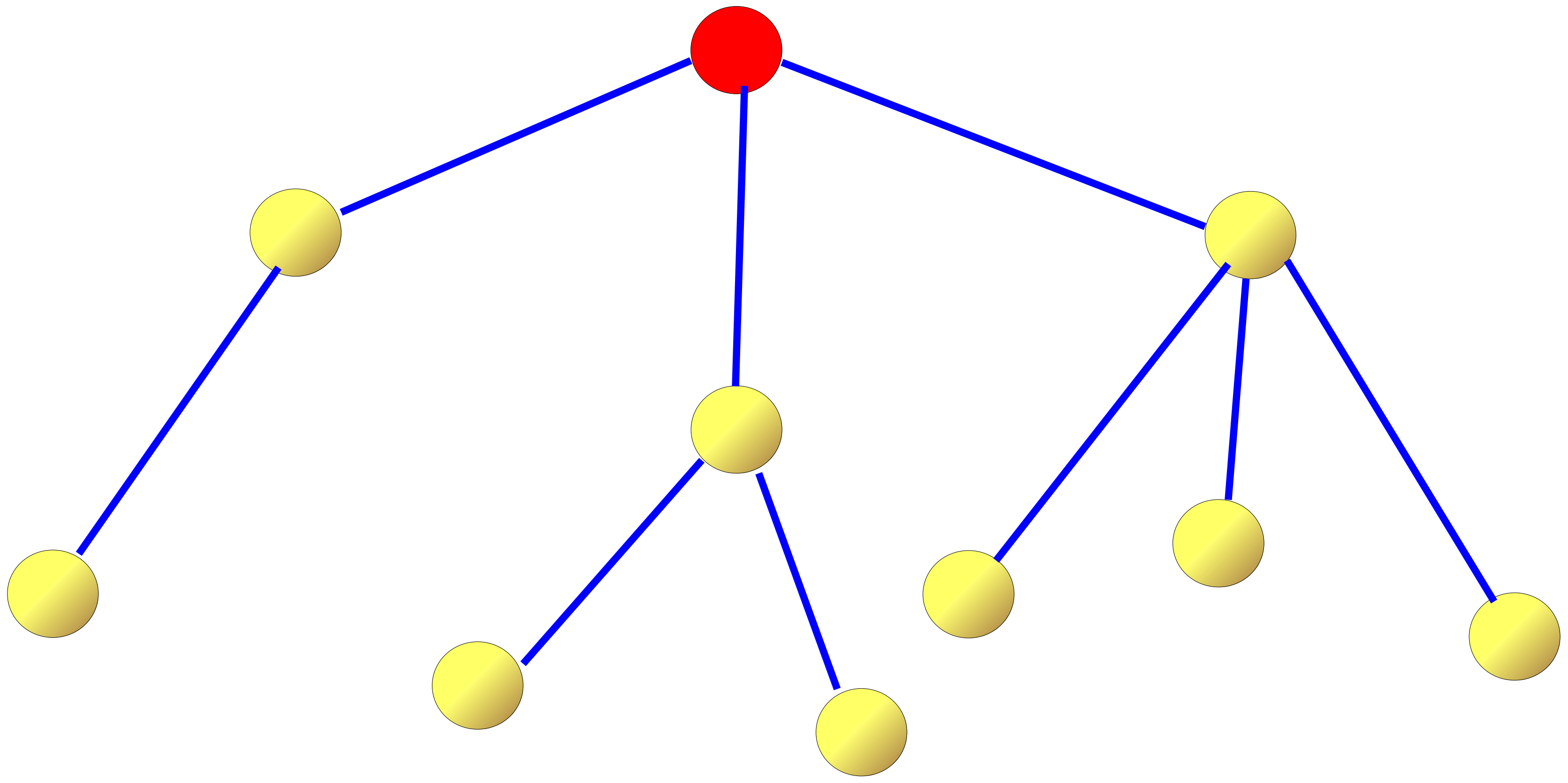
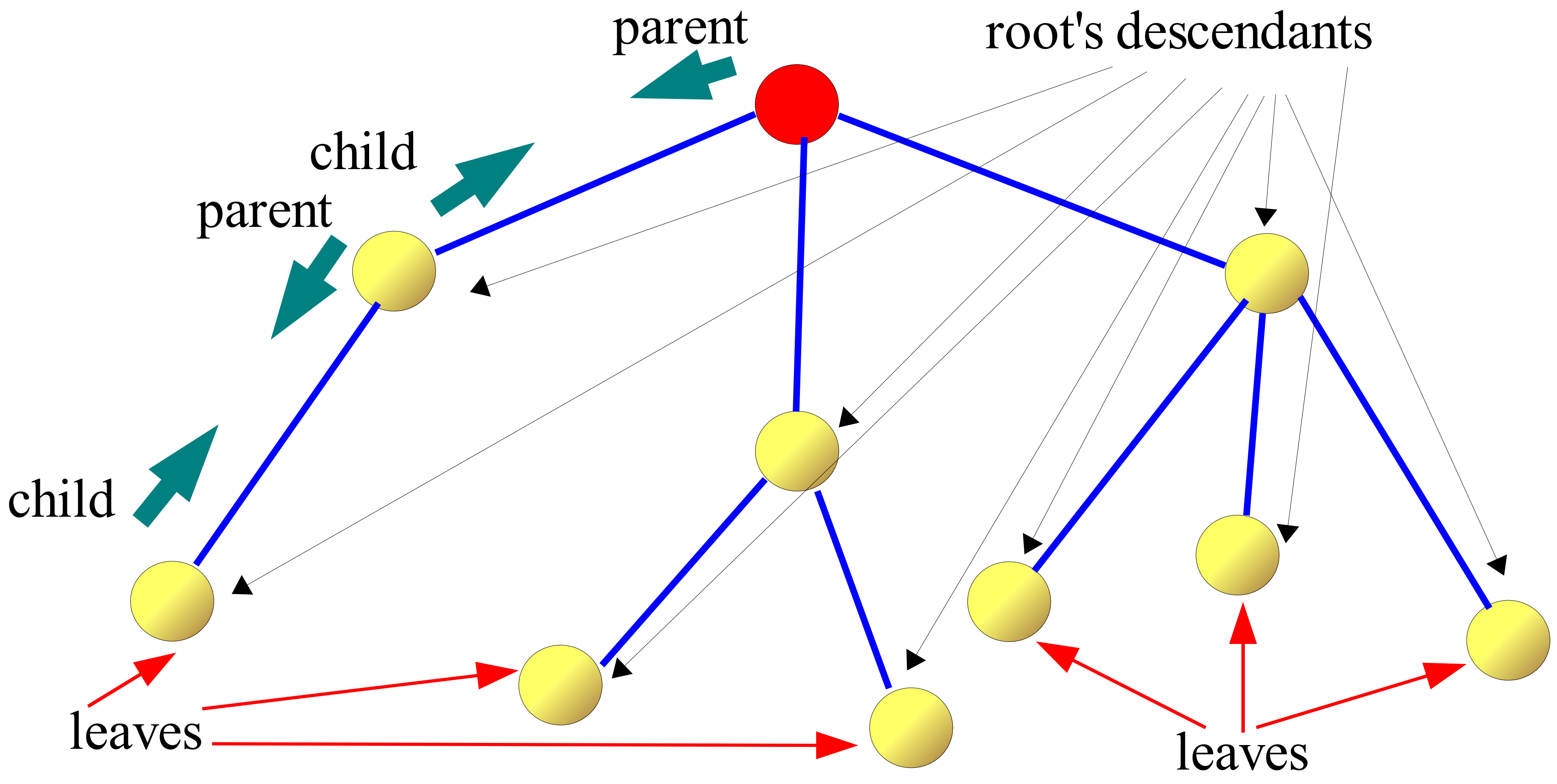
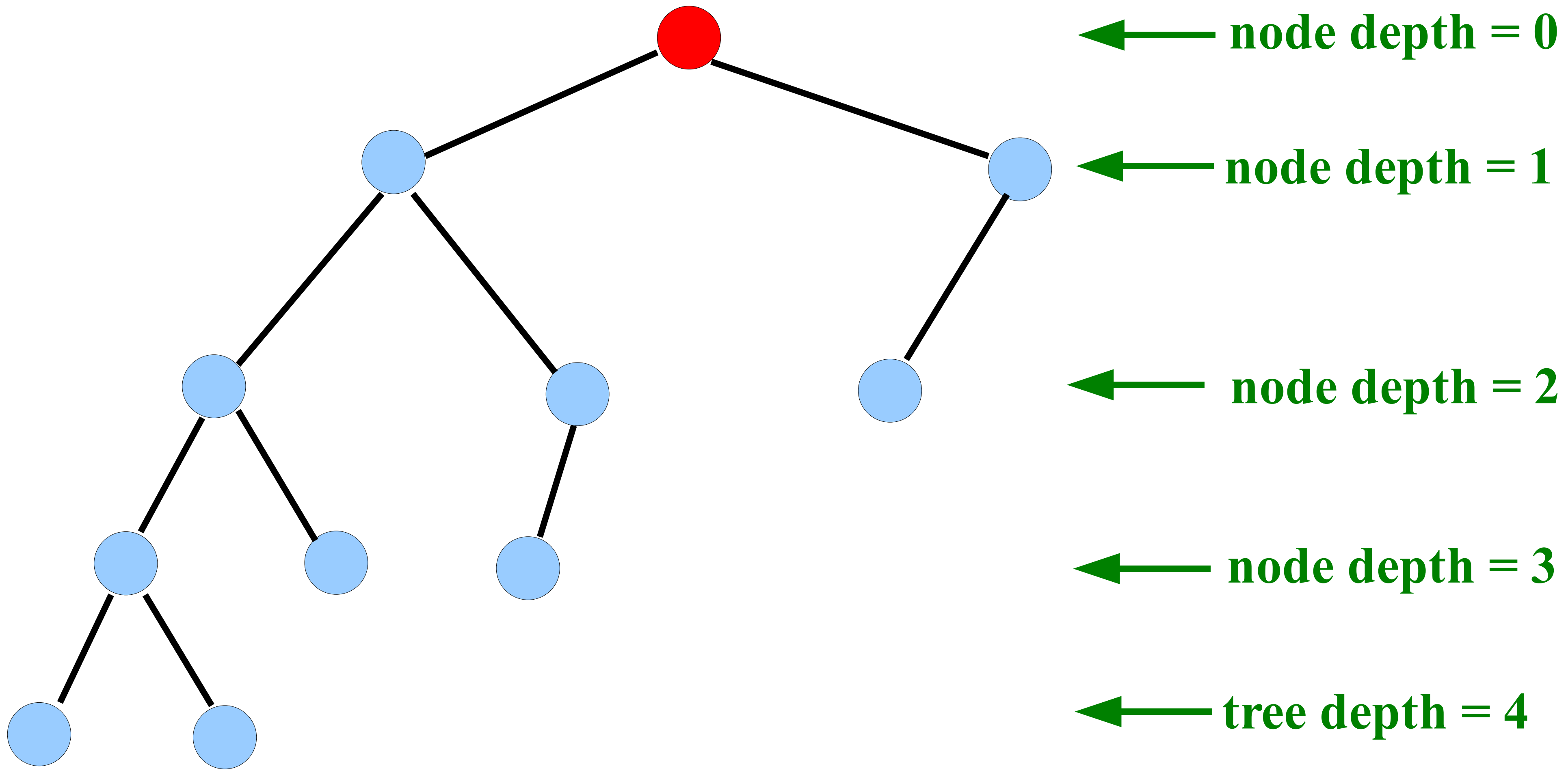
A particular node of a tree may be specified as its root. Such a tree is typically drawn with its root at the top. A tree with a root specified is called a **rooted tree**.



The edges of a rooted tree are often treated as directed. Every non-root node has exactly one edge that leads to the root. Descendant of a node in a directed tree is defined as any other node reachable from that node.

