Data Structures

Priority Queues

Dimitrios Michail

Dept. of Informatics and Telematics
Harokopio University of Athens

Priority Queue

The problem

We have items with keys and we want to be able to access the item with the smallest key. Afterwards we would like to update our collection and again access the element with the smallest key.

Priority Queue

Definition

A **priority queue** is a data structure which contains elements with keys and supports the following three basic operations:

- insert a new element
- find the element with the minimum key
- delete the element with the minimum key

Applications

- events simulation, where the key corresponds to event times
- time scheduling of jobs where keys are job priorities
- sorting numbers (we will talk about this later)

and others which we will visit in other courses

- A^* algorithm
- Dijkstra algorithm
- Huffman coding
- Prim's algorithm for finding a minimum spanning tree (MST)

Operations

A priority queue usually supports:

- create a priority queue from n elements
- insert a new element
- find the minimum
- delete the minimum
- given an element, change its priority (decrease key)
- given an element, delete it
- merge two priority queues

These operations can also function as an abstract data type ADT.

Basic Implementation

With arrays

- insert a new element: at the end of the array
- delete minimum: linear search for the minimum in the array
- etc.

Other basic implementation are

- sorted array
- list
- sorted list

Basic Implementations

Cost per Operation

	insert	delete min	find min
array	1	n	n
sorted array	n	1	1
list	1	n	n
sorted list	n	1	1

Each of the basic implementations has one operation which costs $\mathcal{O}(n)$.

Binary Heap

A tree is **heap ordered** if the key of any node is smaller or equal with the keys of its children.

Binary Heap

A tree is **heap ordered** if the key of any node is smaller or equal with the keys of its children.

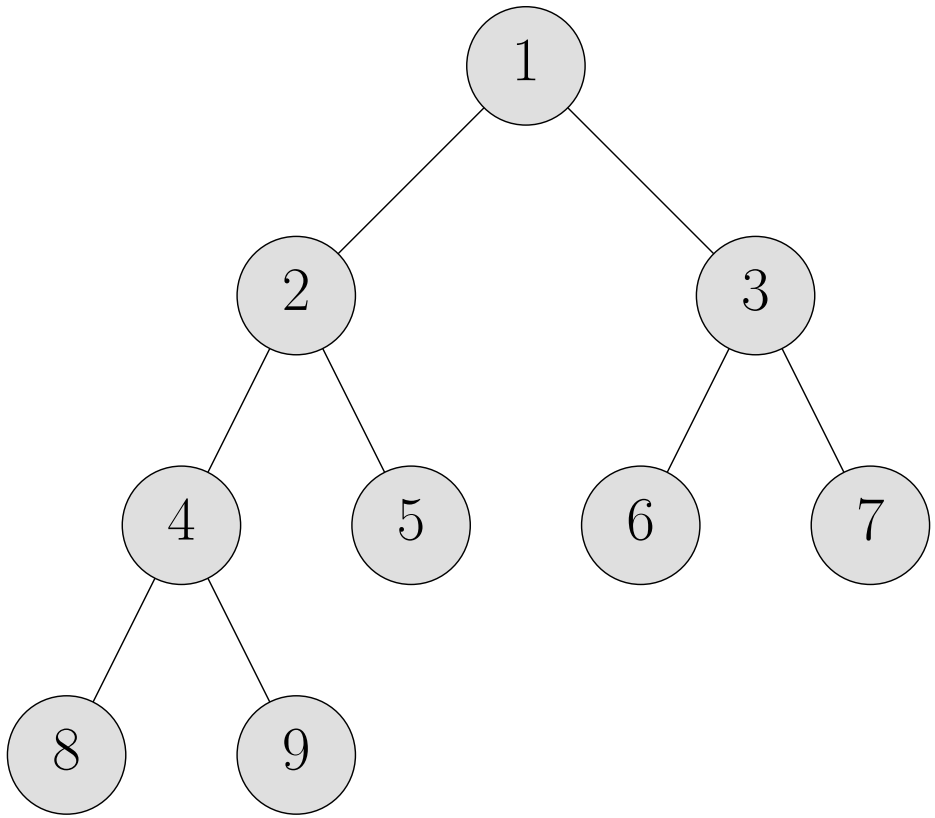
Binary Heap

A **binary heap** is a set of nodes with keys placed on a complete binary tree which is heap-ordered and represented as an array.

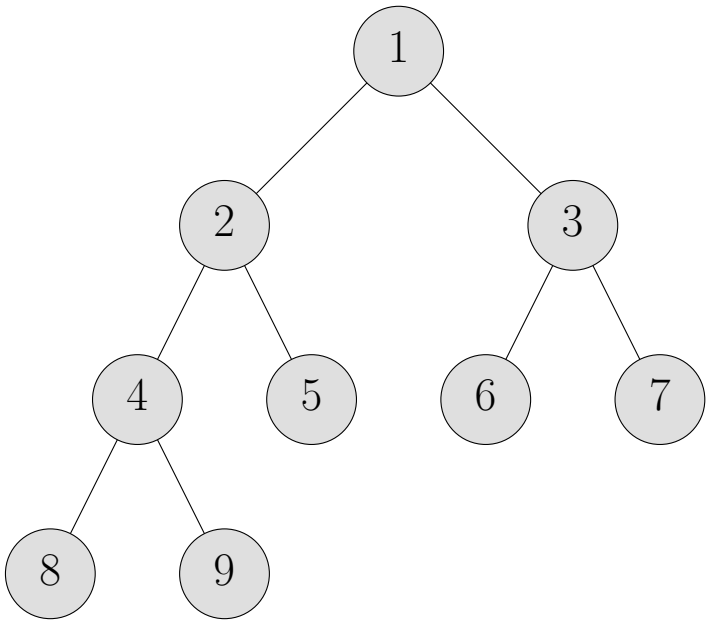
Complete Binary Tree

Definition

A binary tree where all levels, except maybe the last, are full. The last level of the tree if not complete, is filled from left to right.



Complete Binary Tree as an Array

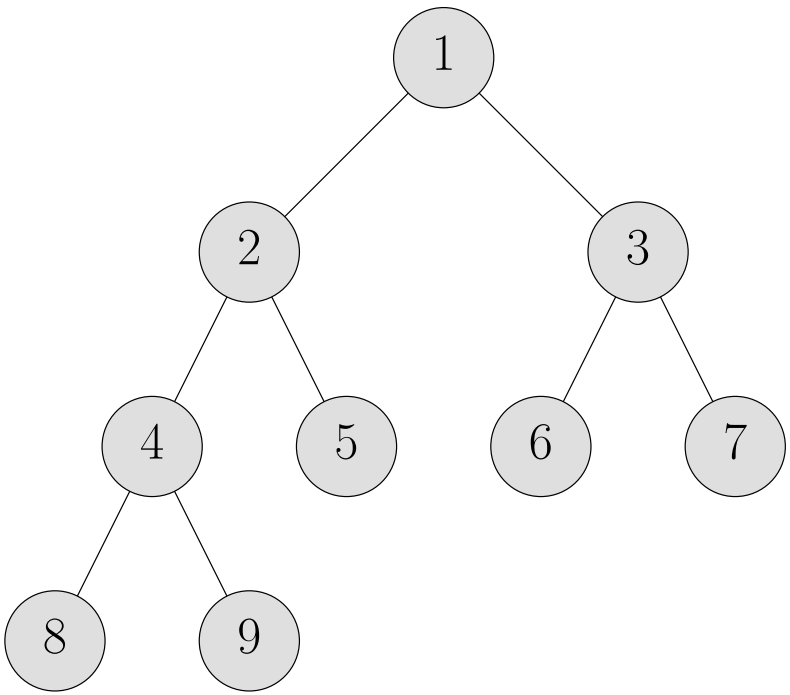


1	2	3	4	5	6	7	8	9

- $parent(i) = \lfloor i/2 \rfloor$
- $left - child(i) = 2i$
- $right - child(i) = 2i + 1$

Complete Binary Tree Height

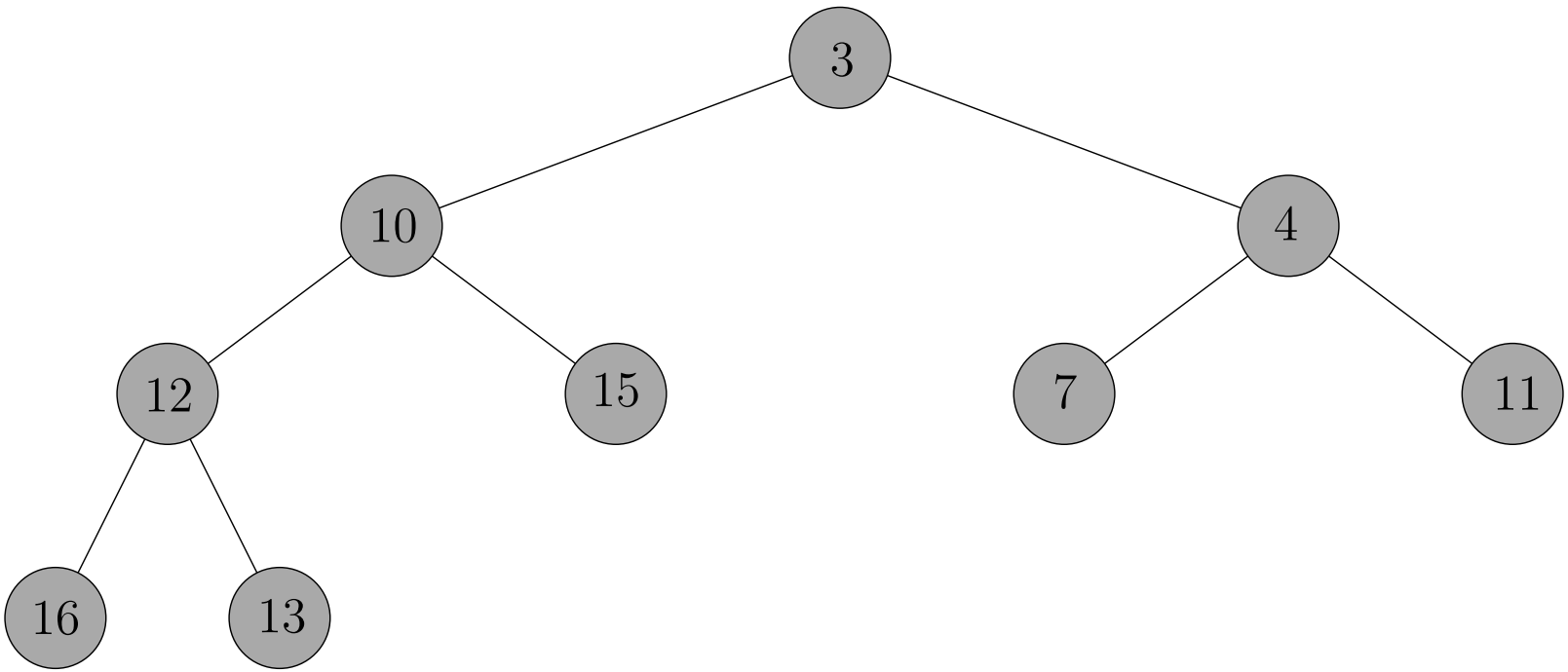
with n nodes



Since at every node its children are divided by two, such a tree has $\mathcal{O}(\log n)$ levels.

