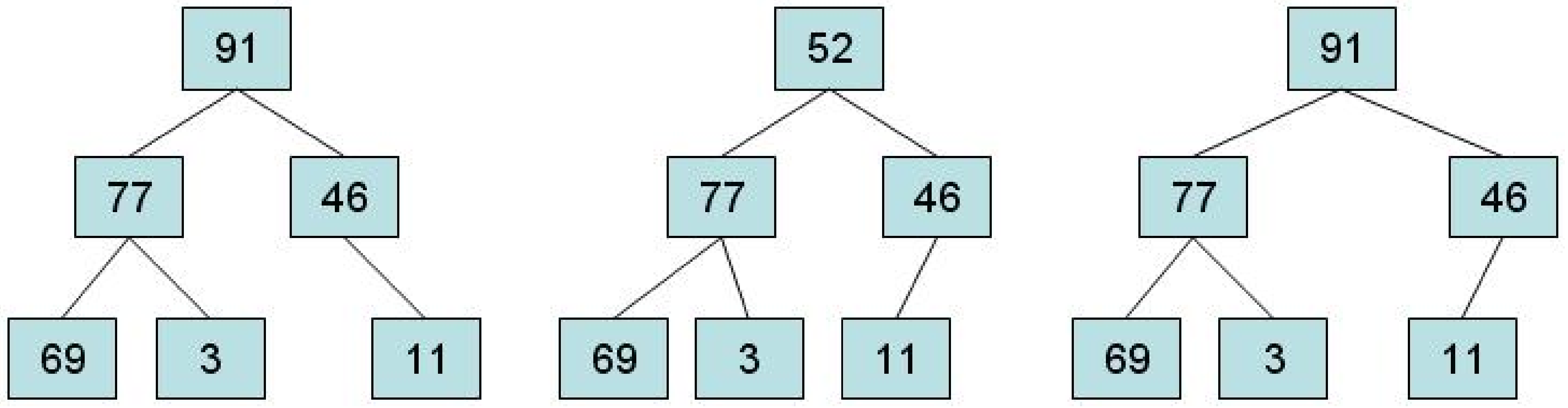
Source: https://www.geeksforgeeks.org/binary-heap/

Binary Heap Tree properties

- In a heap the highest (or lowest) priority element is always stored at the root, hence the name "heap".
- A heap is not a sorted structure and can be regarded as partially ordered.
- Since a heap is a complete binary tree, it has a smallest possible height - a heap with N nodes always has O(log N) height.
- A heap is useful data structure when you need to remove the object with the highest (or lowest) priority. A common use of a heap is to implement a priority queue.

Binary Heap Tree properties

Example: which one is a heap?



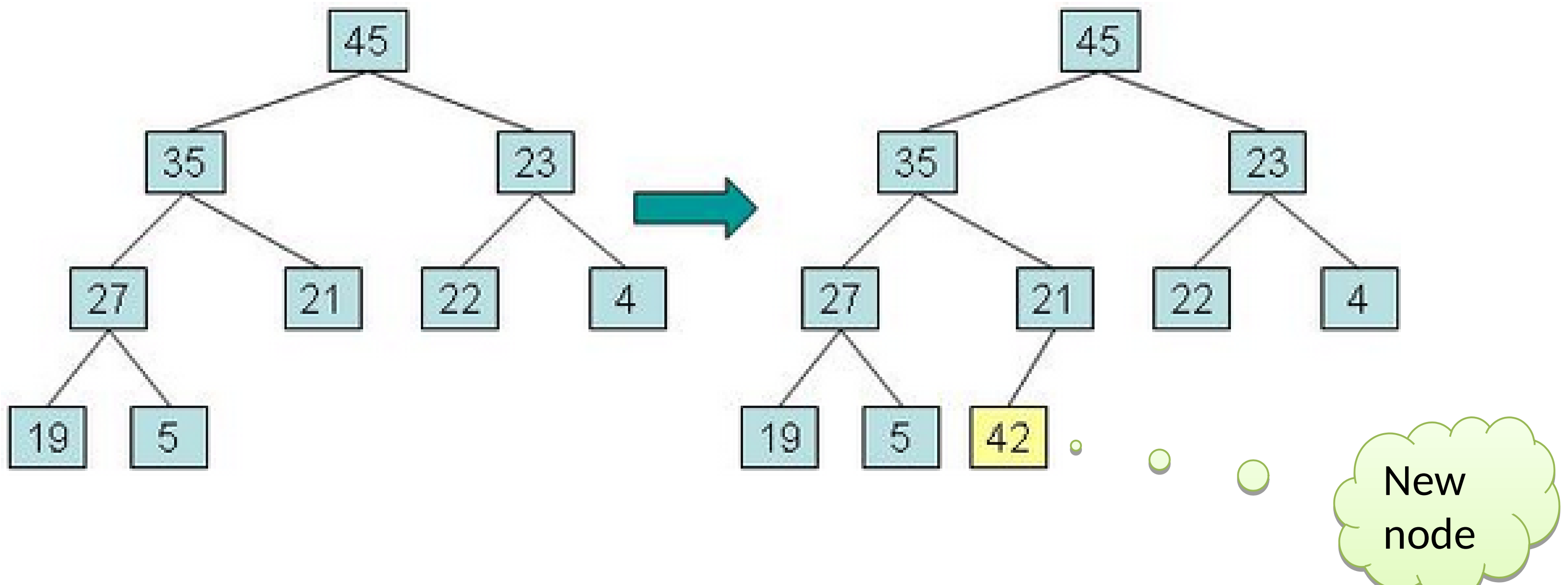
Source: https://www.cpp.edu/~ftang/courses/CS241/notes/heap.htm