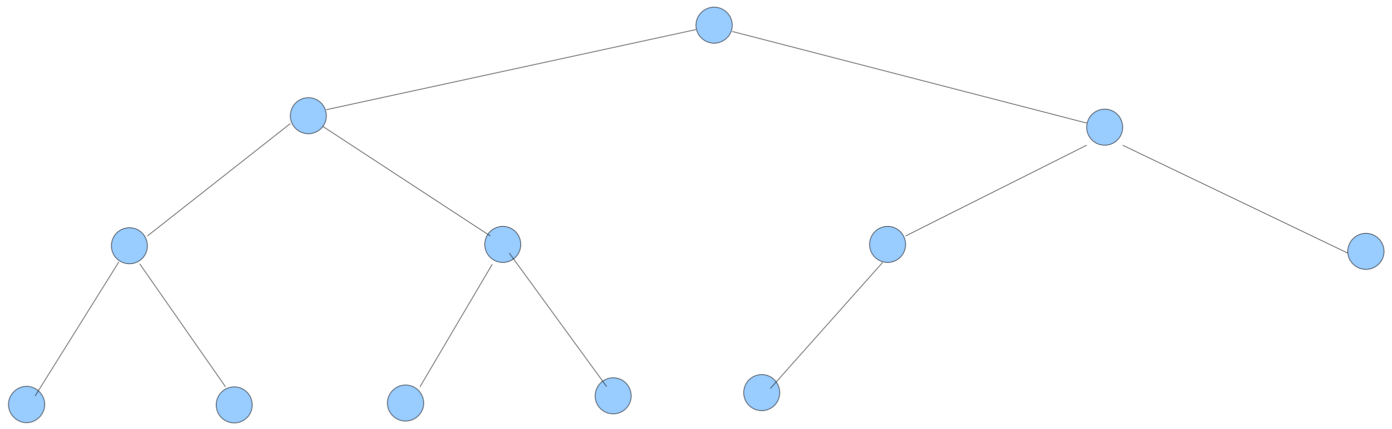


A **binary tree** is a rooted tree with all its nodes having two or fewer children. The **depth** of a binary tree is a number indicating how many nodes are traversed from the root to a leaf:



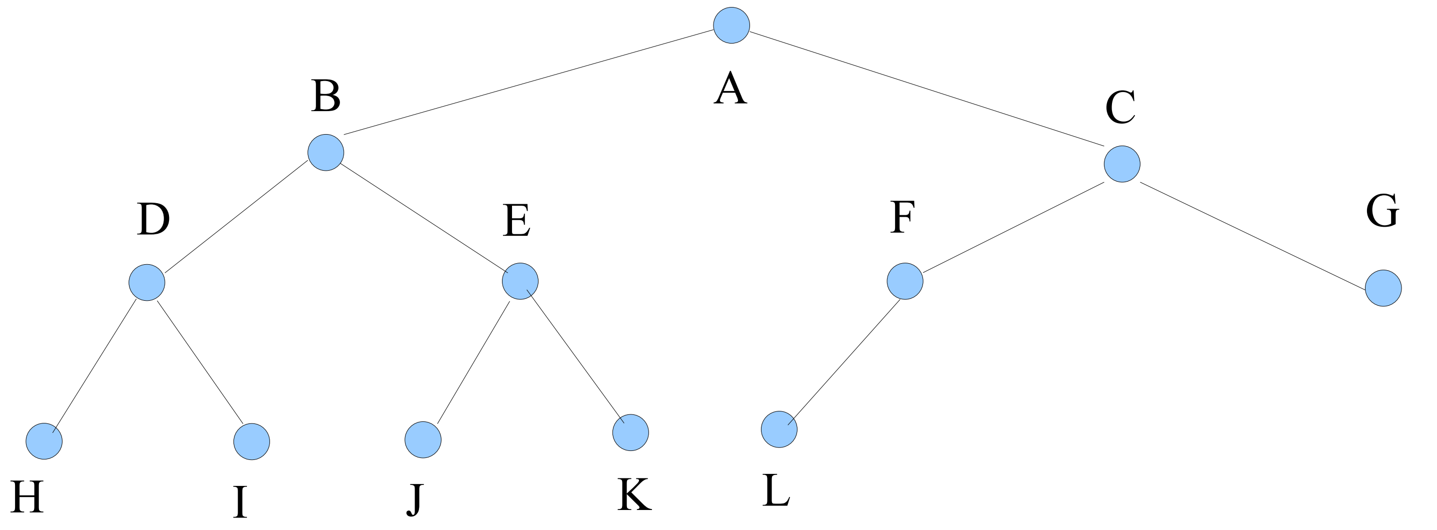
In a **balanced binary tree,** the depths of all the leaves differ by at most 1.

Balanced binary trees have a predictable depth equal to the integer part of $\log_2 n$ where n is the number of nodes of the binary balanced tree.



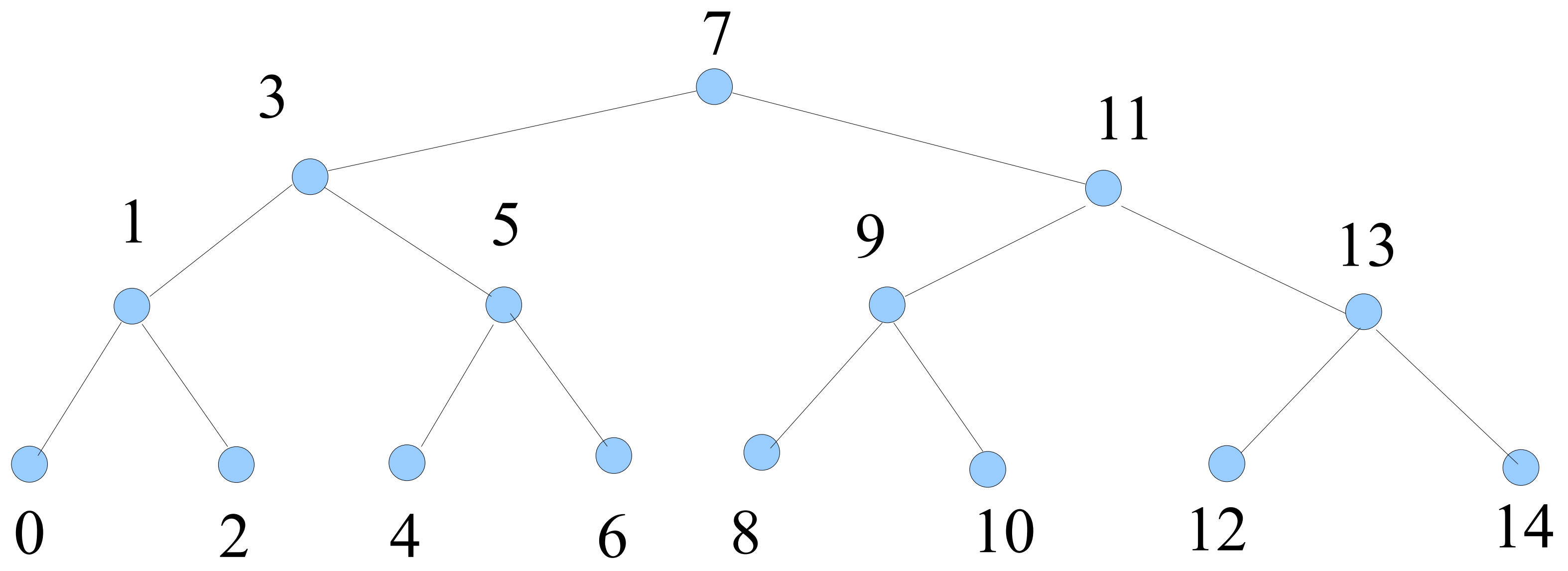
$n = 12$ $\quad\quad$ $\mathrm{trunc}(\log_2 12) = 3$

A balanced binary tree may be stored as an array: the root is stored at index 0, its left child at 1 and right child at 2. Generally, if a node is stored at index *i*, its left child is stored at $2i+1$ and right child at $2i+2$

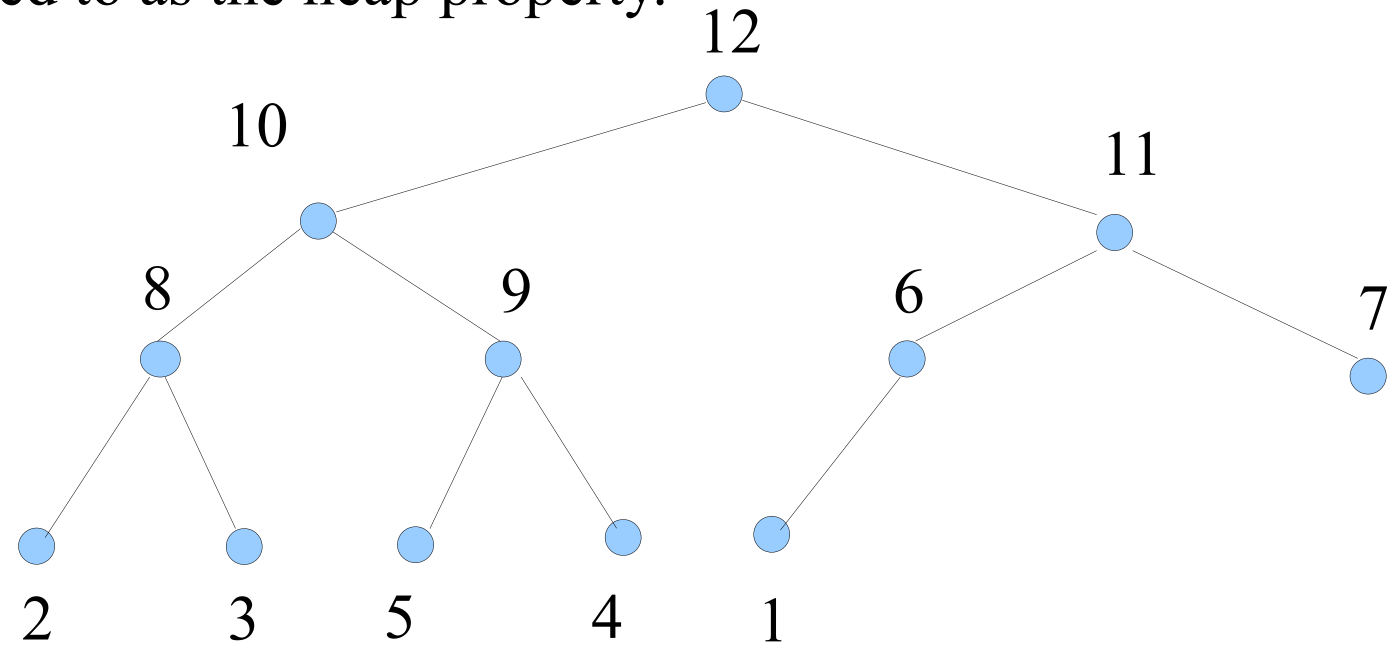


A	B	C	D	E	F	G	H	I	J	K	L
0	1	2	3	4	5	6	7	8	9	10	11

A **binary search tree** is a binary tree in which every node is assigned a unique key from an ordered set. Each node's subtree defined by its smaller child has keys less than the node's key, and the subtree defined by the larger child has keys greater than the node's key.



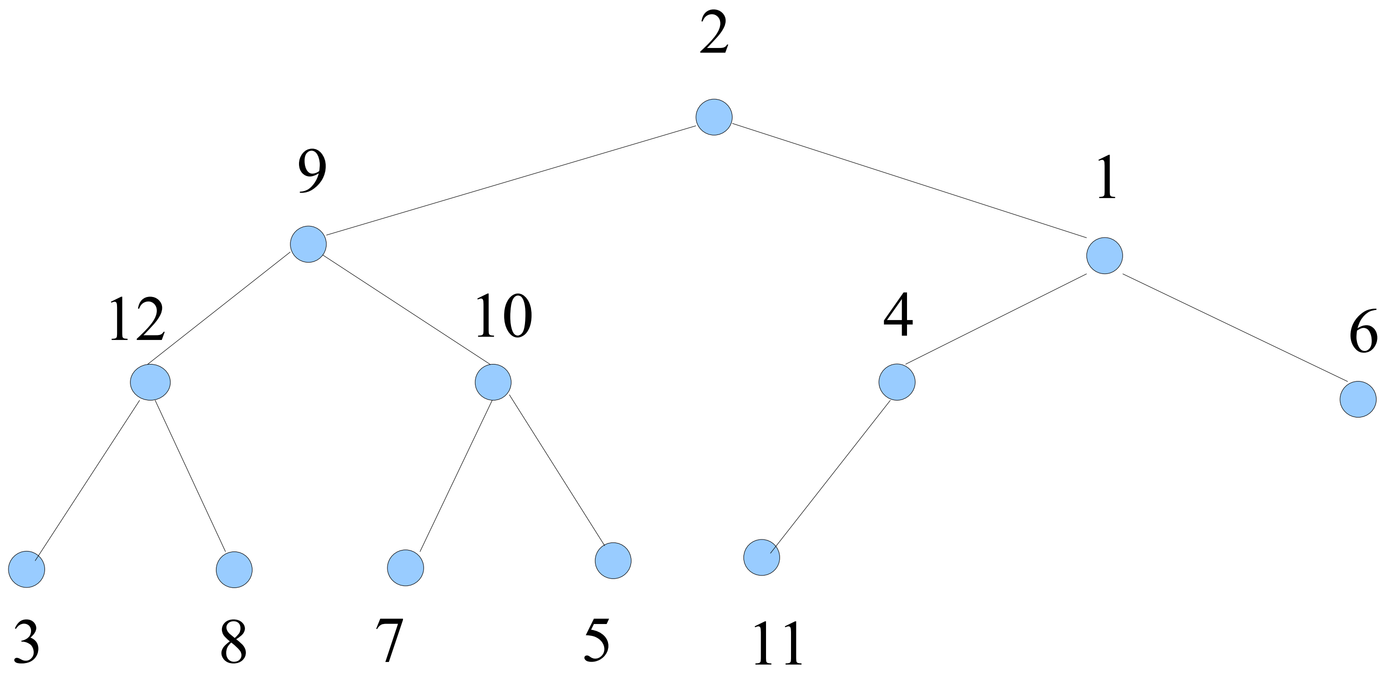
A **heap** is a structure for storing ordered data. It may be viewed as a balanced binary tree with the property that, if node n_1 is a descendent of node n_2 in the heap, then n_1 is less (greater) than n_2. This property is often referred to as the heap property.



The root of a heap is always its largest (smallest) element.

Arranging data into a heap is the underlying idea of a **heap-sort**.