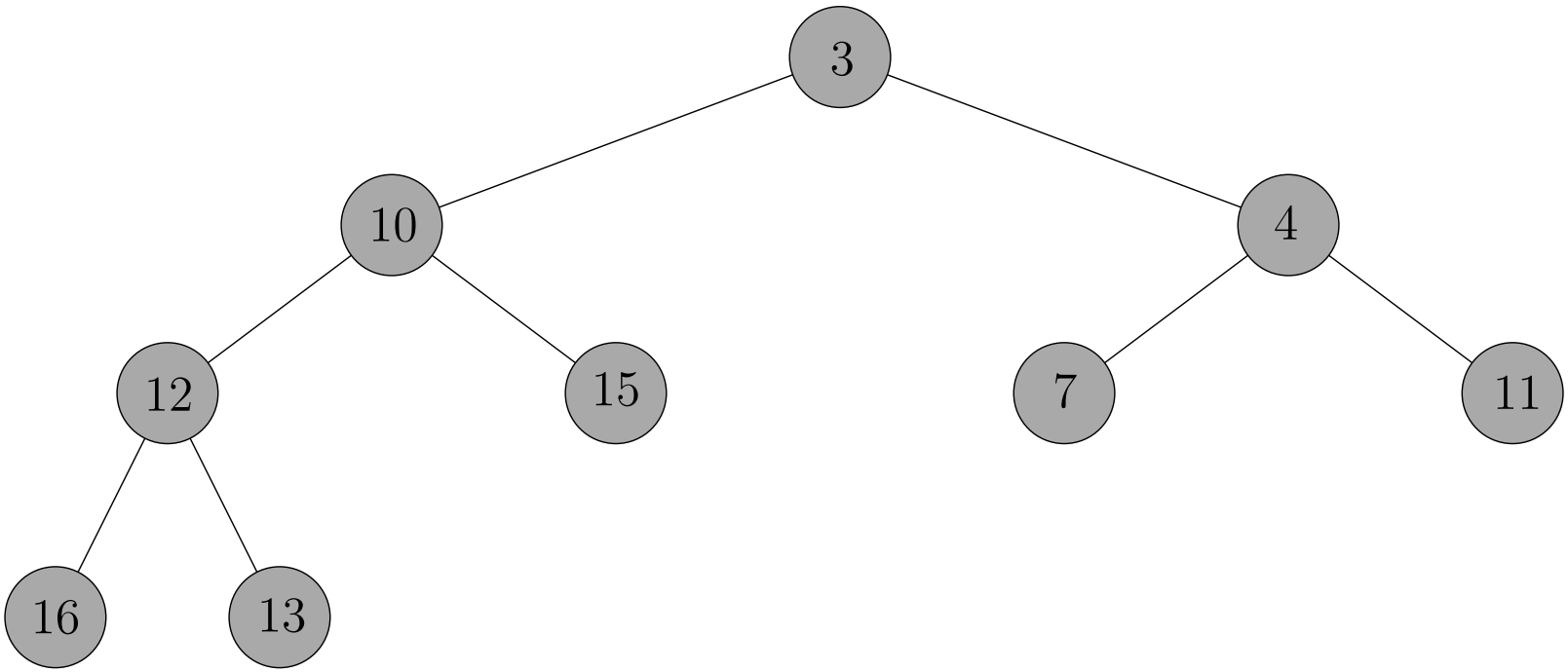
This is the property that binary heaps are based on.