Adding an element to heap

- Pseudocode for Adding an Element:
- 1-Place the new element in the heap in the first available location. This keeps the structure as a complete binary tree, but it might no longer be a heap since the new element might have a greater value than its parent.
- 2-Reheapification upward :

```
    while (the new element has a greater value than its parent)
                swap the new element with its parent.
```

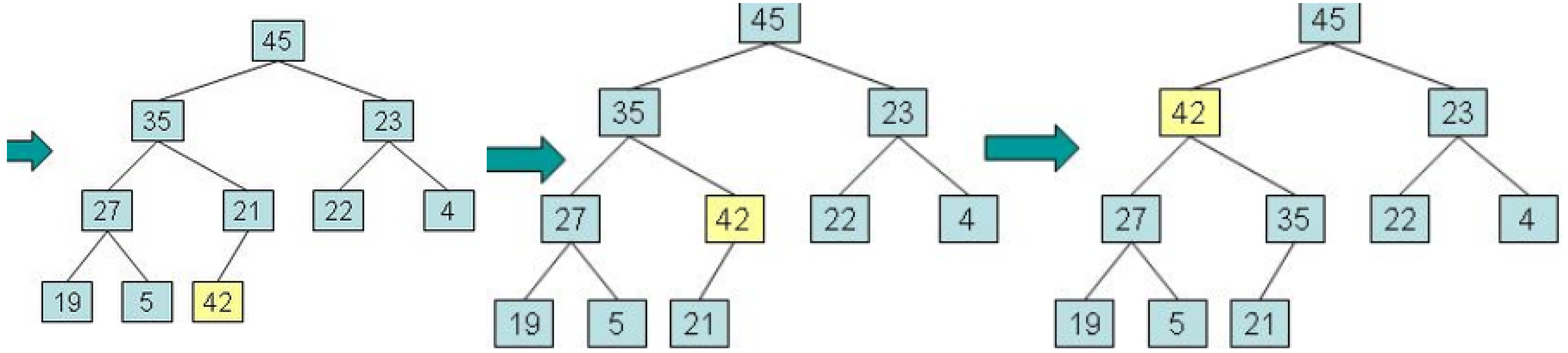
- Notice that Step 2 will stop when the new element reaches the root or when the new element's parent has a value greater than or equal to the new element's value.
- **Example:** We want to insert a node with value 42 to the heap on the left.