Suppose the sequence 2, 9, 1, 12, 10, 4, 6, 3, 8, 7, 5, 11 is to be sorted in an ascending order. It may be represented by the balanced binary tree:



In the first phase, the nodes of this tree are rearranged to form a heap. This process is called **heapifying** or **heapification**. Subsequently, the following steps are repeated in a cycle:

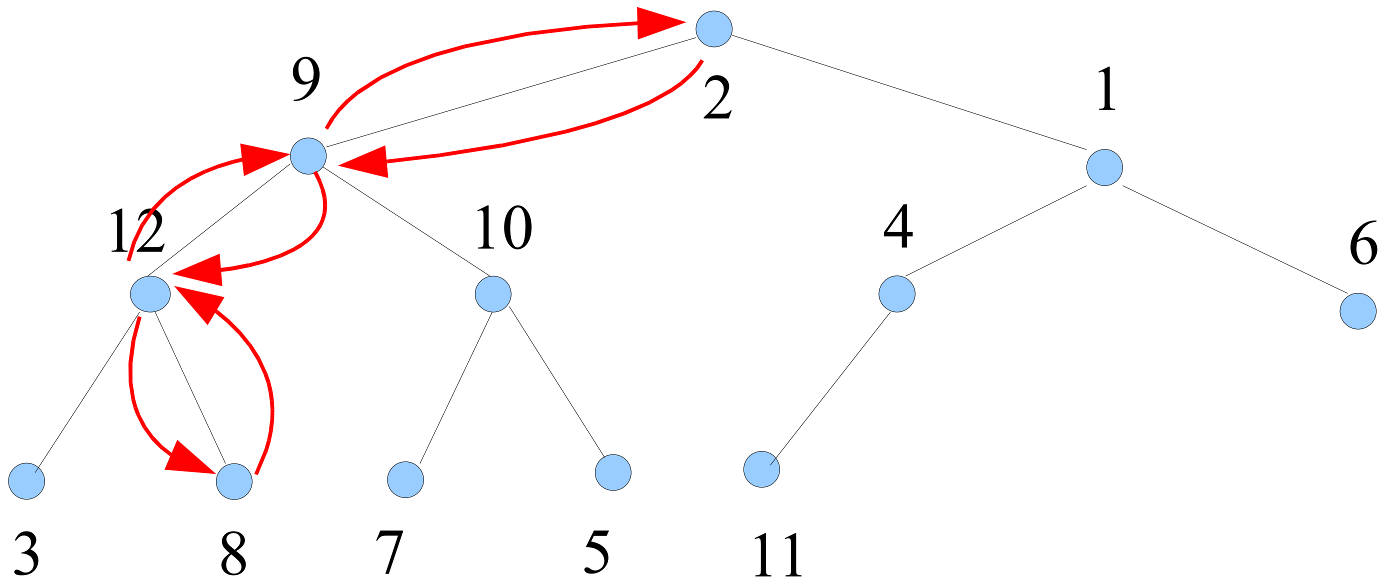
- Swap the first (largest) number *m* of the heap of not yet sorted numbers of the sequence with the number at the index at which *m* is supposed to be after the sort. Once swapped, *m* is thought of as sorted.
- Heapify the resulting balanced binary tree of not yet sorted sequence numbers.

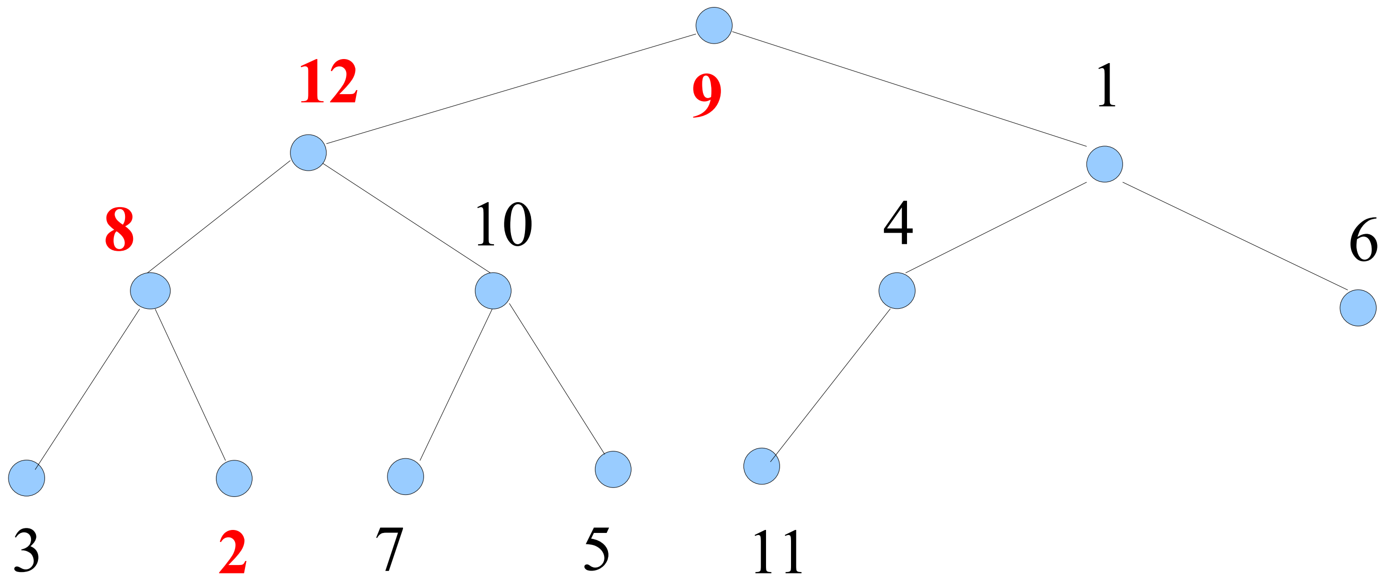
The cycle is ended if the number of unsorted numbers equals 1.

To describe the process of heapification, we first describe the **sifting down** of a node: a node n is sifted down if the following operation is recursively repeated until one of the leaves is reached:

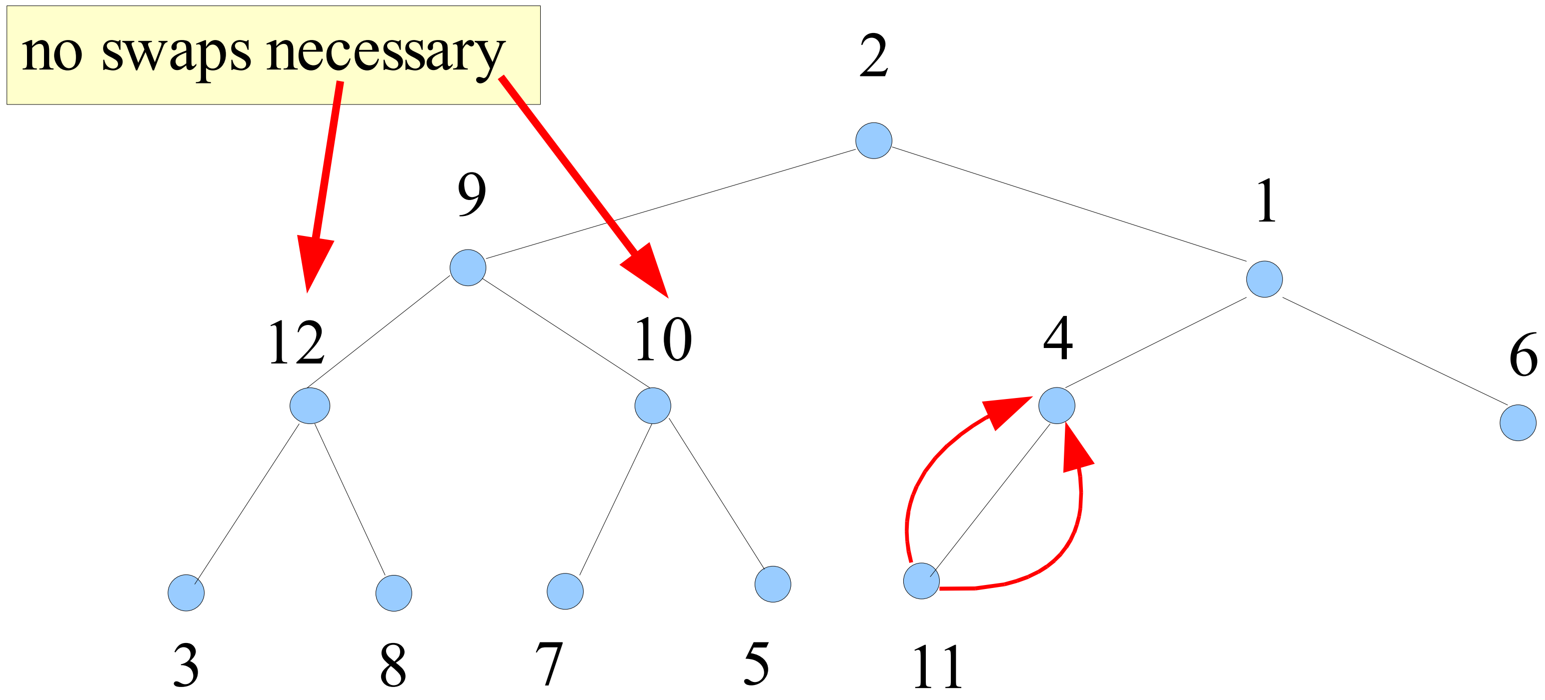
- n is compared with its children c_1 and c_2. If, say, $c_1 > n$ and $c_1 > c_2$, n is swapped with c_1 and compared with its new children, etc.

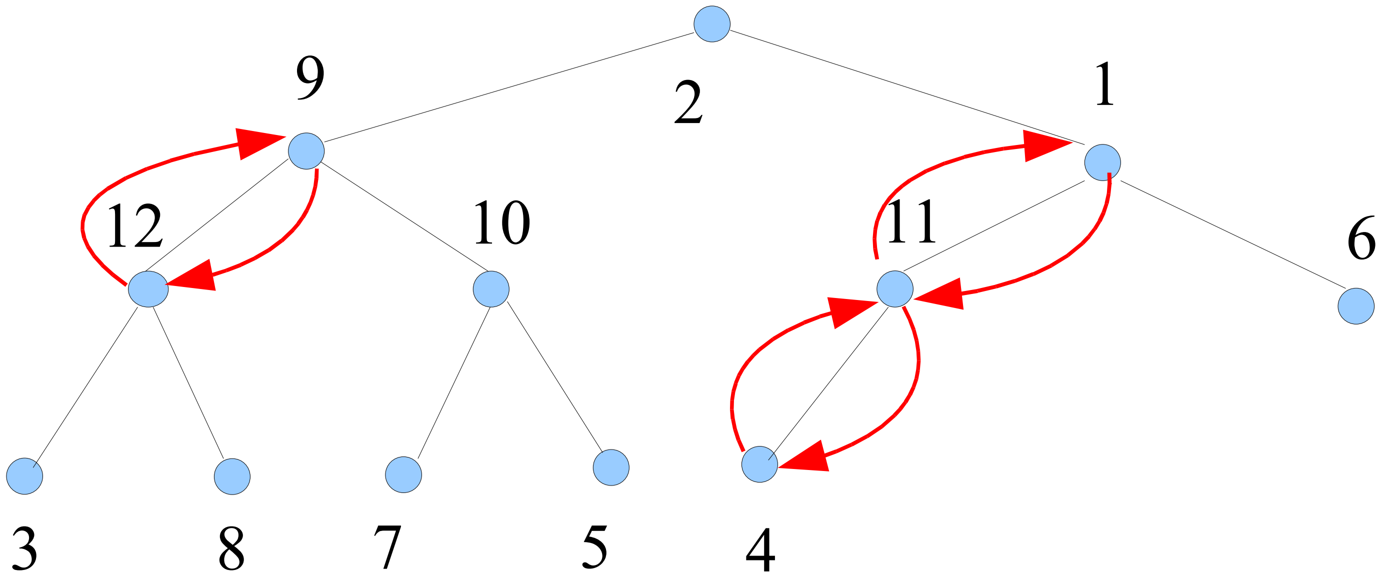
In the following example, node 2 is sifted down to the lowest position:

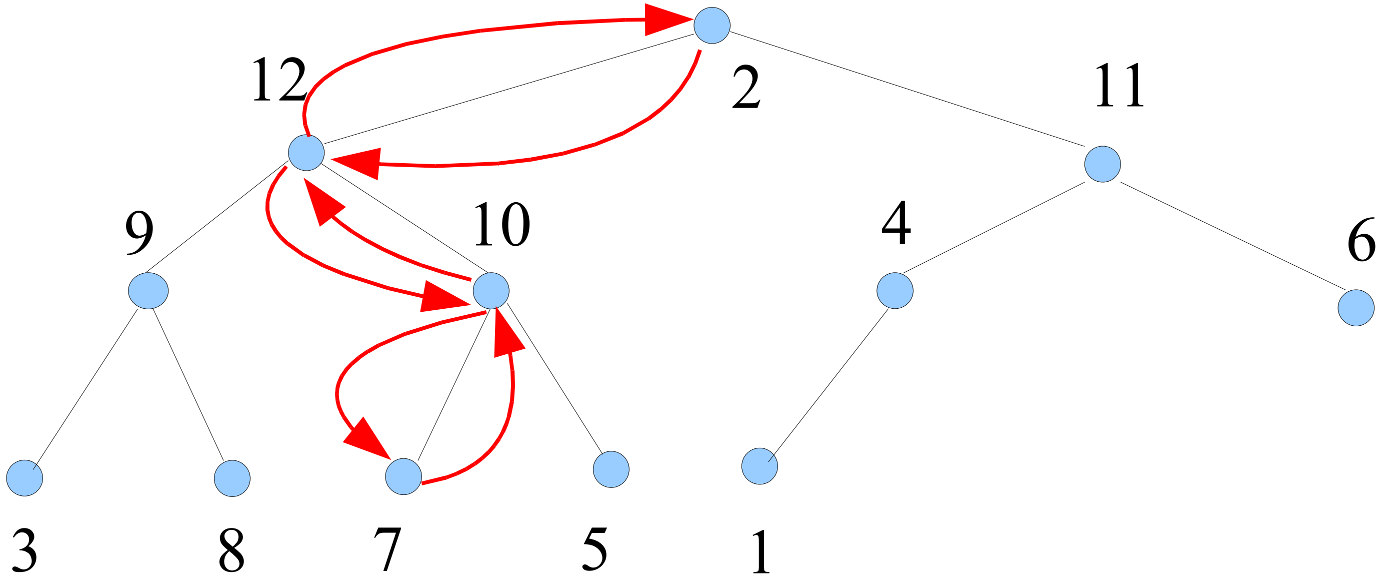

2
9
1
12
10
4
6
3
8
7
5
11

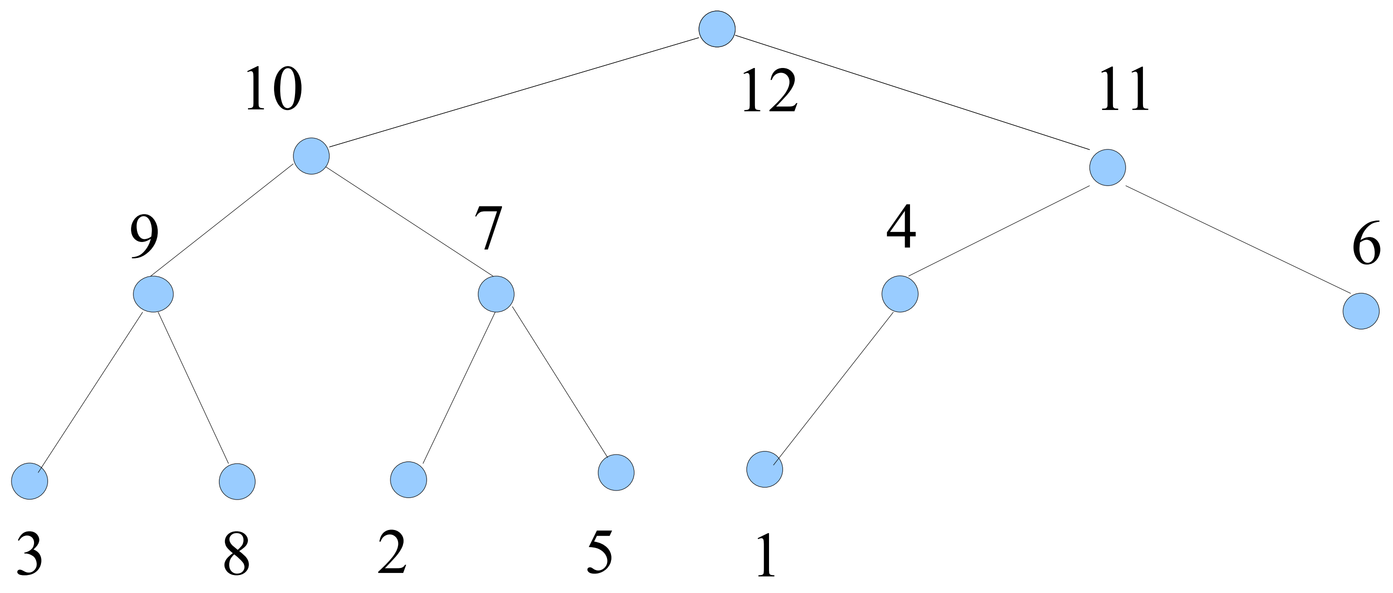

9
12
1
8
10
4
6
3
2
7
5
11

To heapify a balanced binary tree, sift down its nodes beginning with its rightmost non-leaf and ending with its root. See the following example:

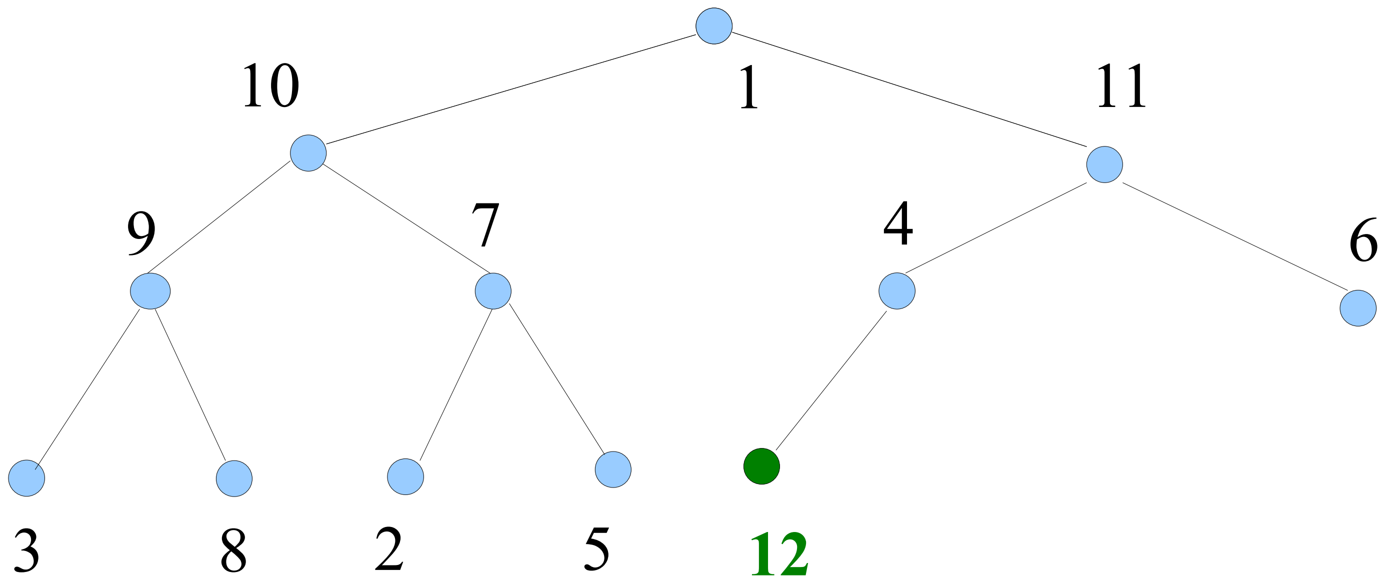



9
2
1
12
10
11
6
3
8
7
5
4

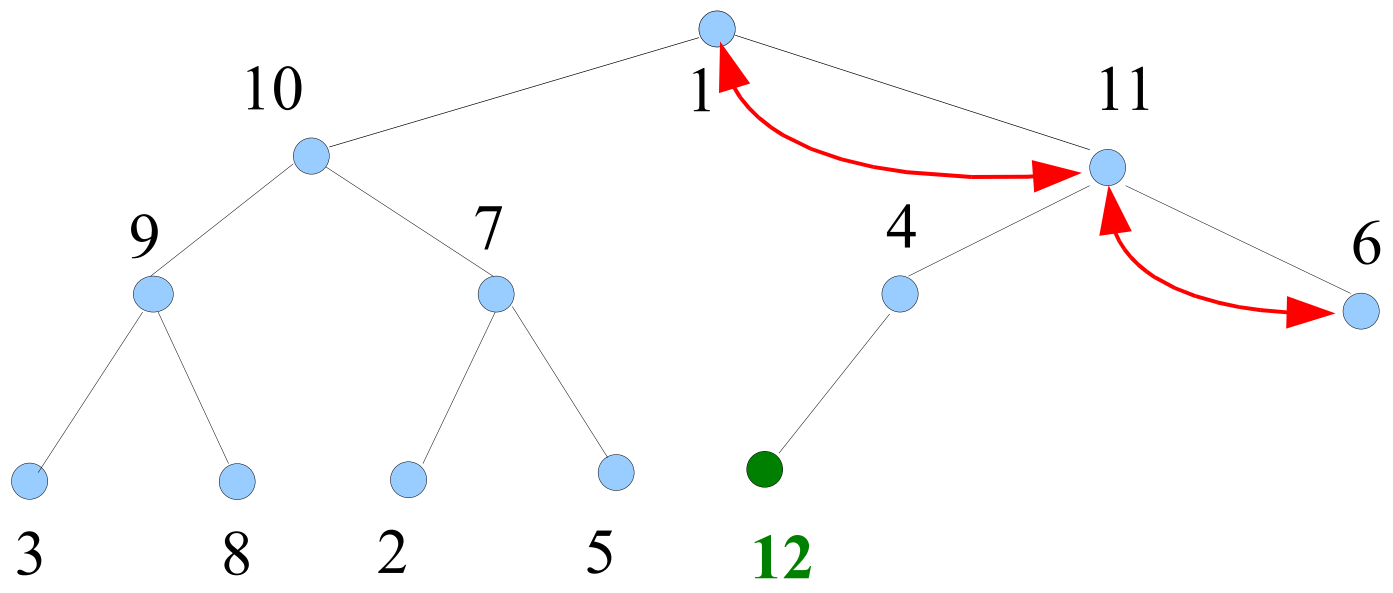

12
2
11
9
10
4
6
3
8
7
5
1



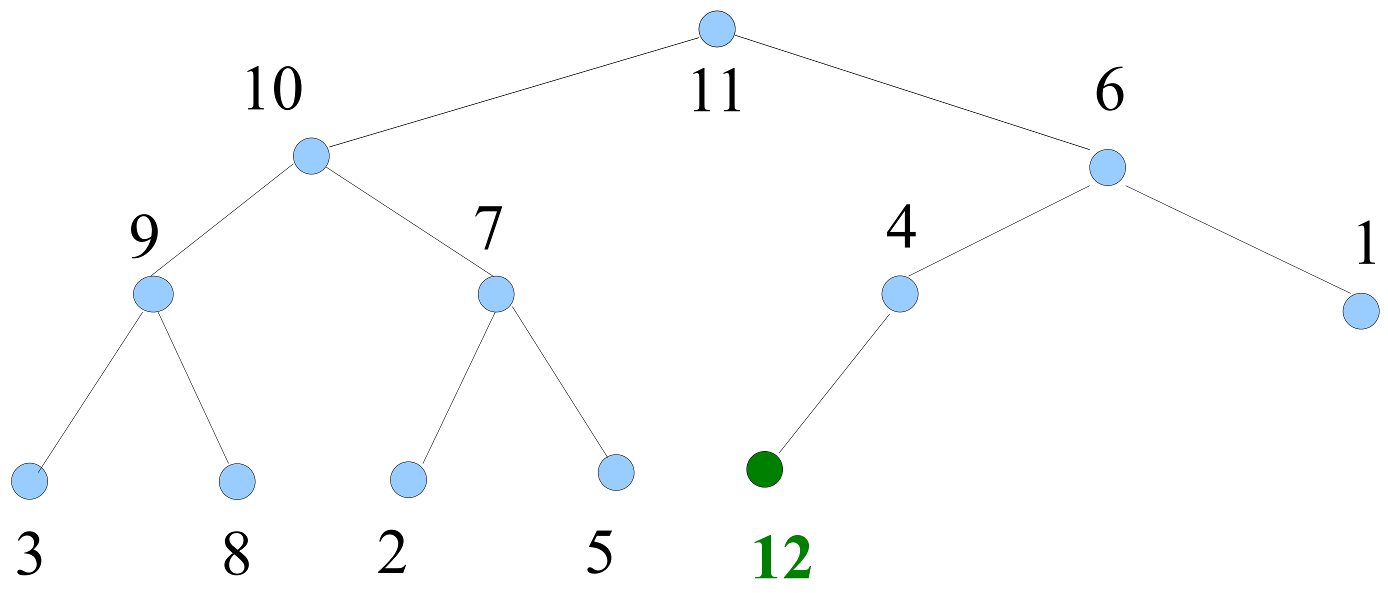
The binary tree representing the number sequence has been heapified. Now the root will be swapped with the rightmost leaf.