Binary Heap Example



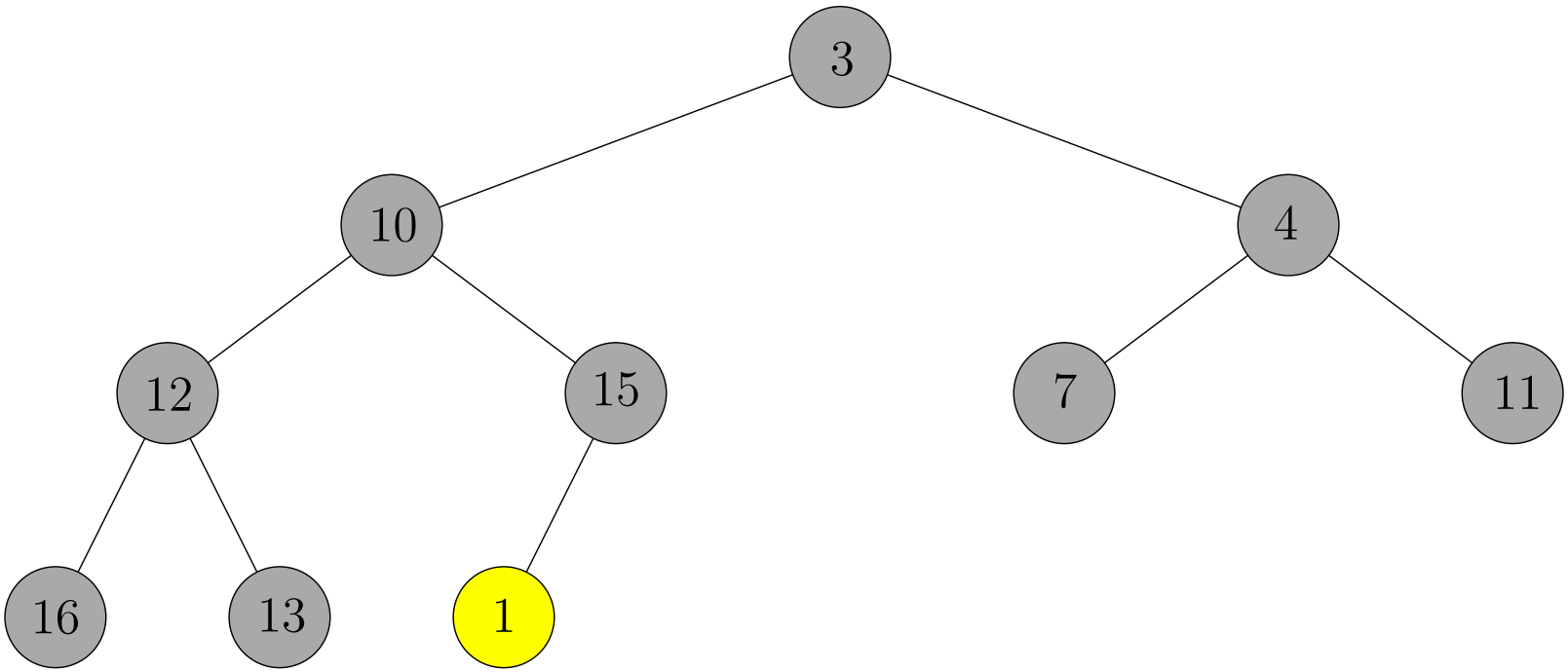
Insert

fixup



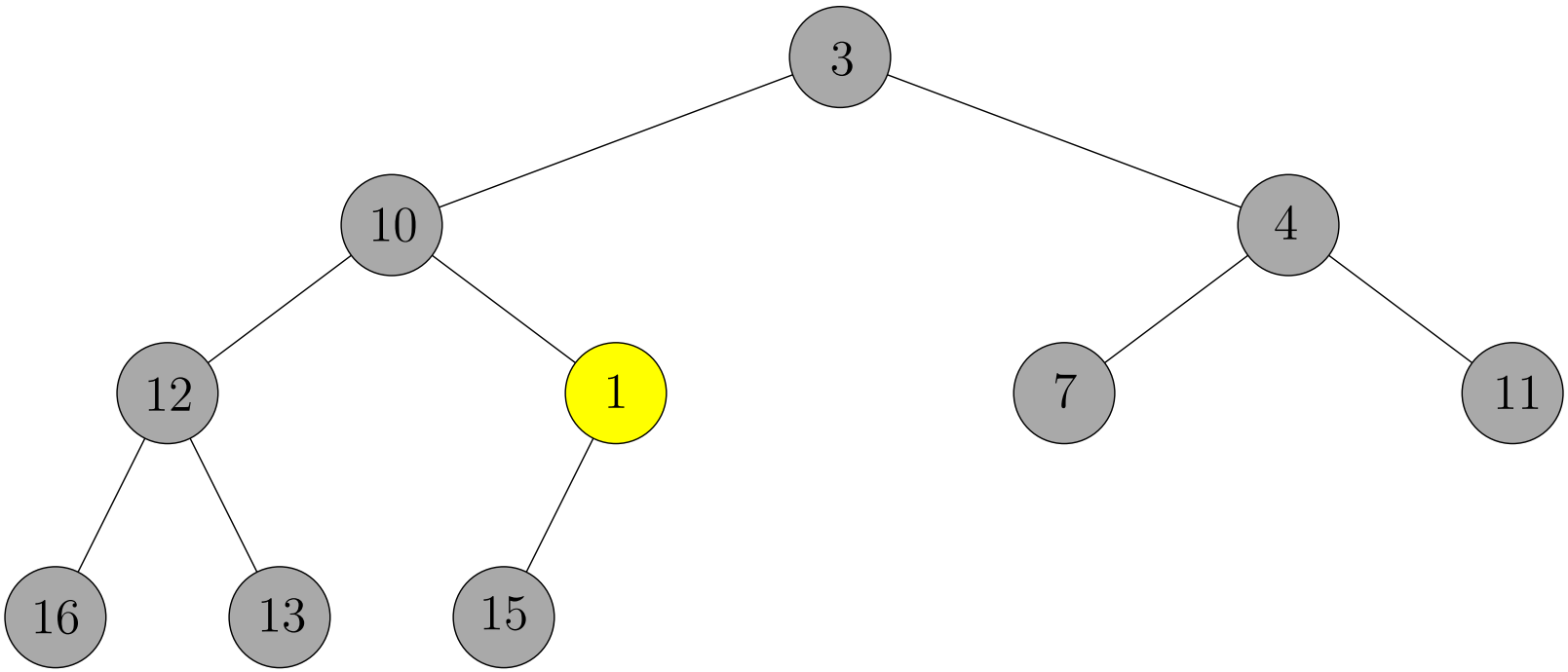
Insert

fixup



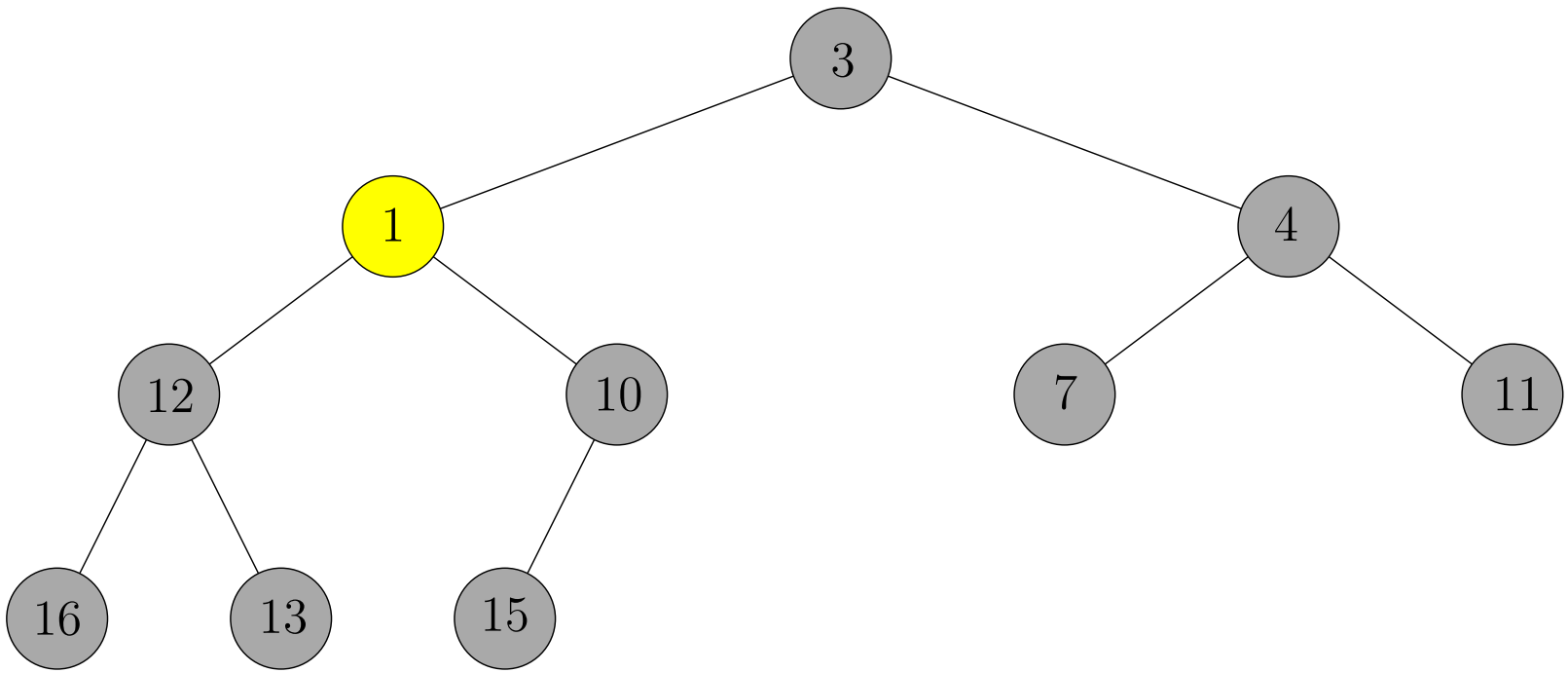
Insert

fixup



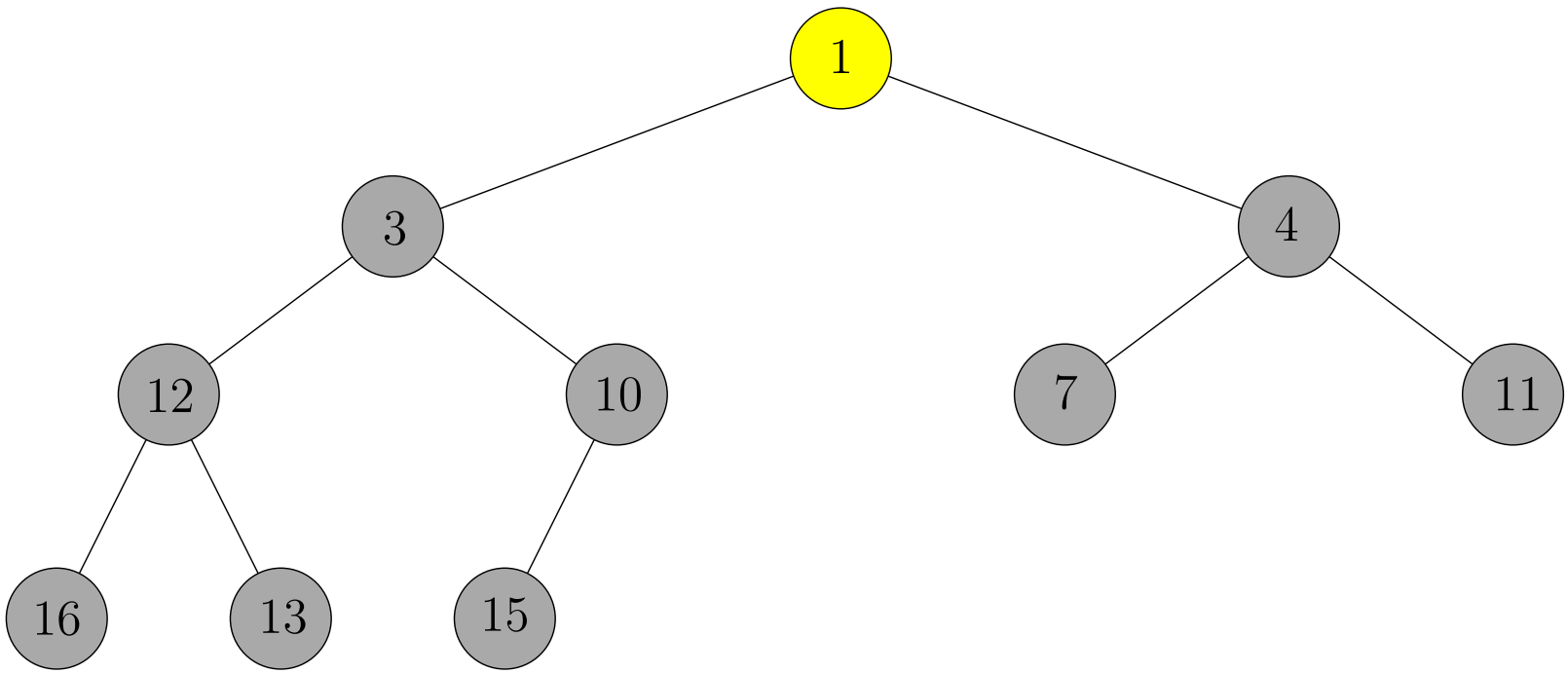
Insert