Adding an element to heap

- Reheapification upward

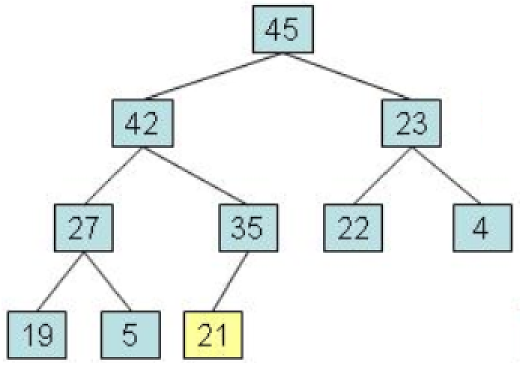
The new binary tree satisfies heap properties

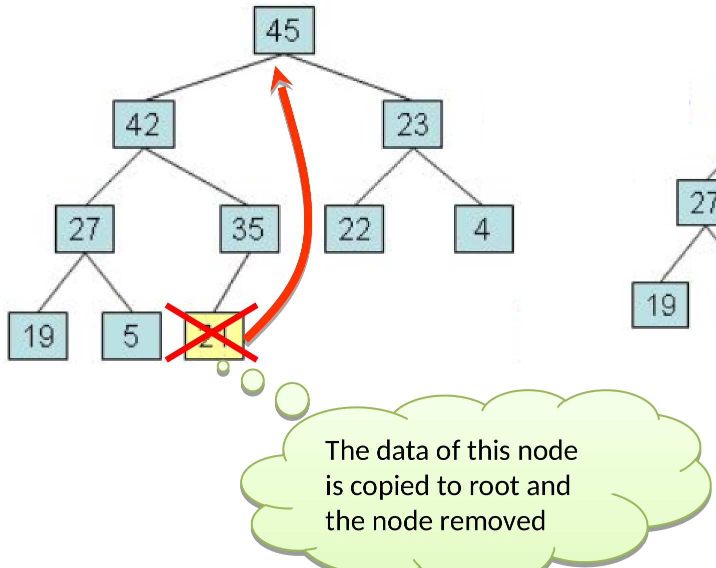


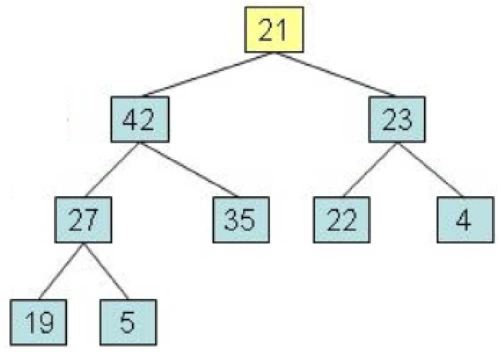
Removing the Root of a Heap

- Psuedocode for Removing the Root:

```
1. Copy the element at the root of the heap to the variable used to return a value.
2. Copy the last element in the deepest level to the root and then take this last node out of the tree. This
   element is called the "out-of-place" element.
3. while (the out-of-place element has a value that is lower than one of its children)
           swap the out-of-place element with its greatest-value child.
4. Return the answer that was saved in Step 1.
5. Notice that Step 3 will stop when the out-of-place element reaches a leaf or it has a value that is
   greater or equal to all its children.
```

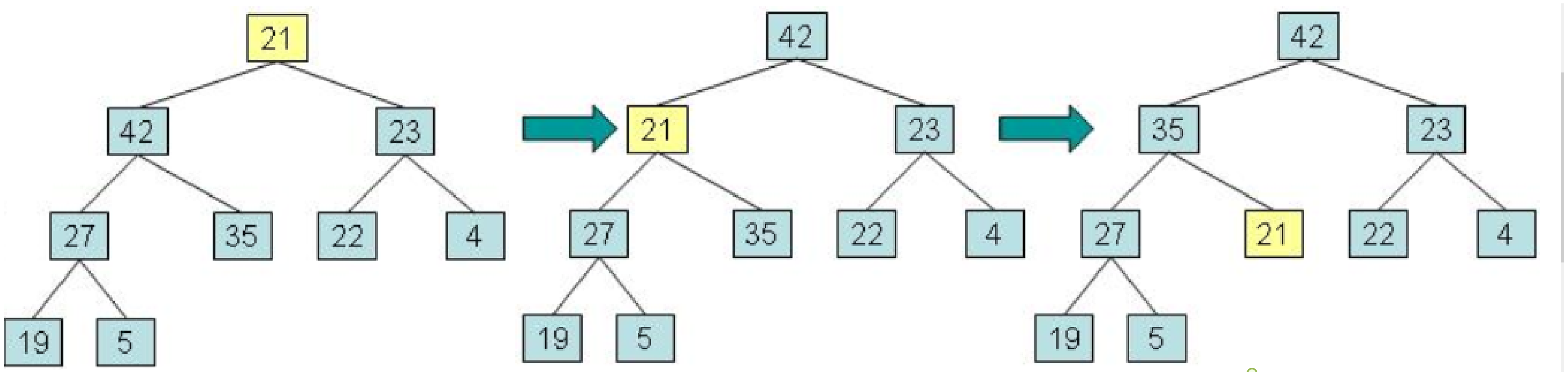



45
42
23
27
35
22
4
19
5
21
The data of this node is copied to root and the node removed