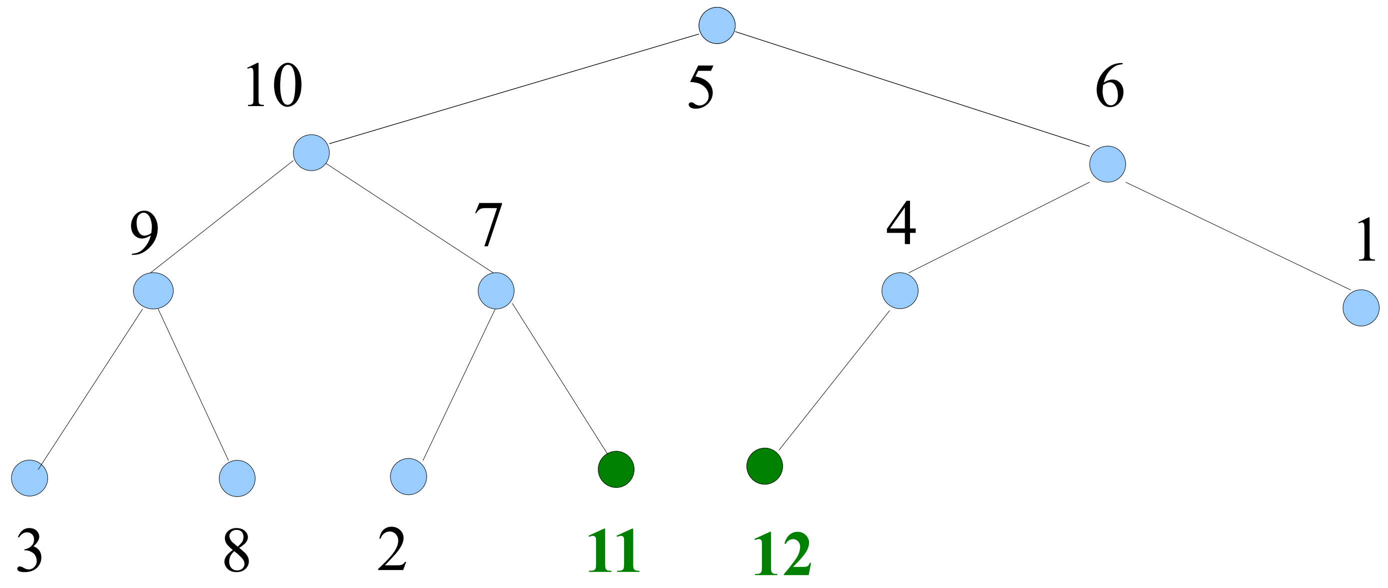


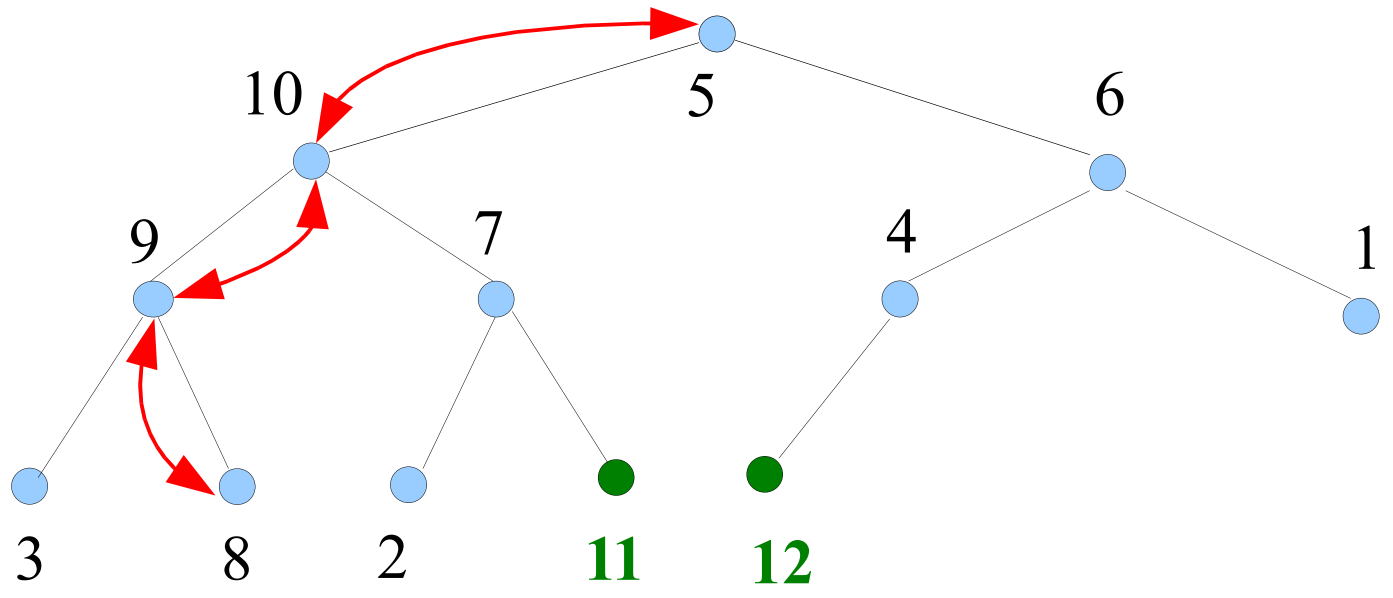
This violates the heap property of the binary tree, which has to be heapified again.



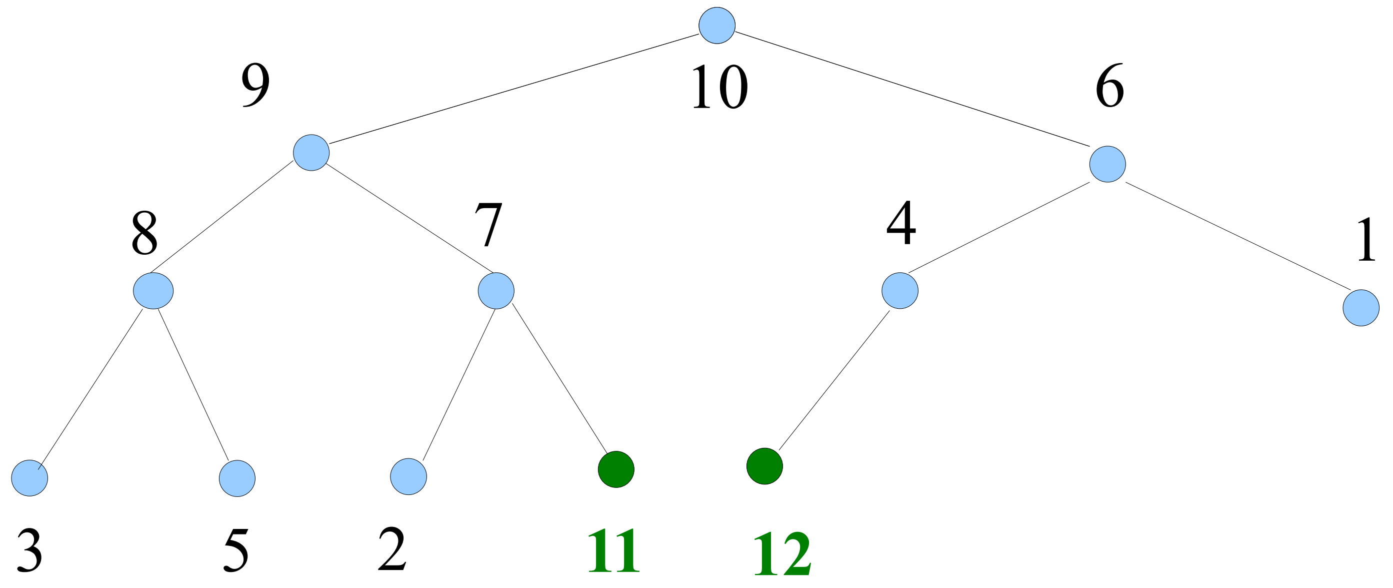
Note that, this time, it is sufficient to sift down the root to restore the heap property.