fixup



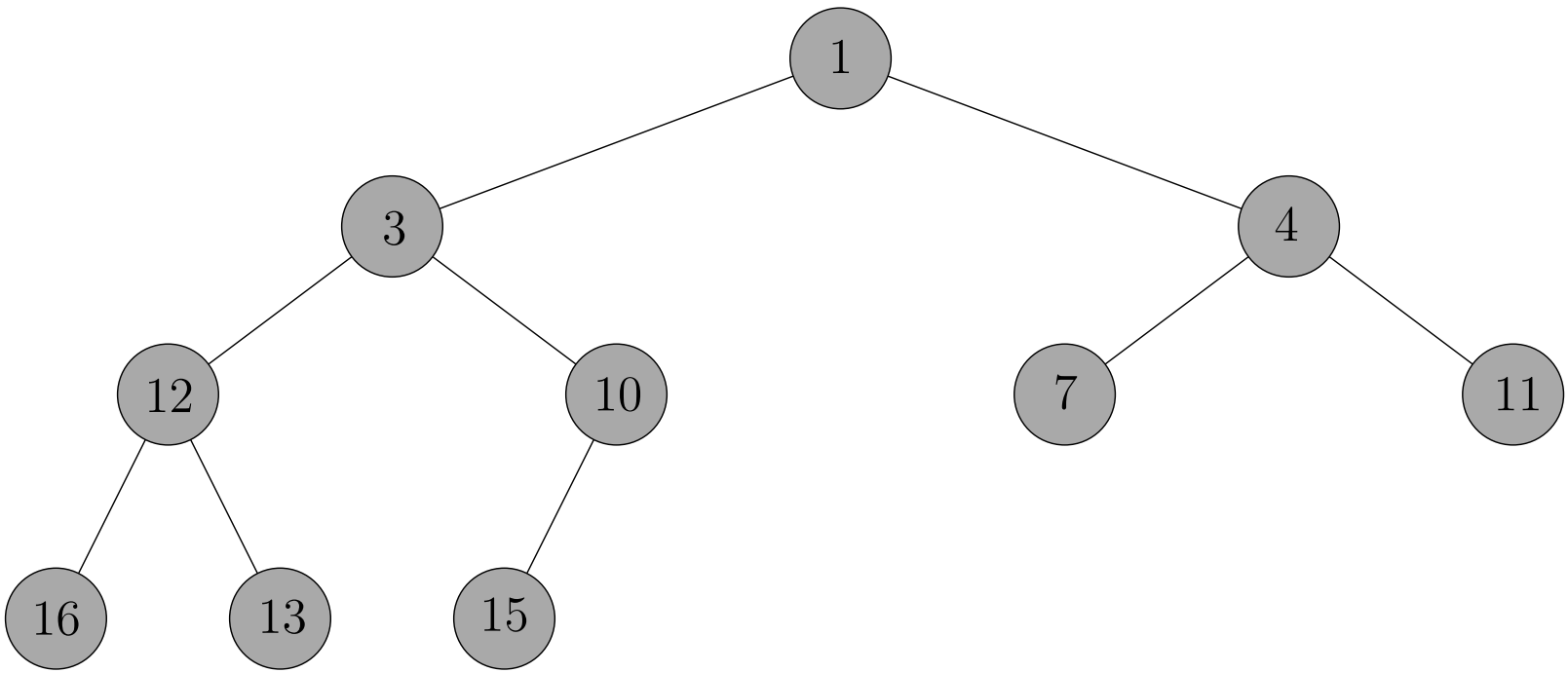
Insert

fixup



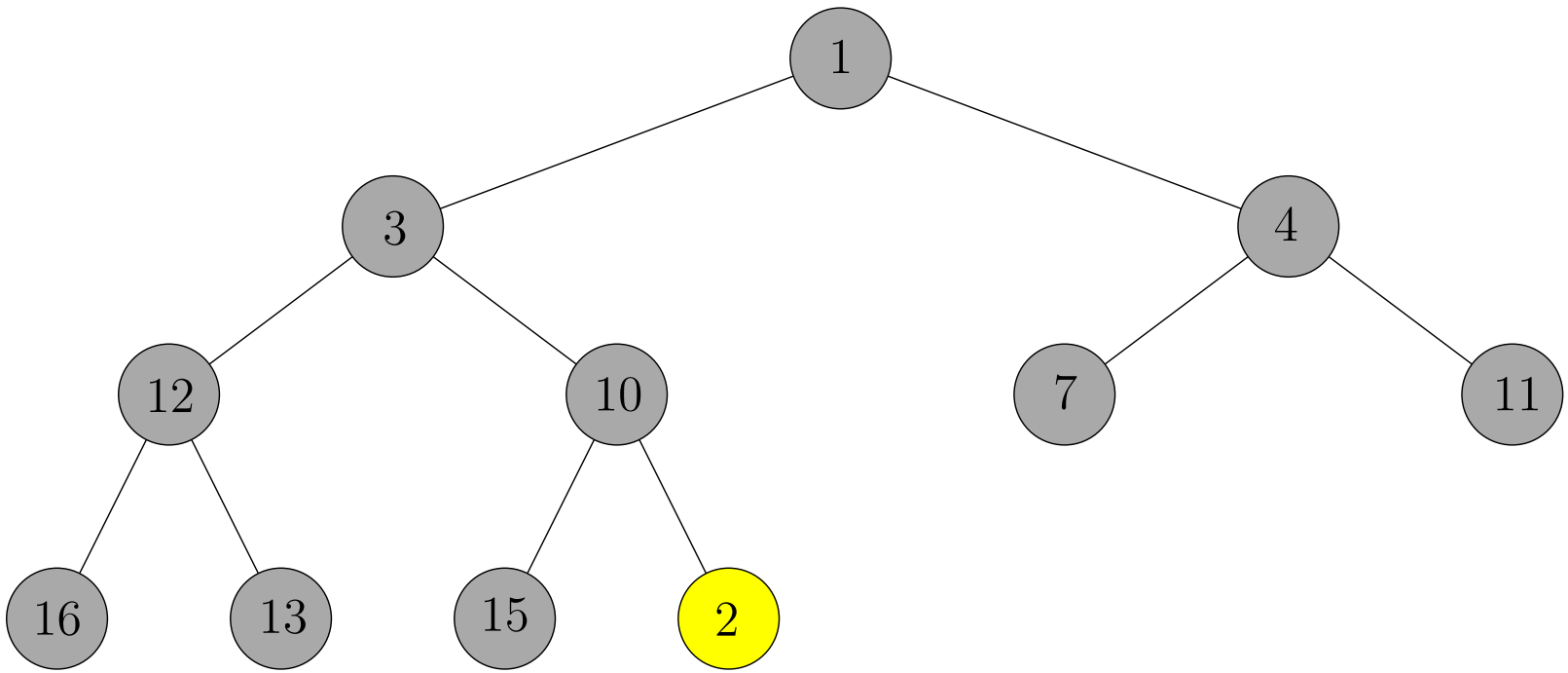
Insert

fixup



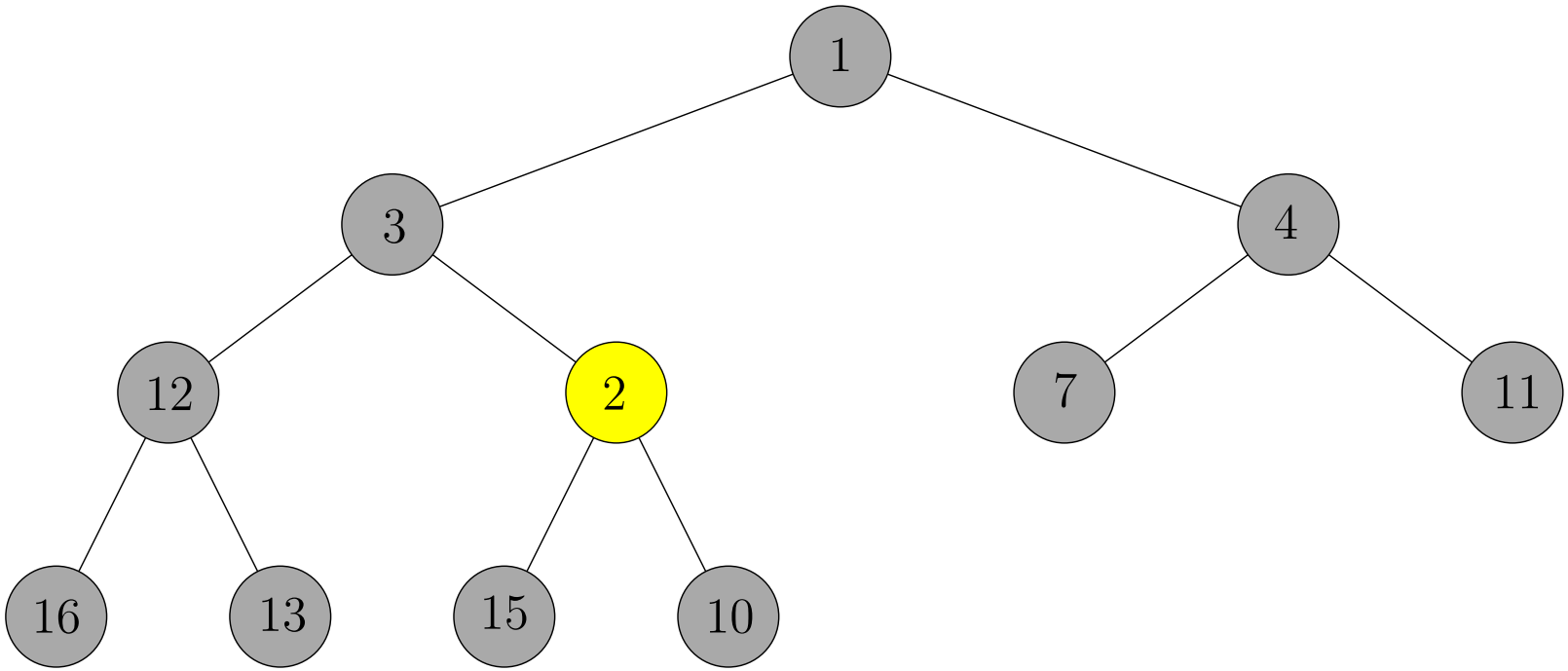
Insert

fixup



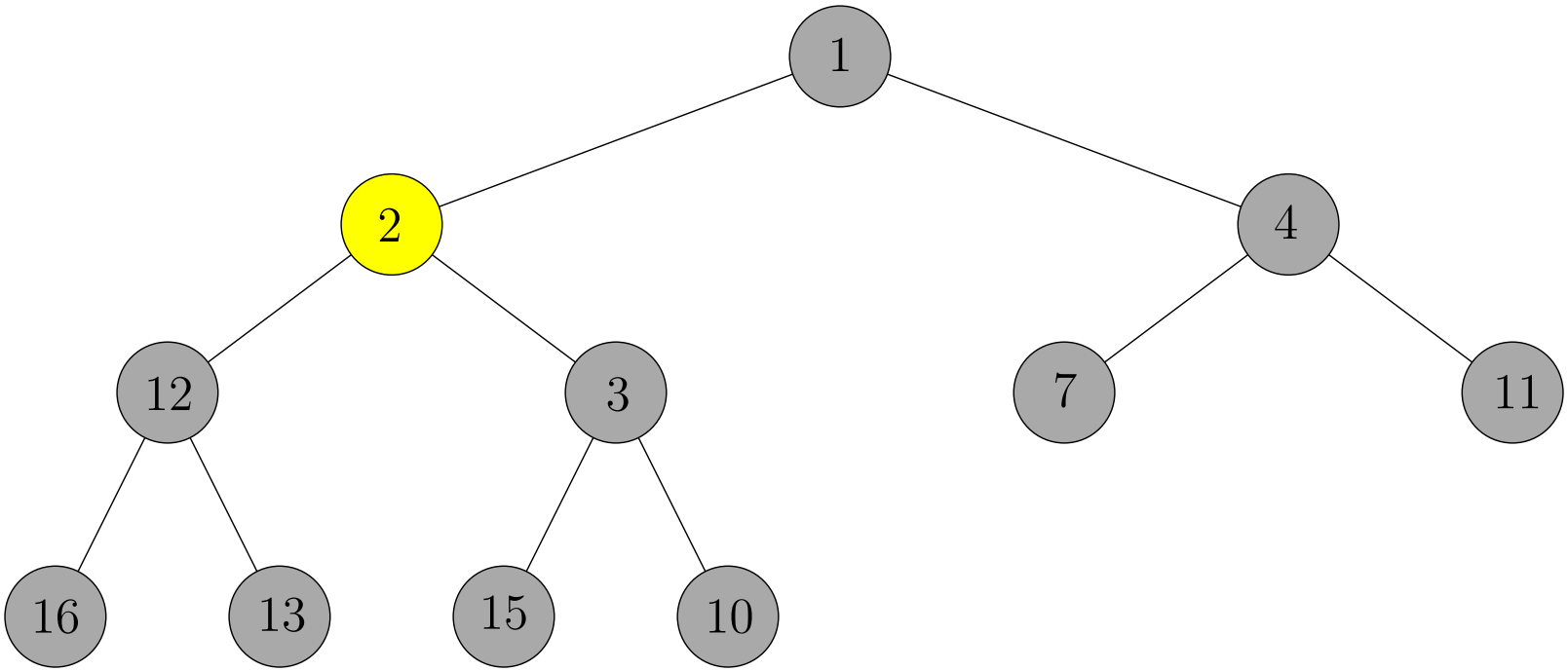
Insert

fixup