Removing the Root of a Heap

- Reheapification downward:

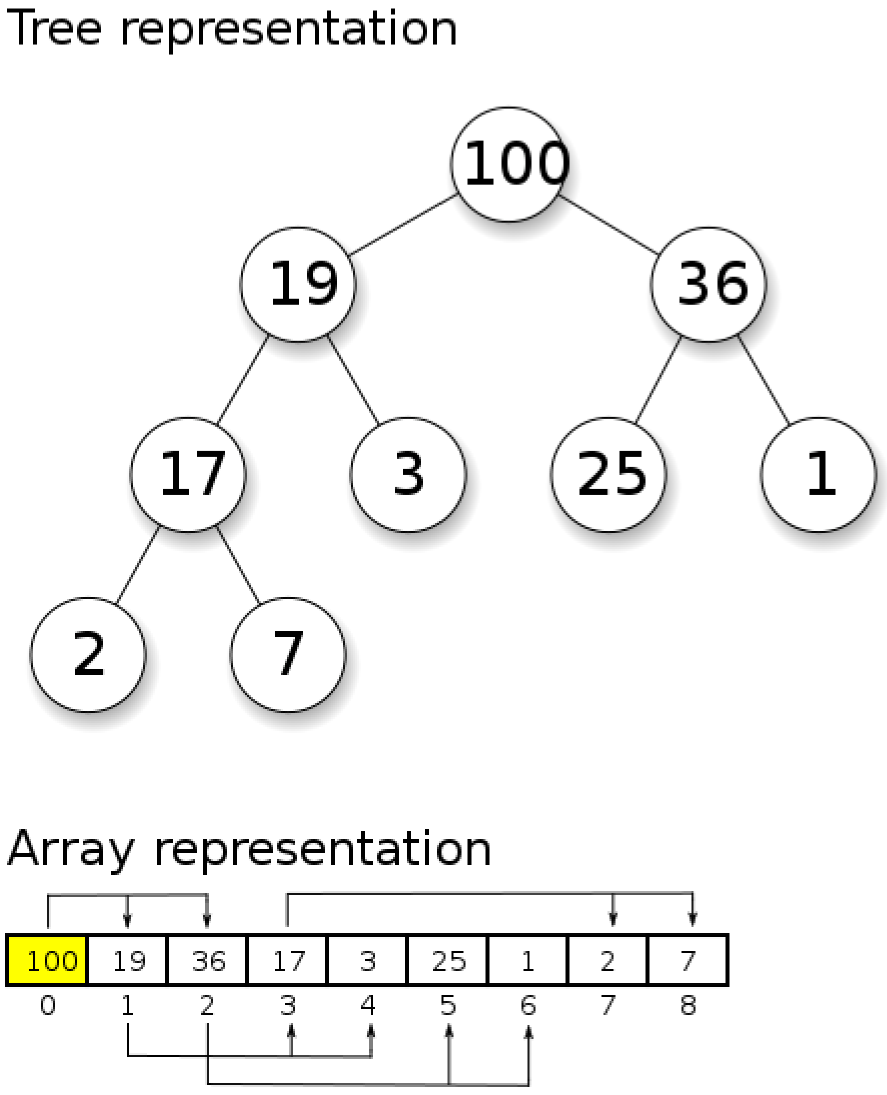


Following reheapification downward, the new tree is a binary heap.

Array Implementation

- A Binary Heap is a Complete Binary Tree. A binary heap is typically represented as an array.

- The root element: Arr[0]
- Parent node : Arr[**(i-1)/2**]
- Left child node: Arr[**(2*i)+1**]
- Right child node: Arr[**(2*i)+2**]


Tree representation
100
19
36
17
3
25
1
2
7
Array representation
100 19 36 17 3 25 1 2 7
0 1 2 3 4 5 6 7 8