11
10
6
9
7
4
1
3
8
2
5
12

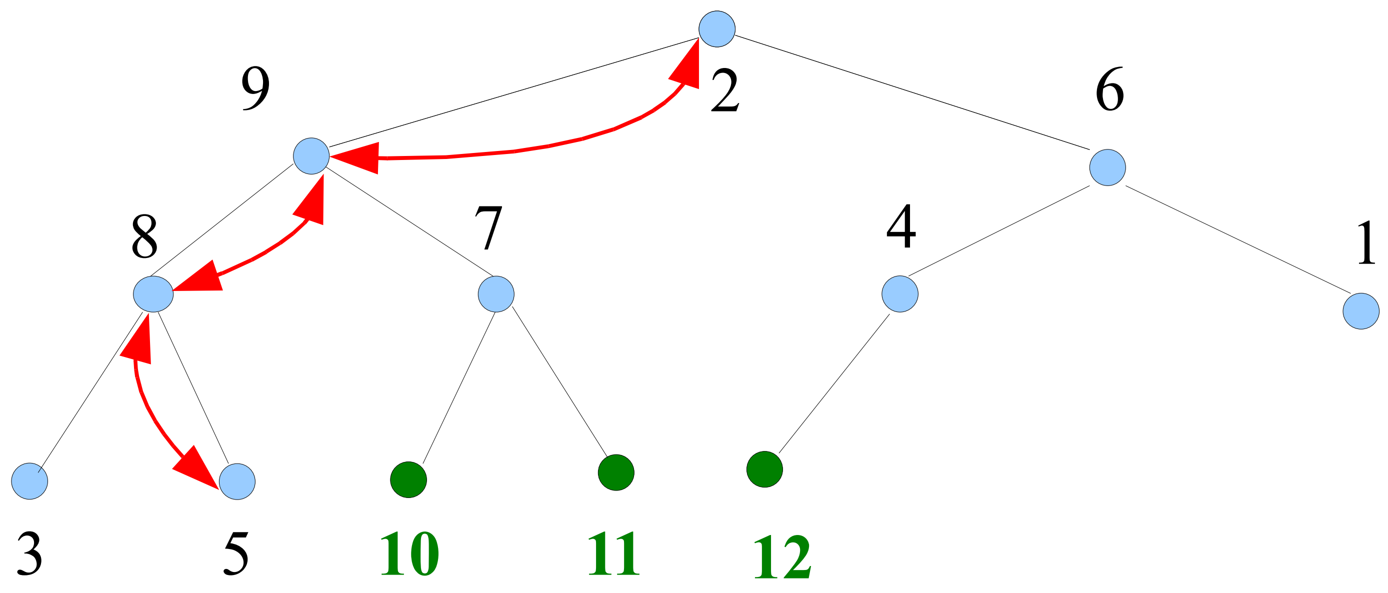

5
10
6
9
7
4
1
3
8
2
11
12



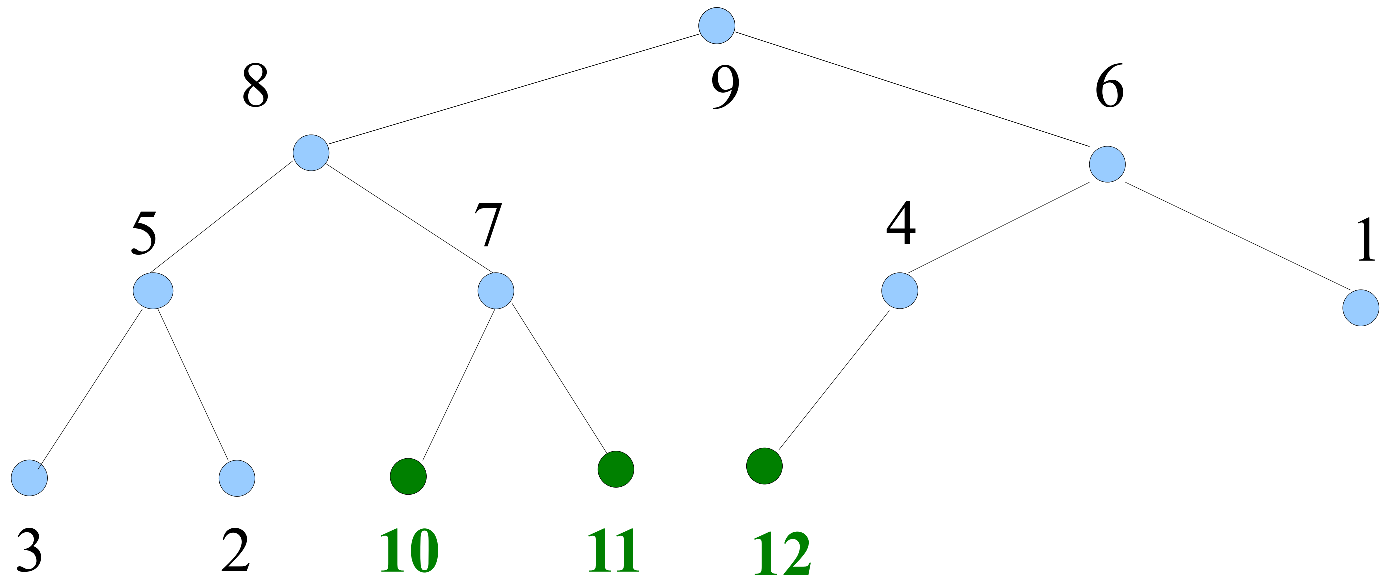
5
10
6
9
7
4
1
3
8
2
11
12



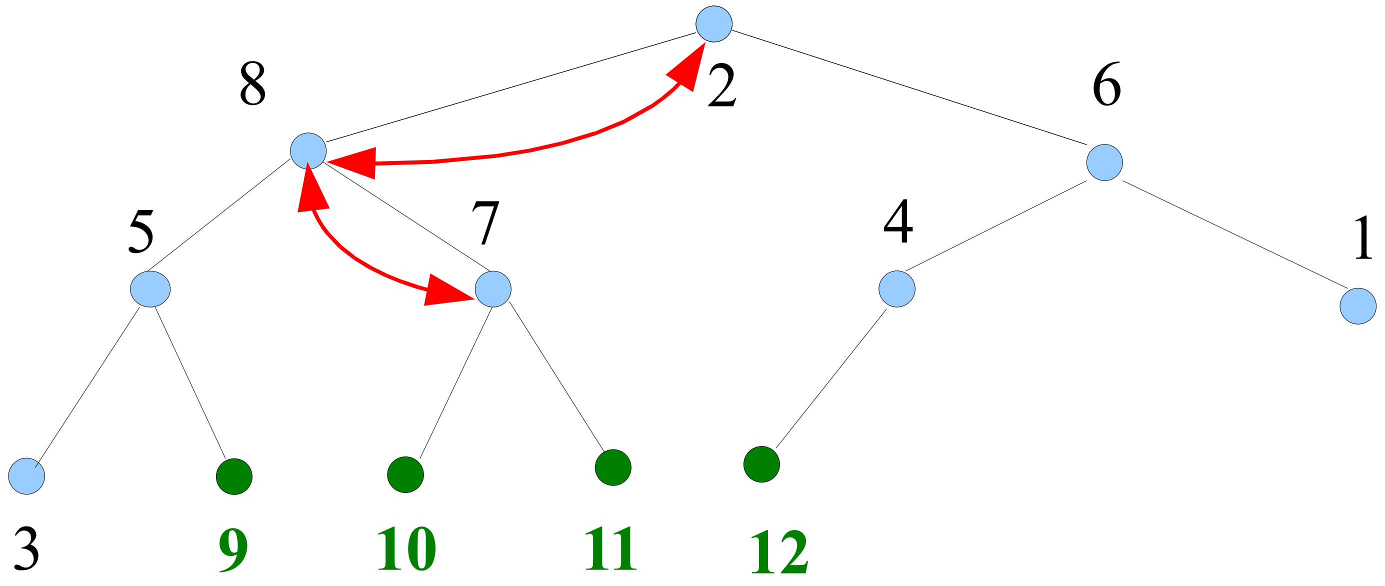
10
9
6
8
7
4
1
3
5
2
11
12



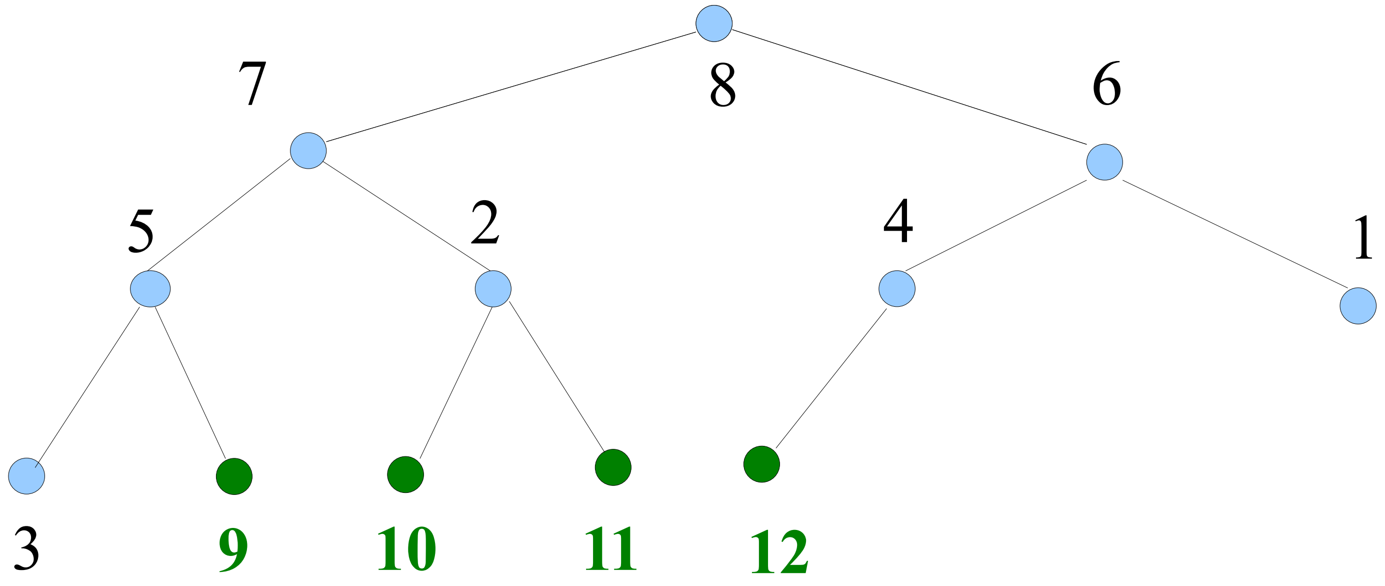
2
9
6
8
7
4
1
3
5
10
11
12



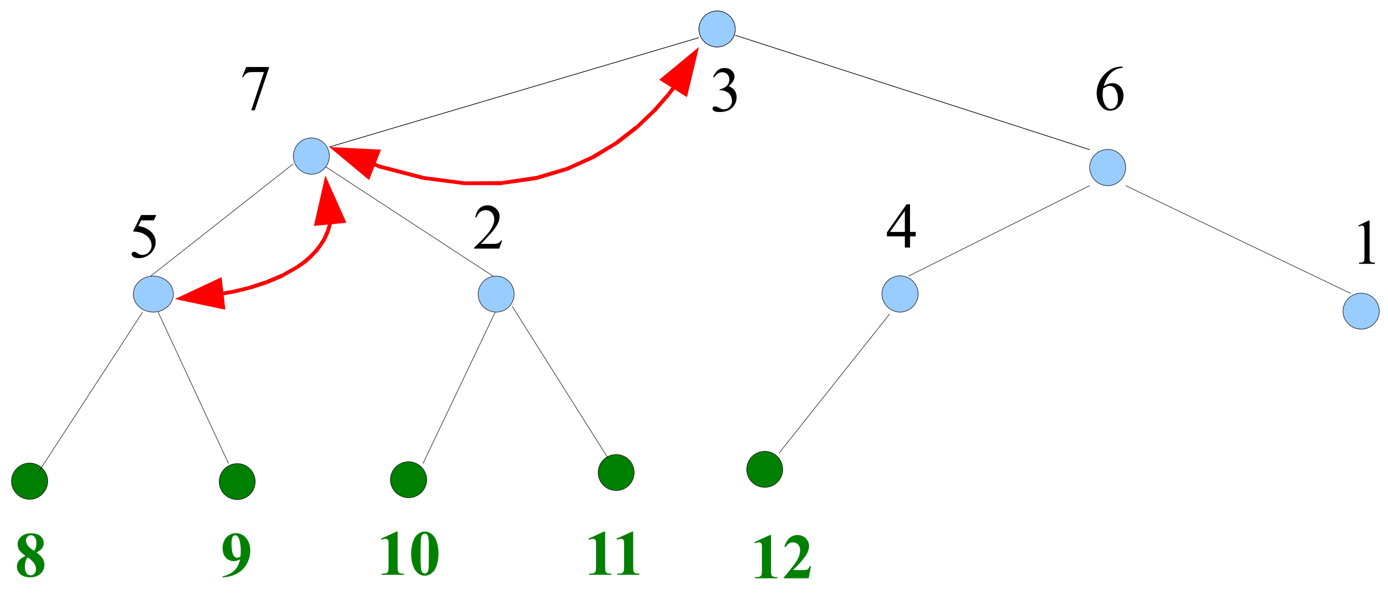
9
8
6
5
7
4
1
3
2
10
11
12



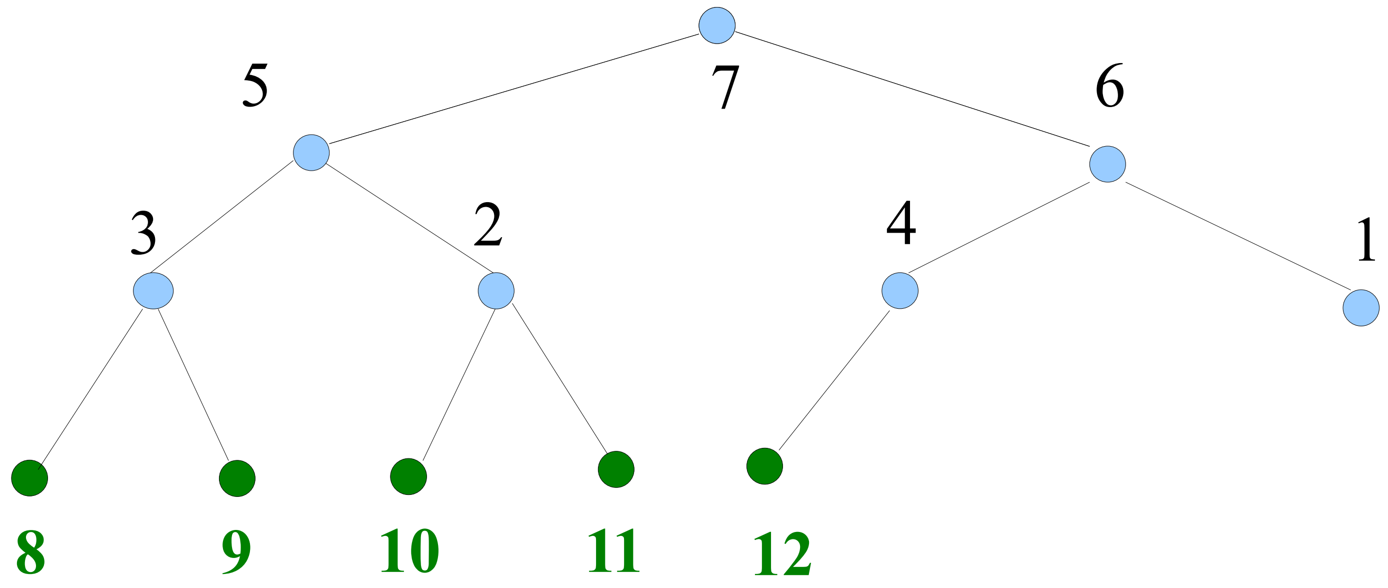
2
8
6
5
7
4
1
3
9
10
11
12



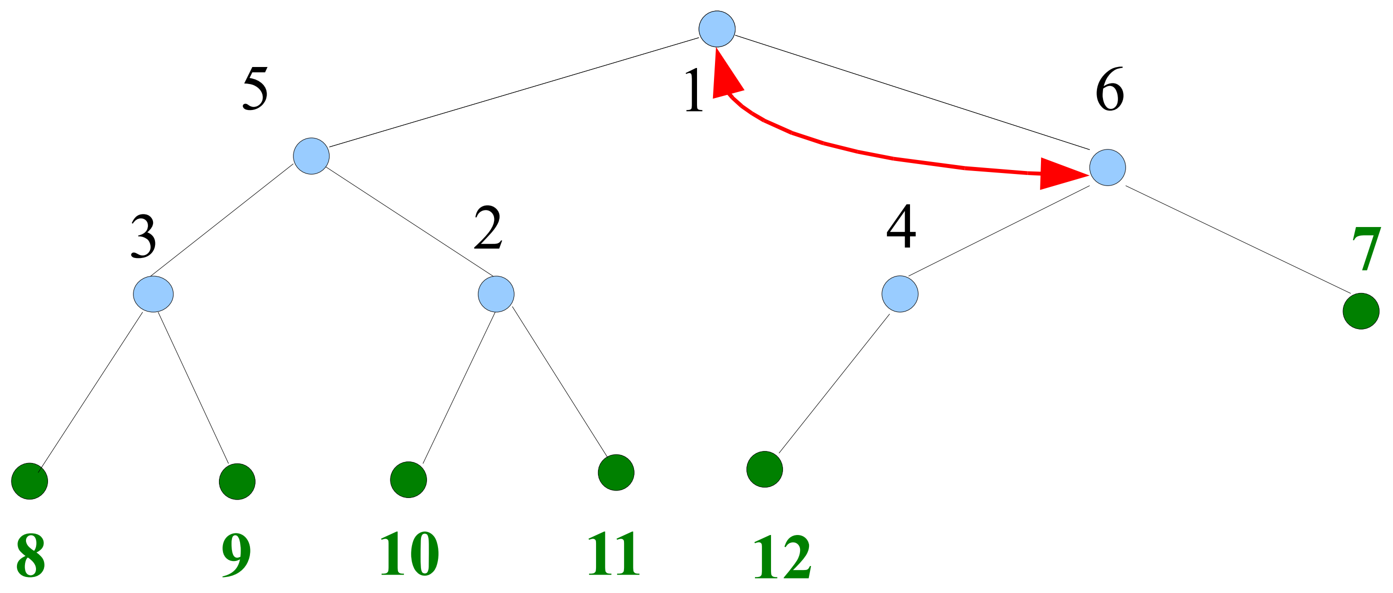
8
7