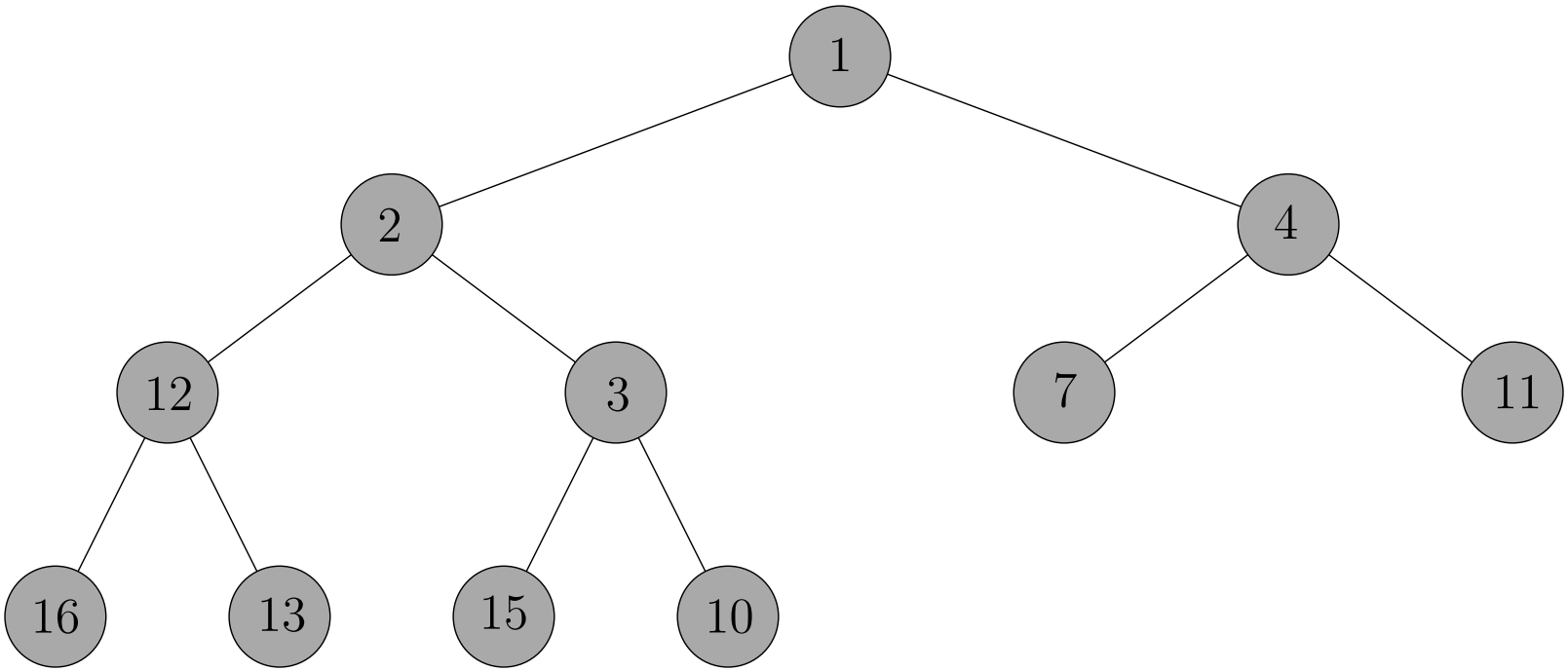
Insert

fixup



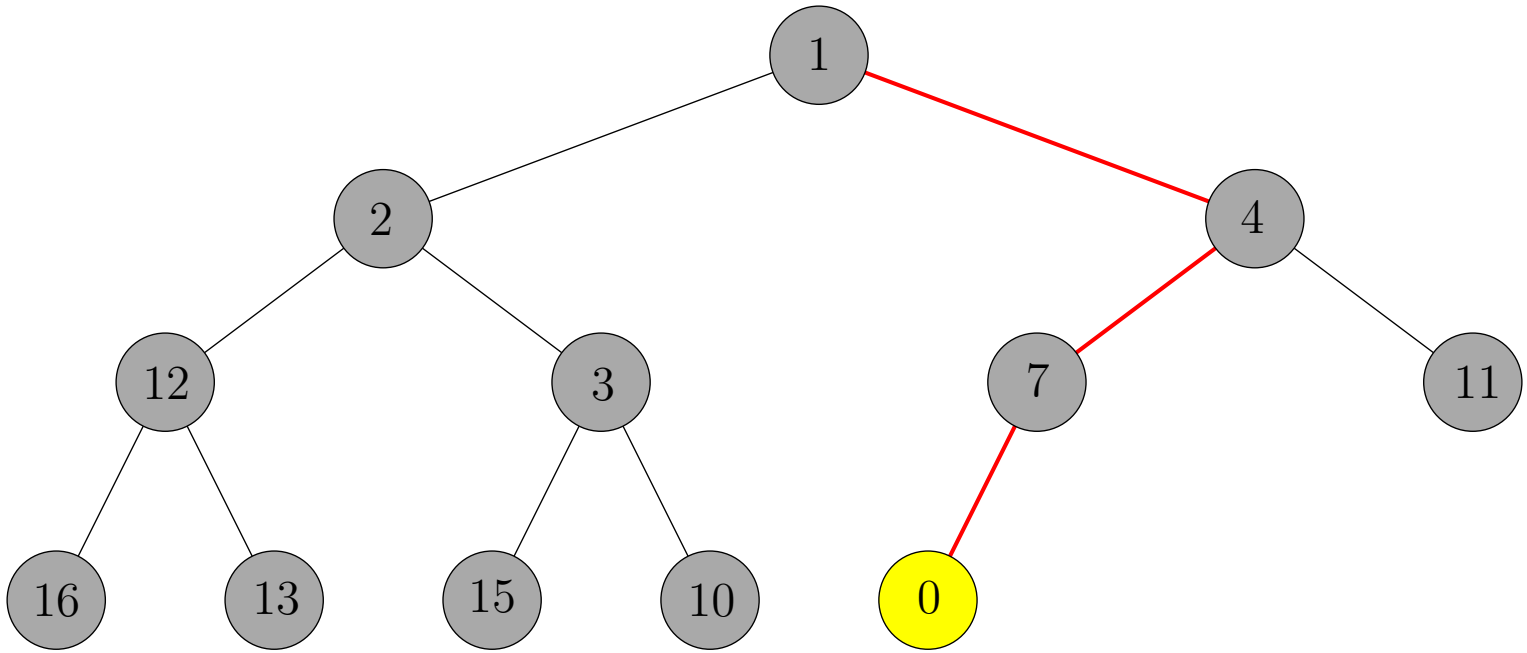
Insert

fixup



Insert

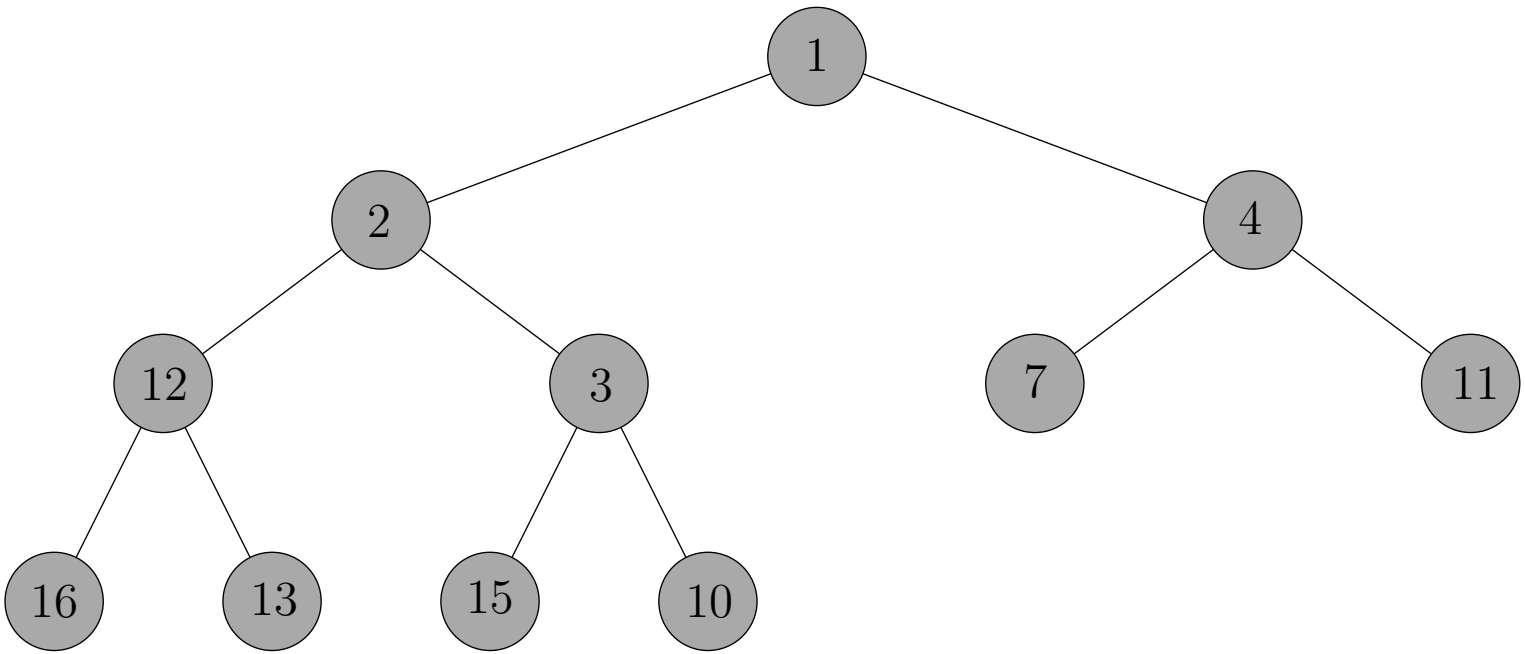
Time



The execution time is proportional to the of the tree, which is $O(\log n)$.