6
5
2
4
1
3
9
10
11
12

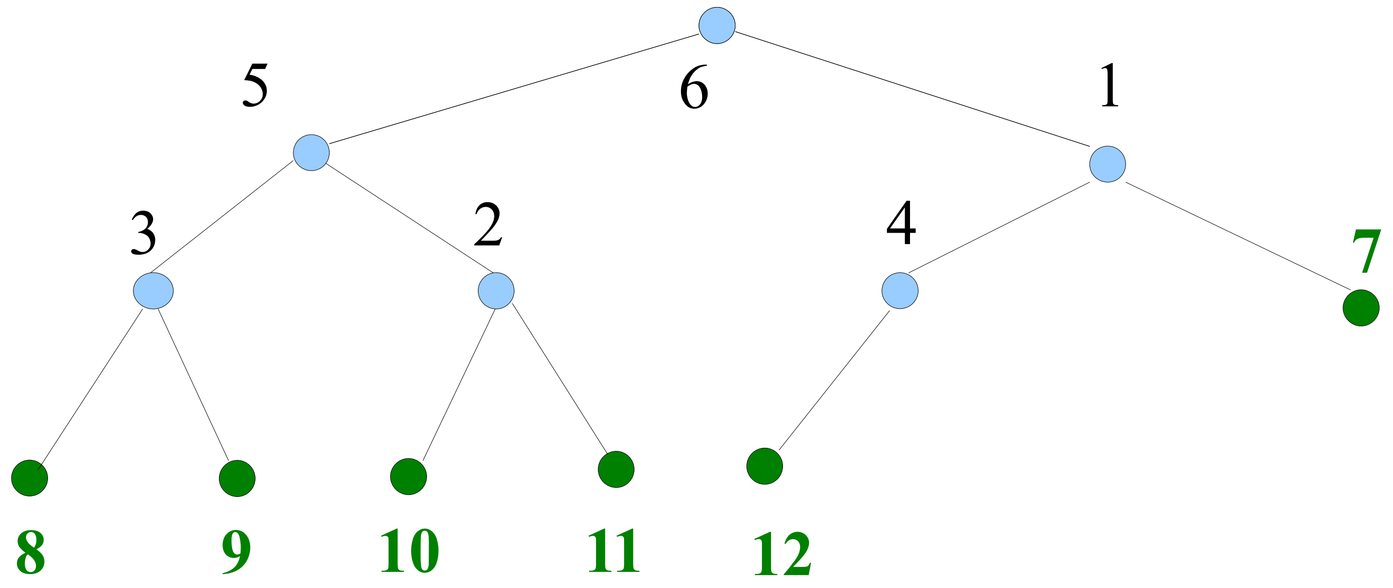


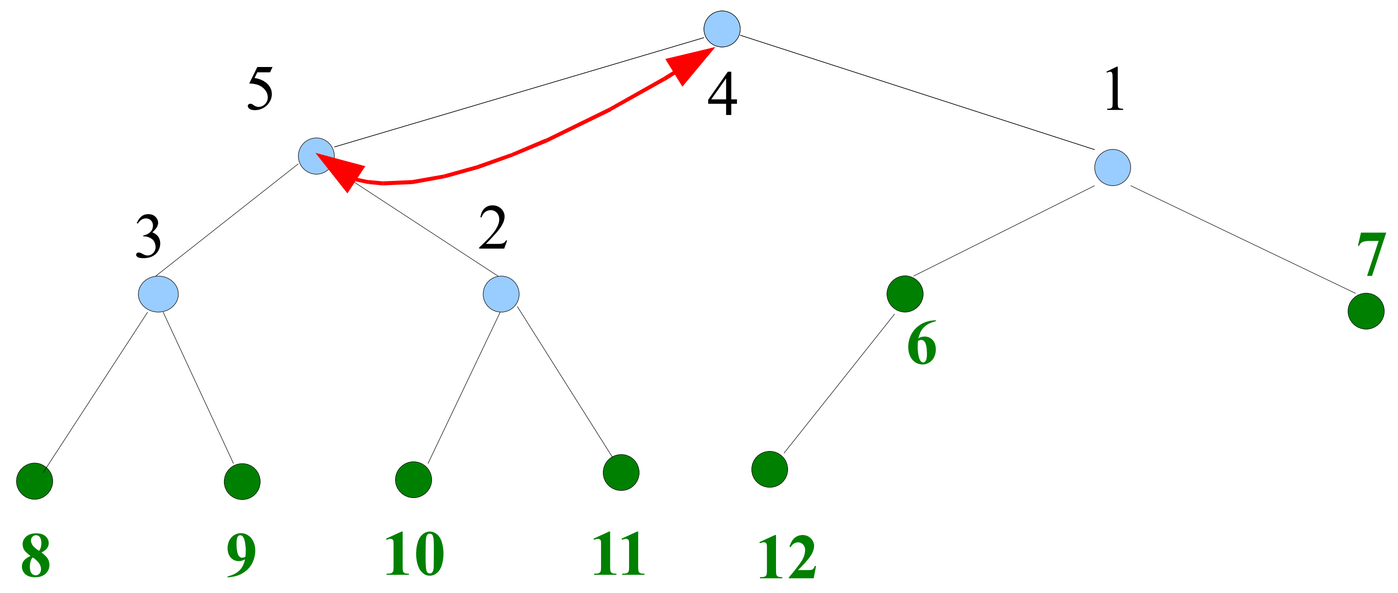
3
7
6
5
2
4
1
8
9
10
11
12

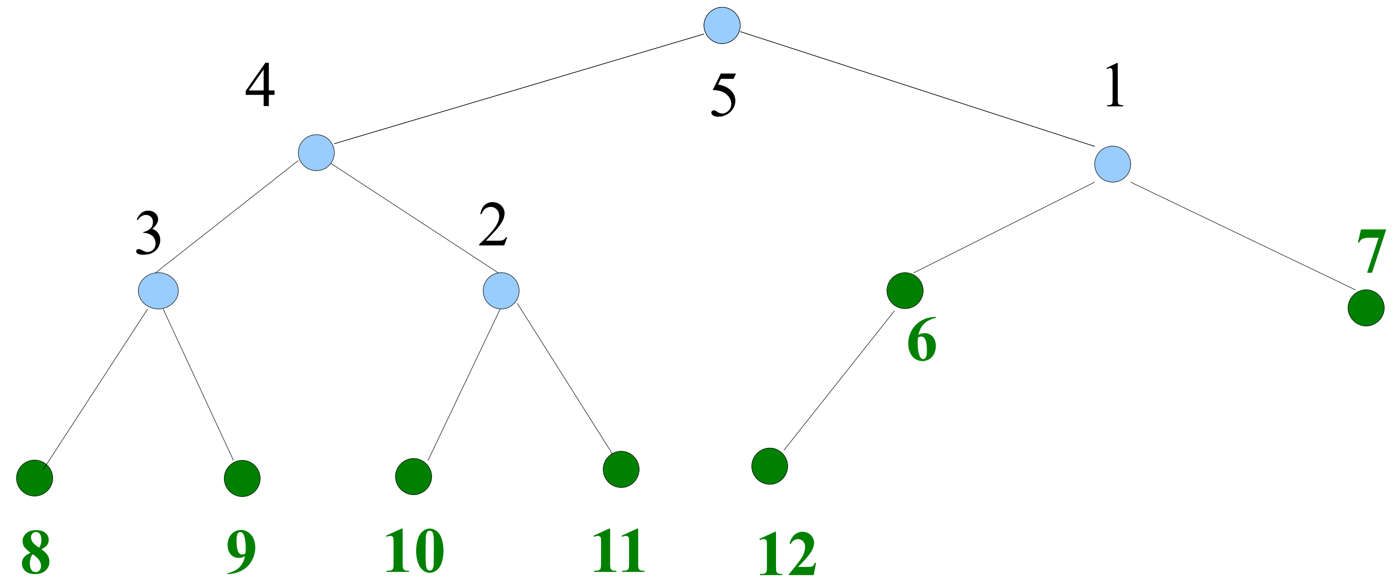


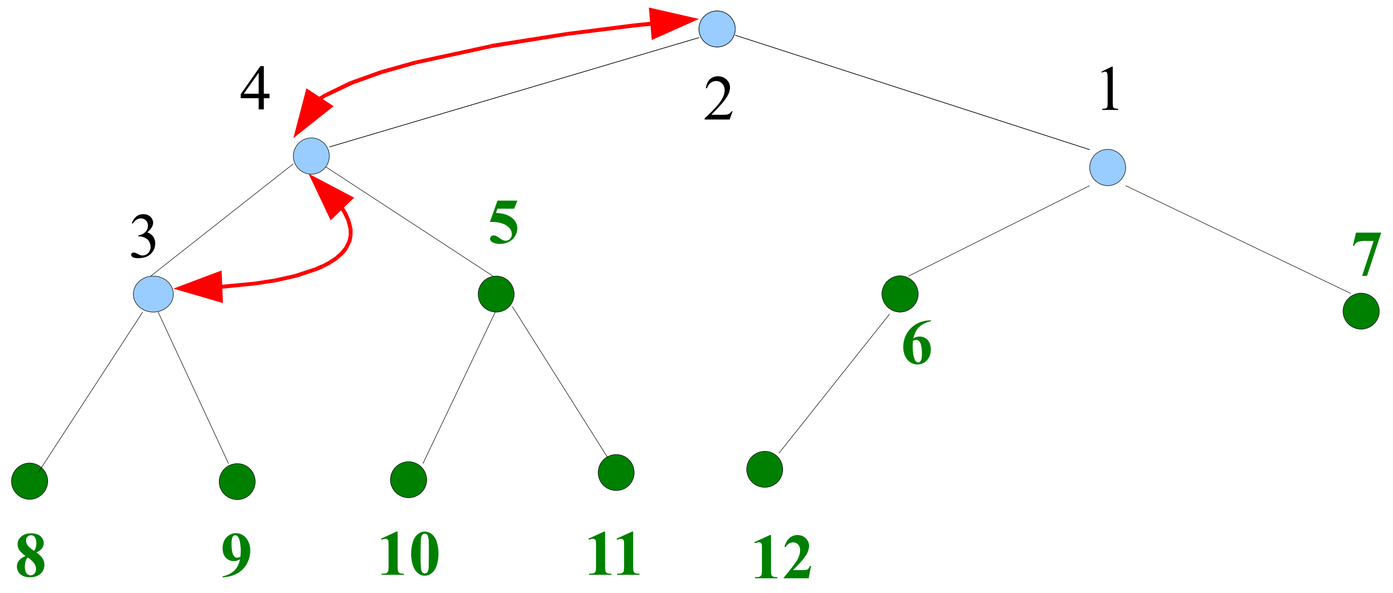
7
5
6
3
2
4
1
8
9
10
11
12

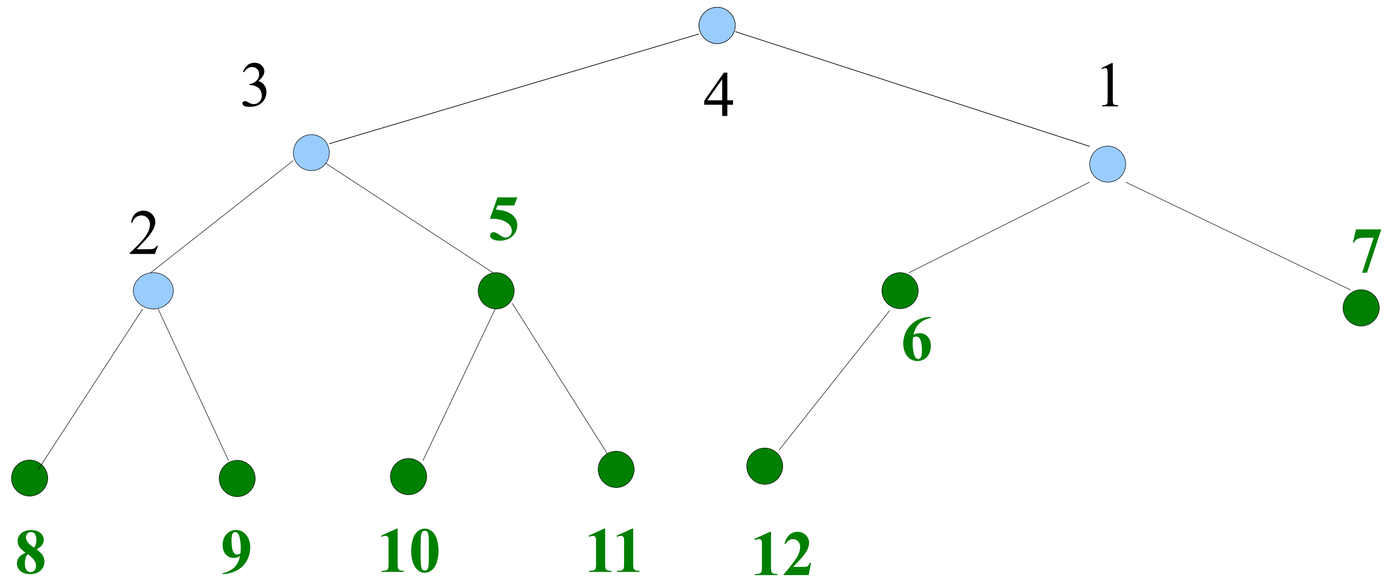

5
1
6
3
2
4
7
8
9
10
11
12

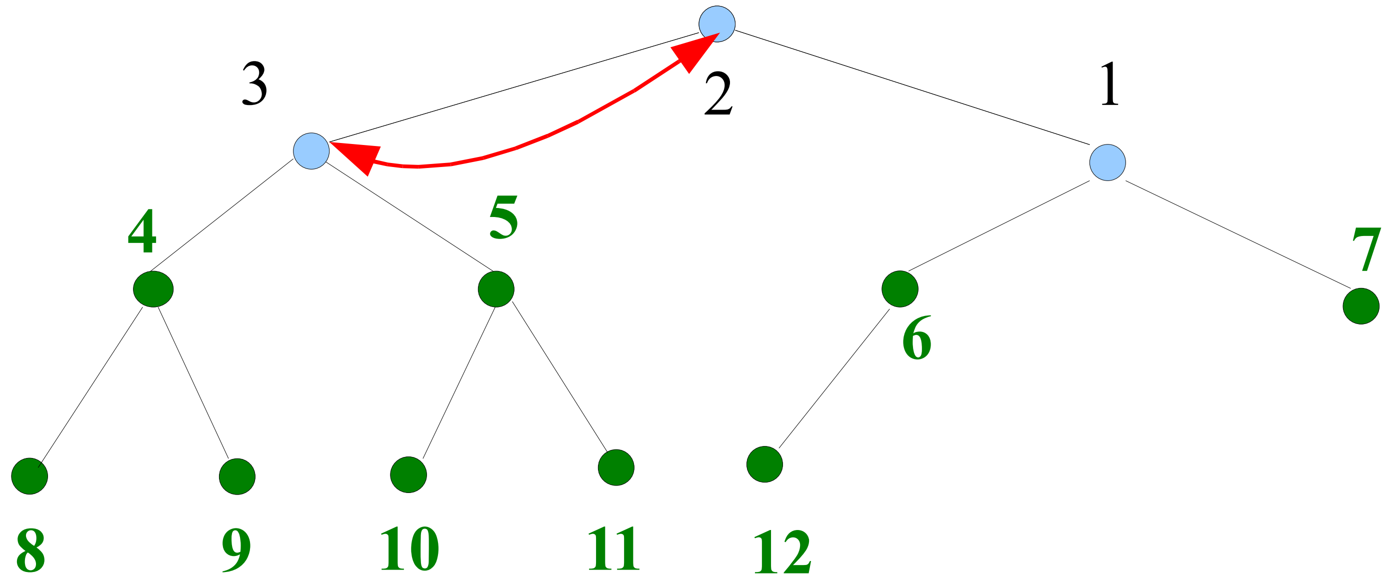

5
6
1
3
2
4
7
8
9
10
11
12

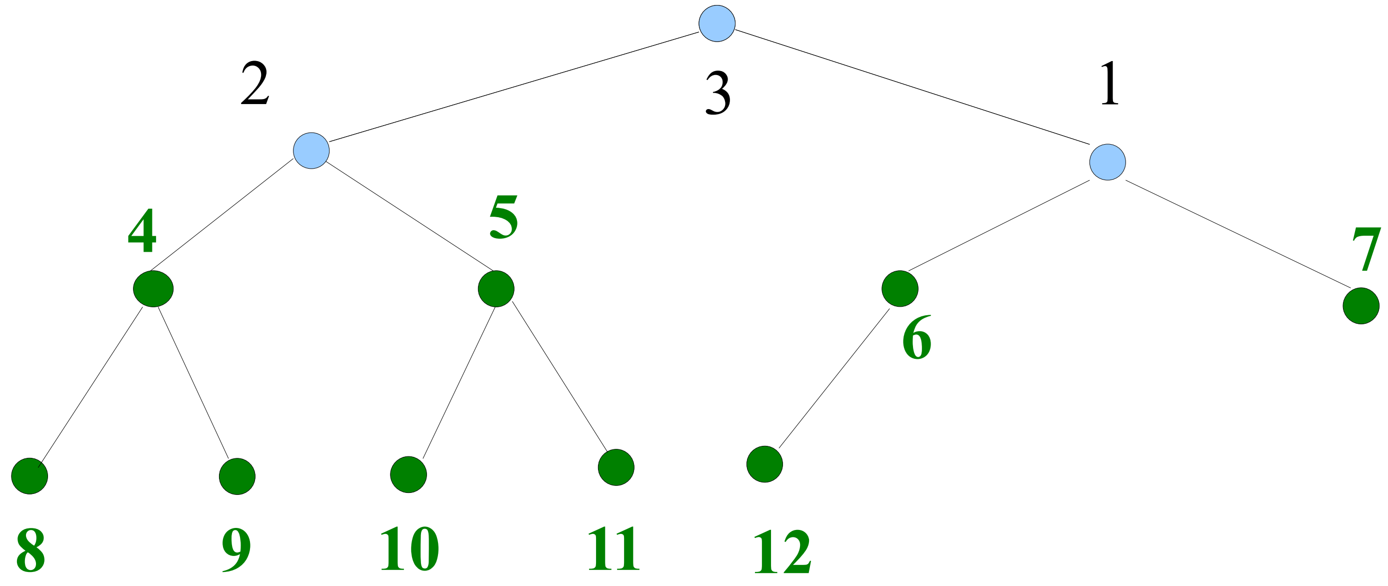

5
4
1
3
2
7
6
8
9
10
11
12

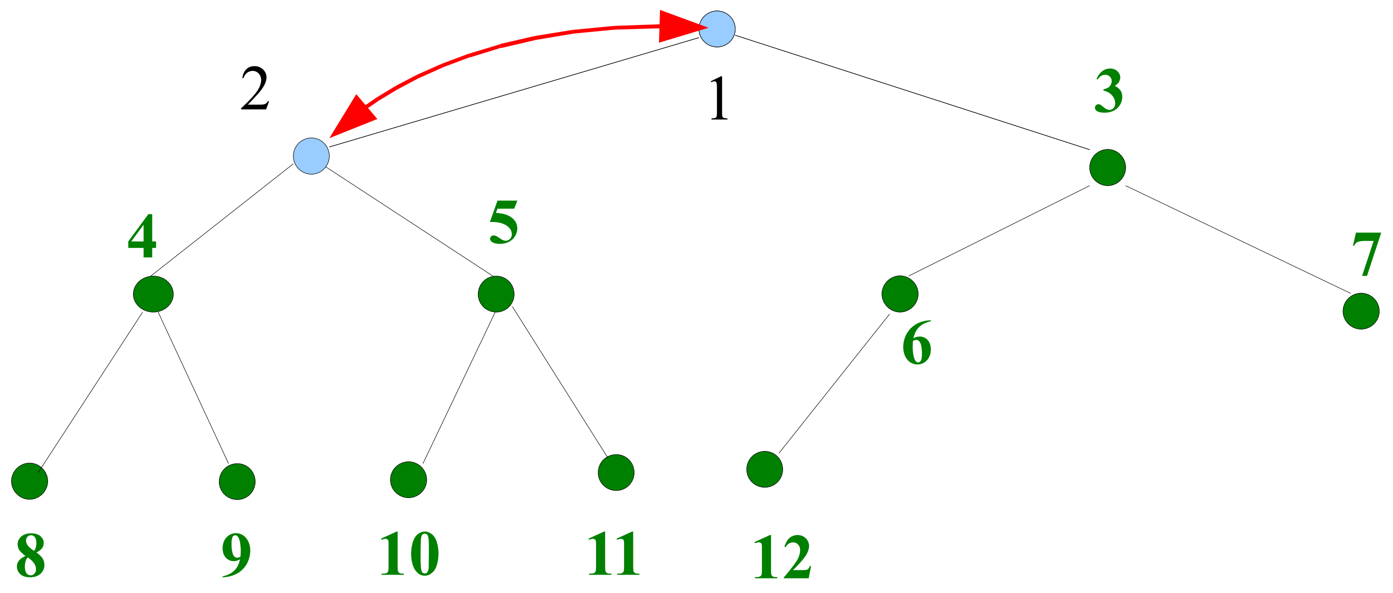

4