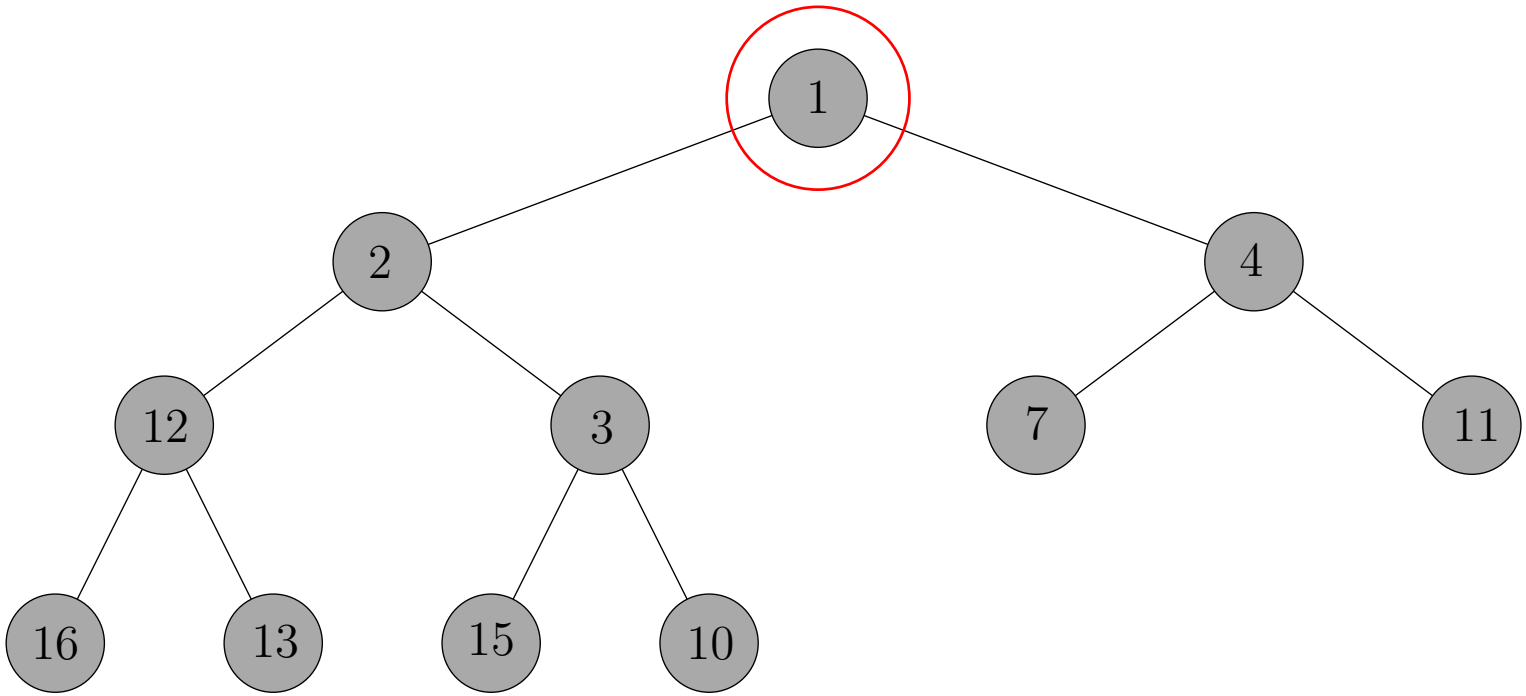
Delete Minimum

fixdown



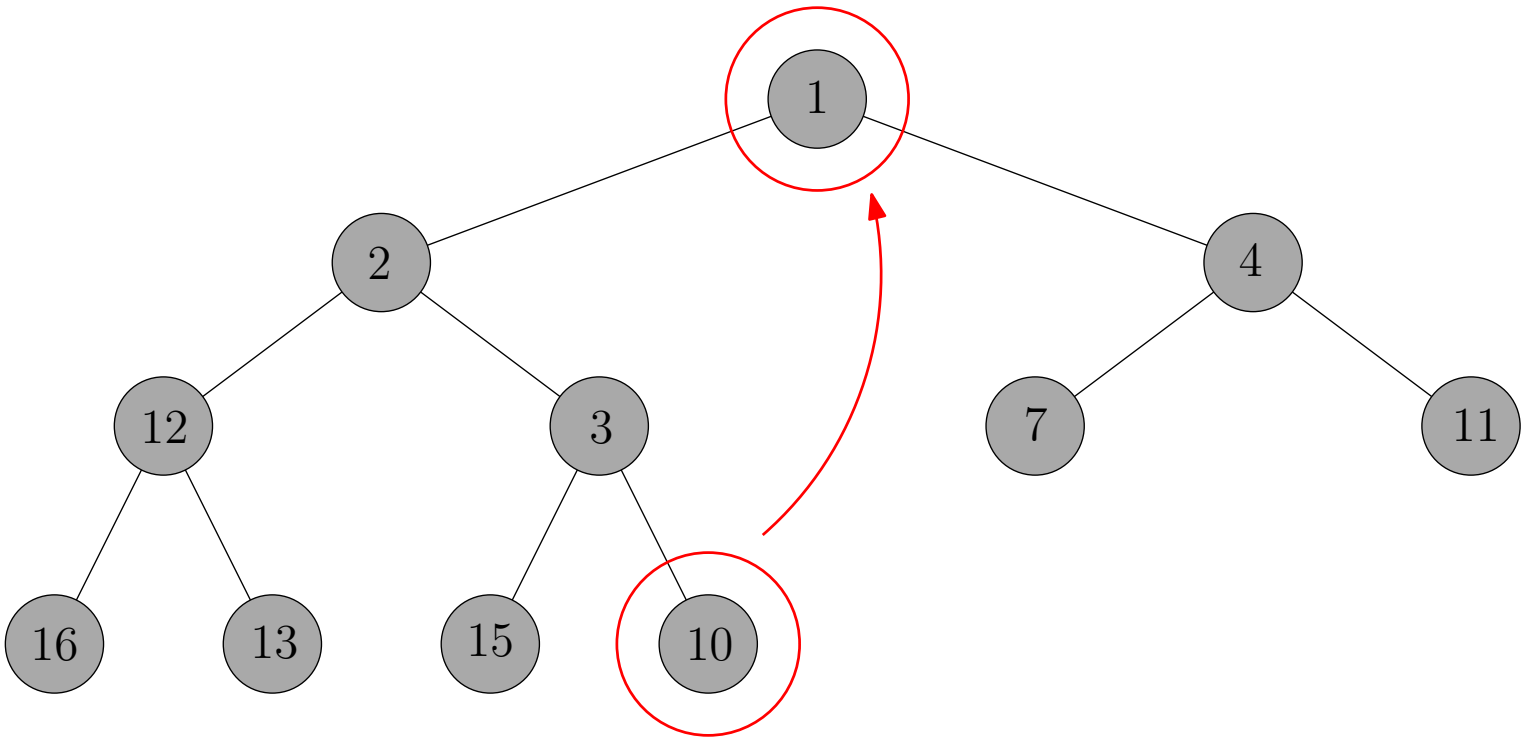
Delete Minimum

fixdown



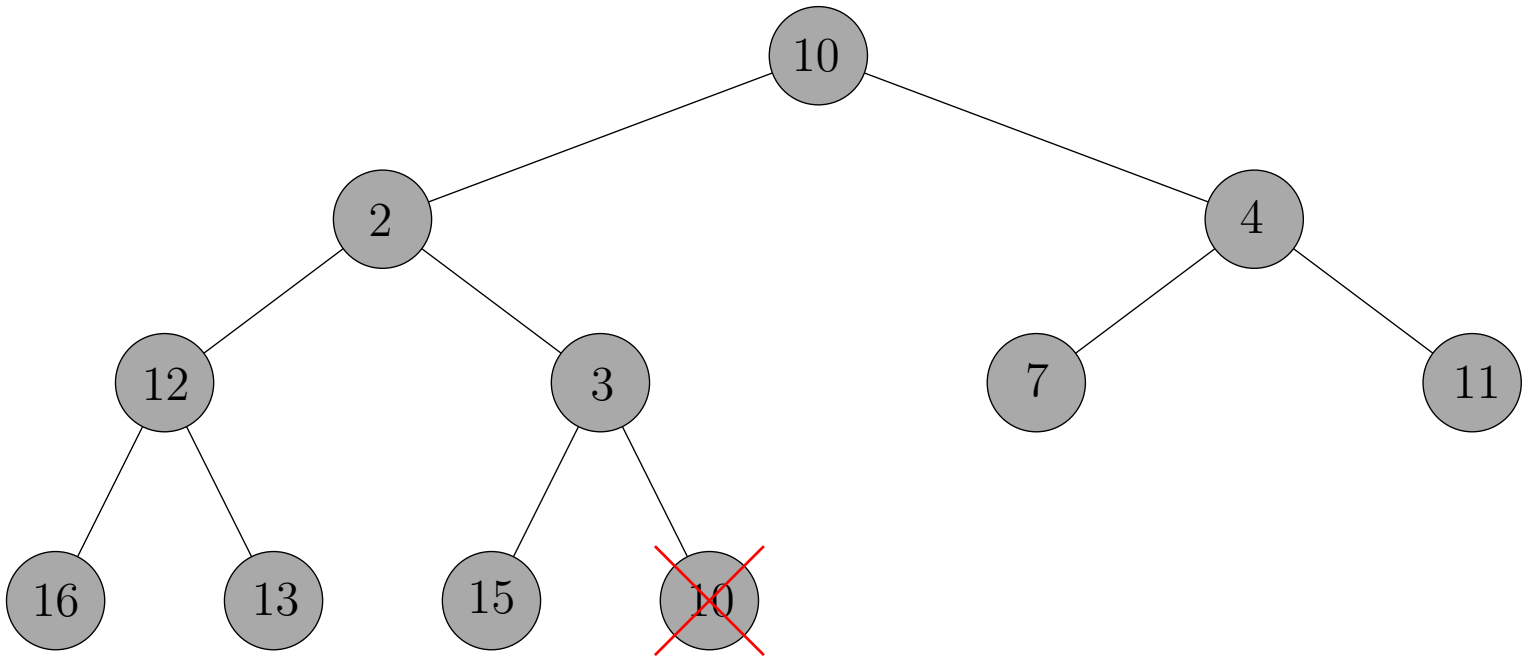
Delete Minimum

fixdown