5
1
3
2
7
6
8
9
10
11
12

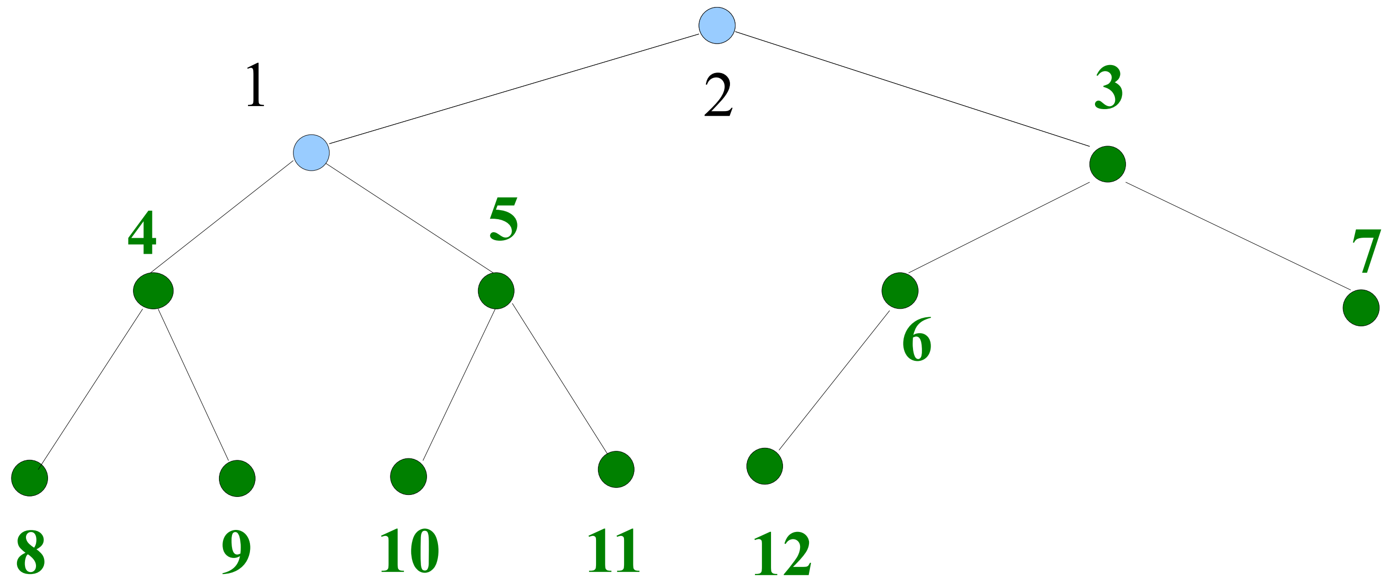

4
2
1
3
5
7
6
8
9
10
11
12

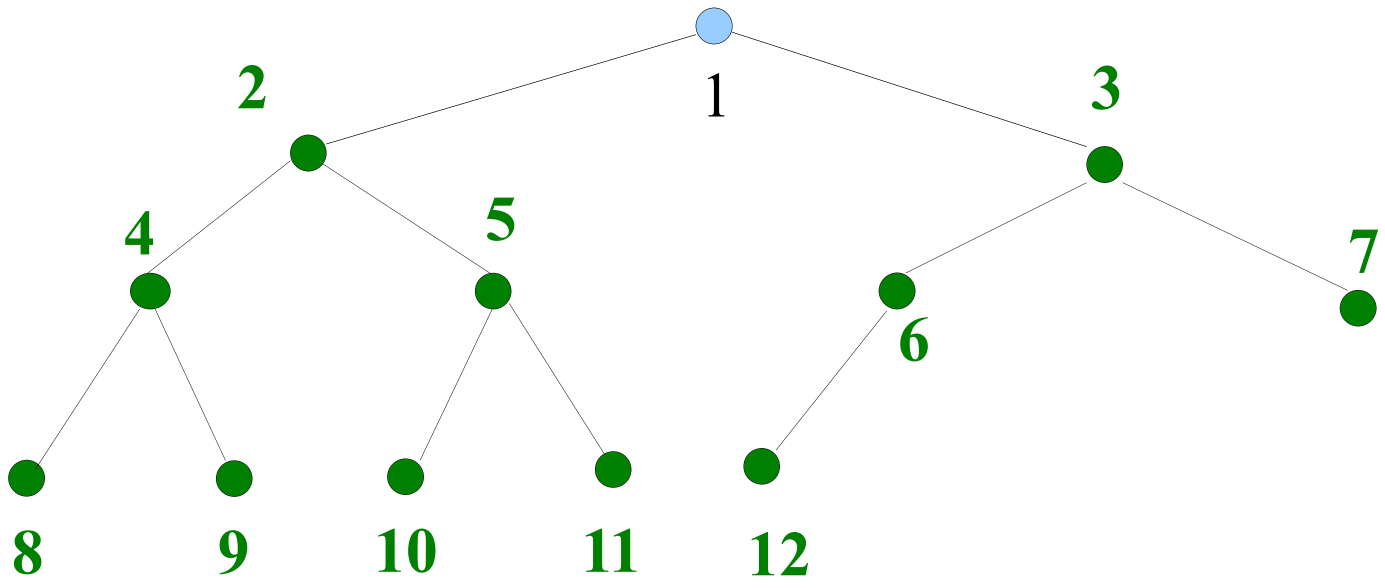

3
4
1
2
5
7
6
8
9
10
11
12

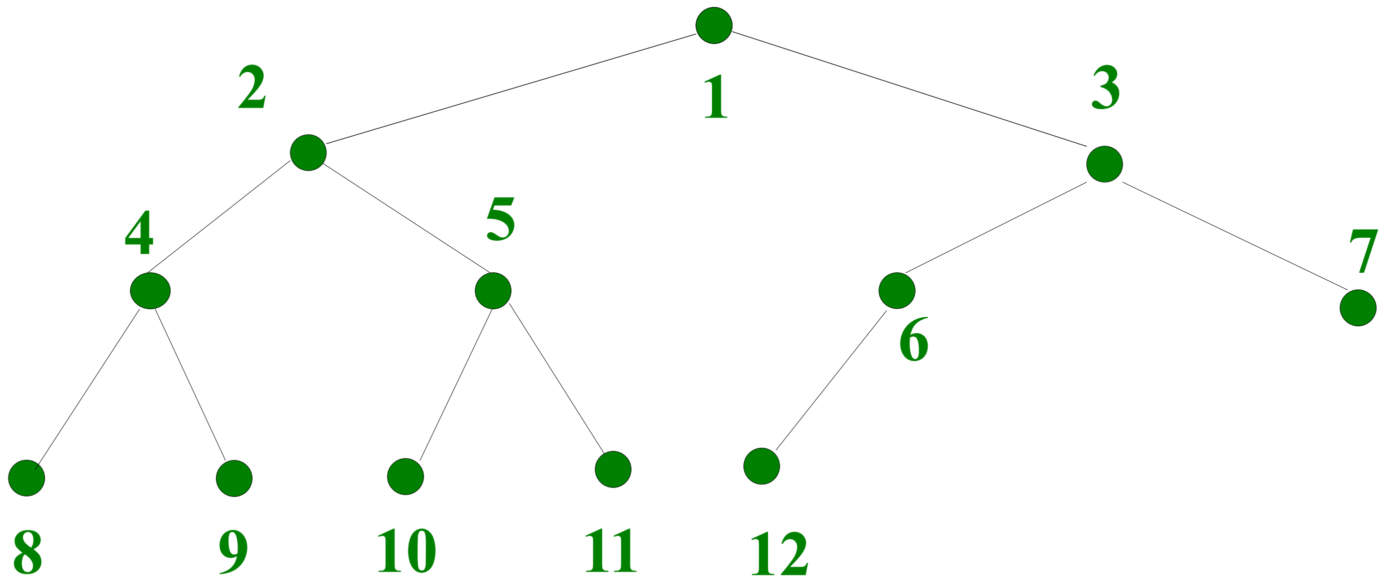

3
2
1
4
5
7
6
8
9
10
11
12


2
3
1
4
5
7
6
8
9
10
11
12


2
1
3
4
5
7
6
8
9
10
11
12


1
2
3
4
5
7
6
8
9
10
11
12


1
2
3
4
5
6
7
8
9
10
11
12


1
2
3
4
5
6
7
8
9
10
11
12

The heap-sort algorithm is known to perform in a time proportional to $n \log n$ where n is the length of the sequence.