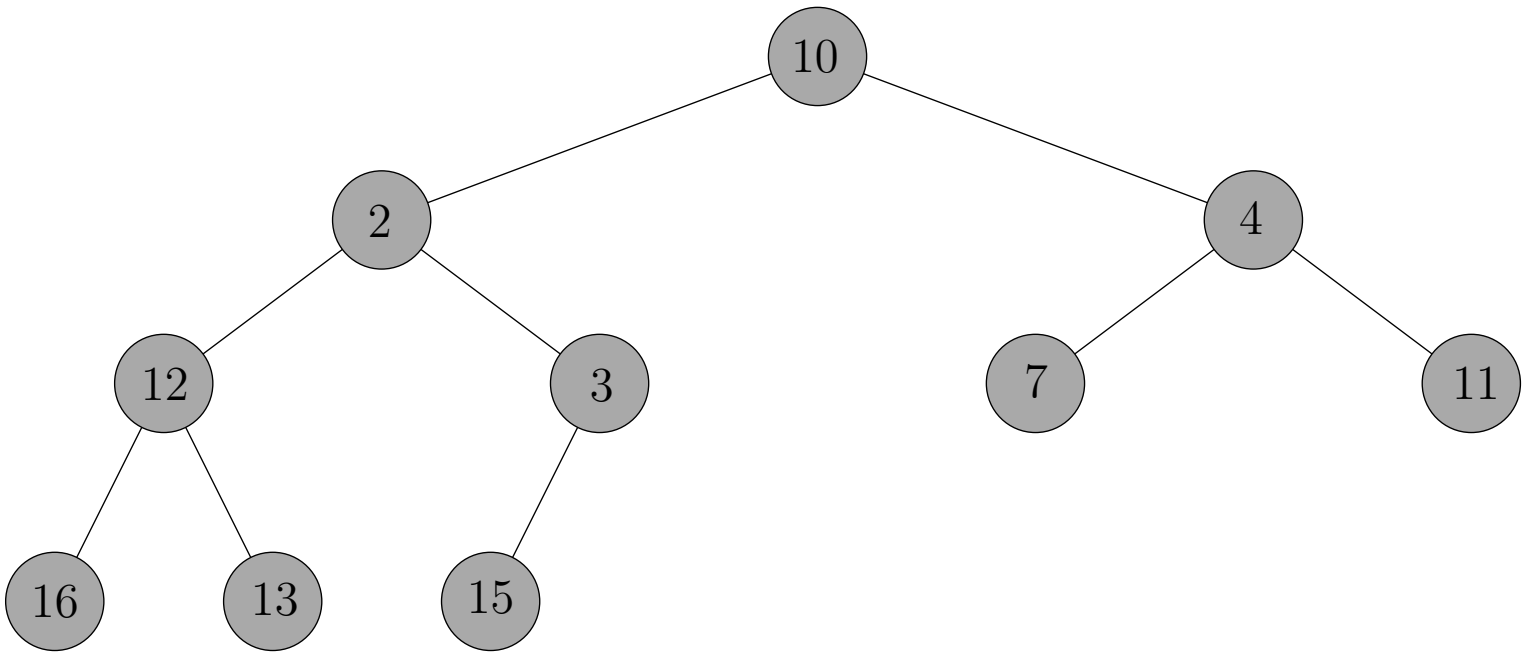
Delete Minimum

fixdown



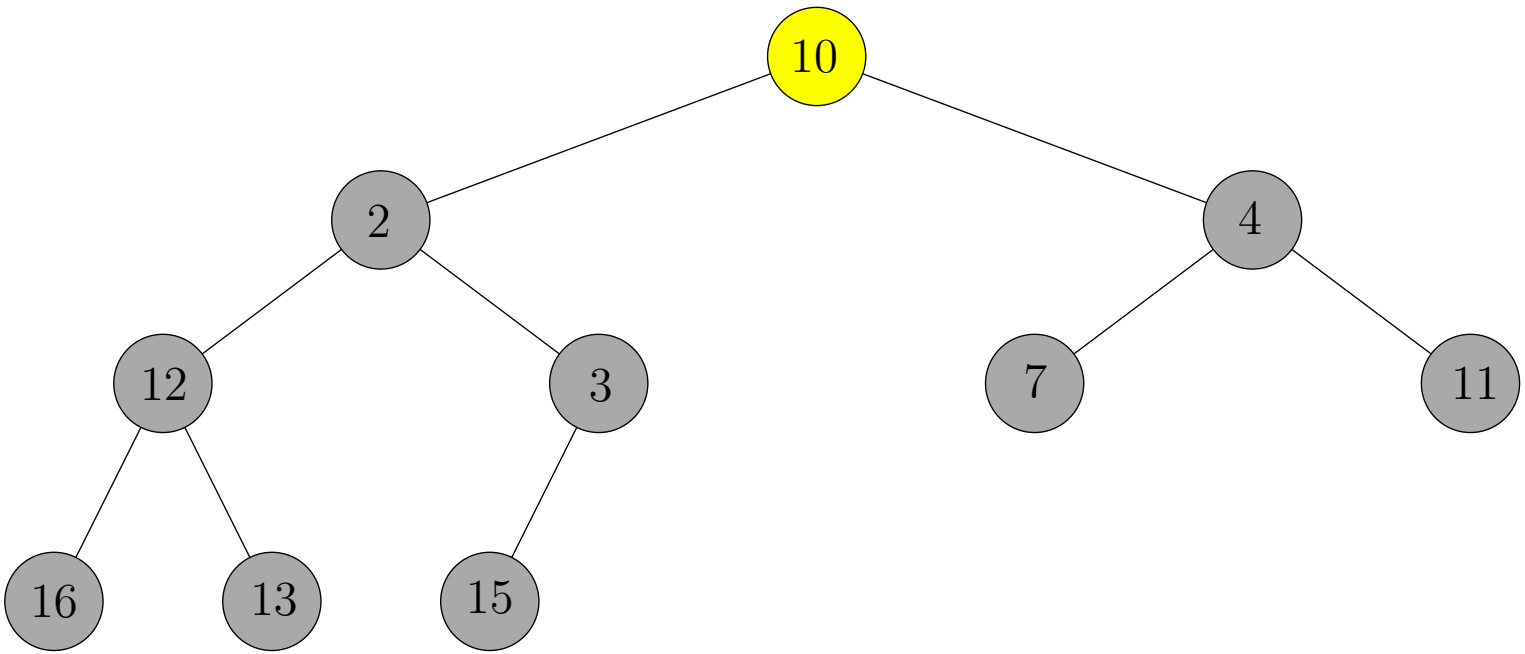
Delete Minimum

fixdown



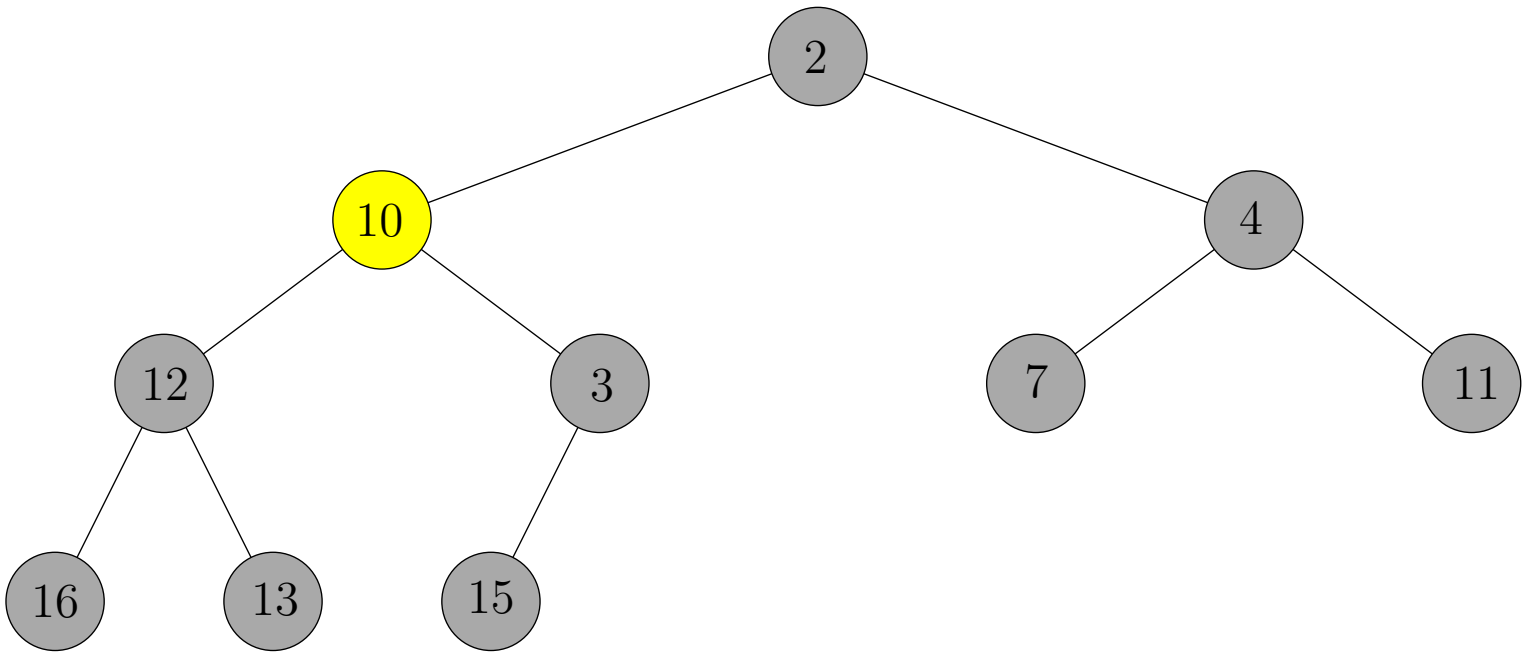
Delete Minimum

fixdown



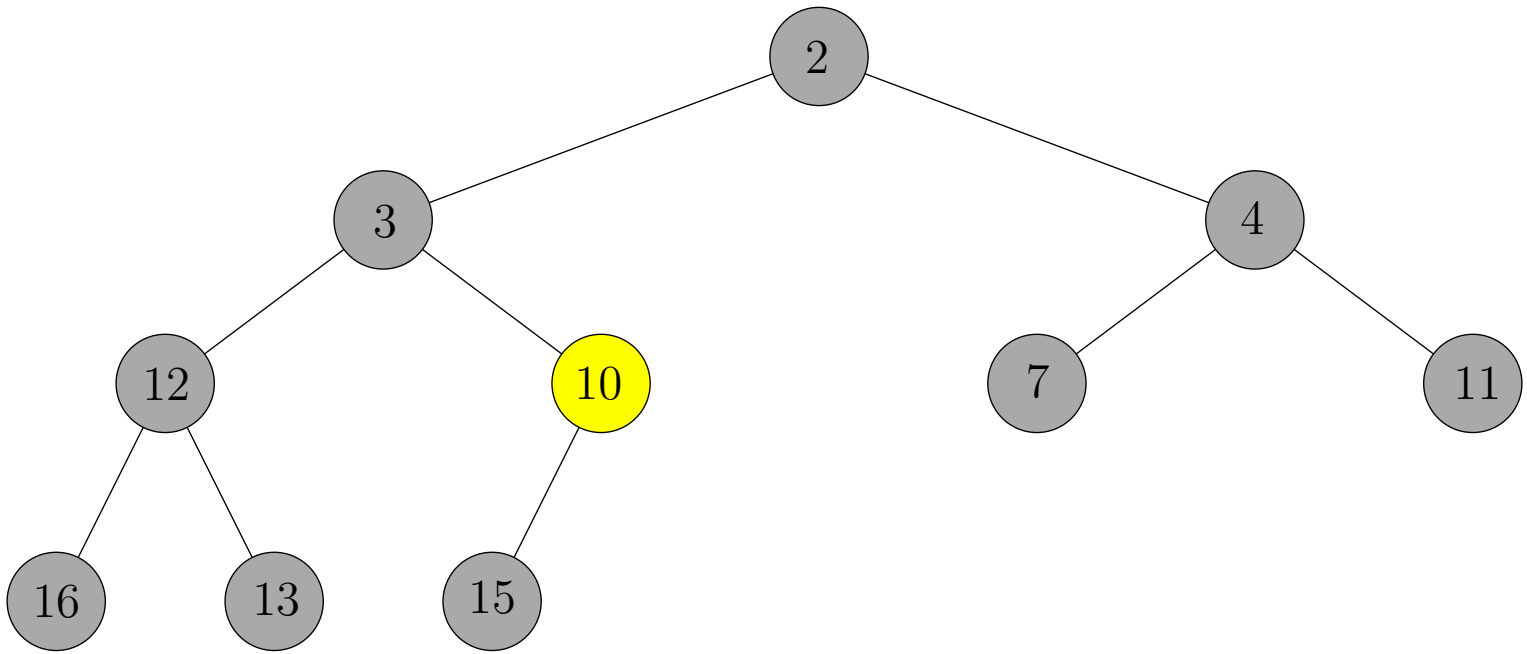
Delete Minimum

fixdown



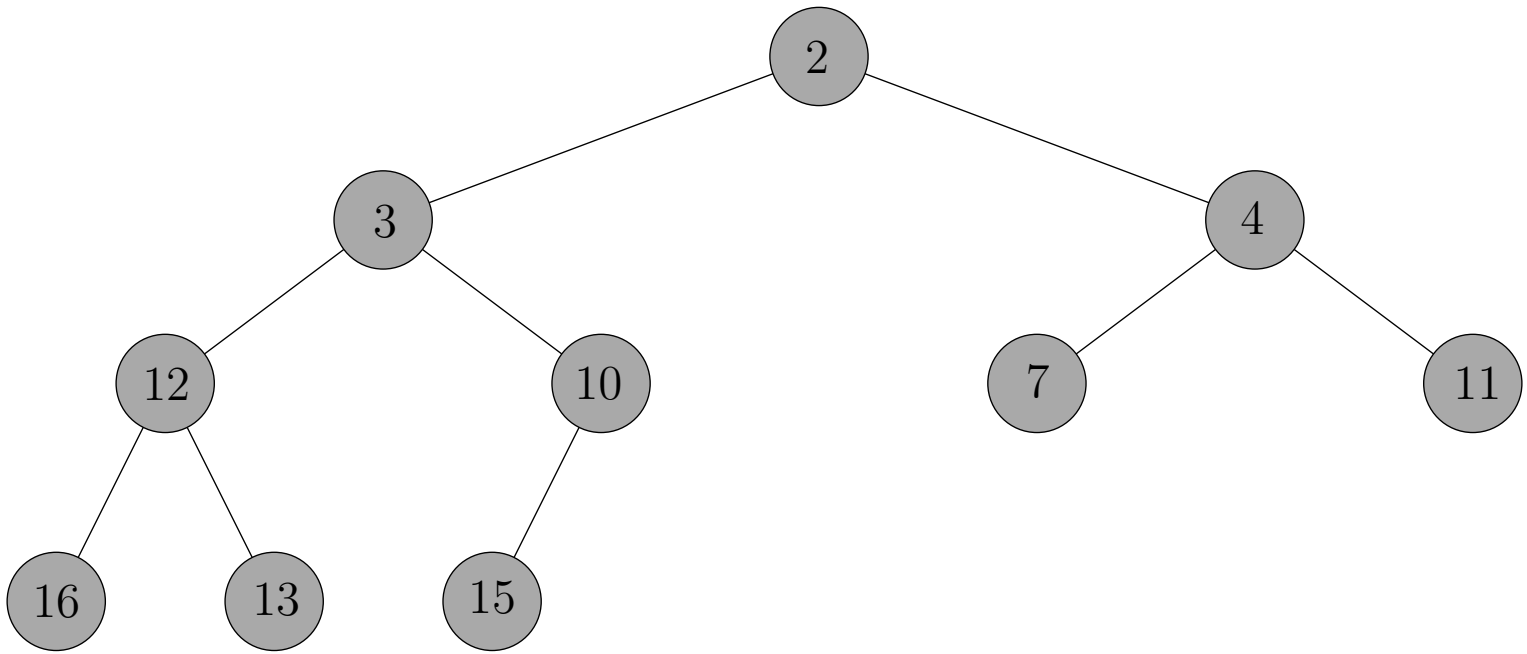
Delete Minimum

fixdown

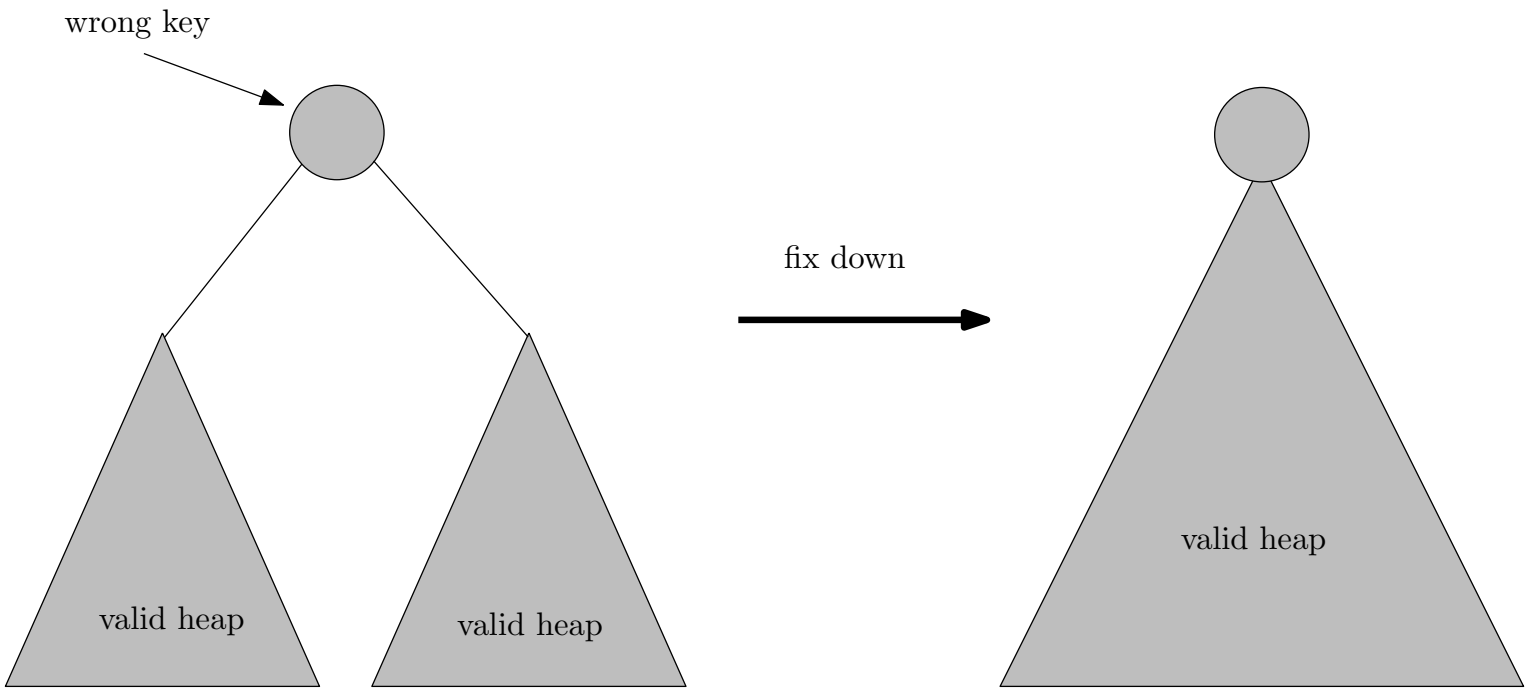


Delete Minimum

fixdown



Top-Down Fix

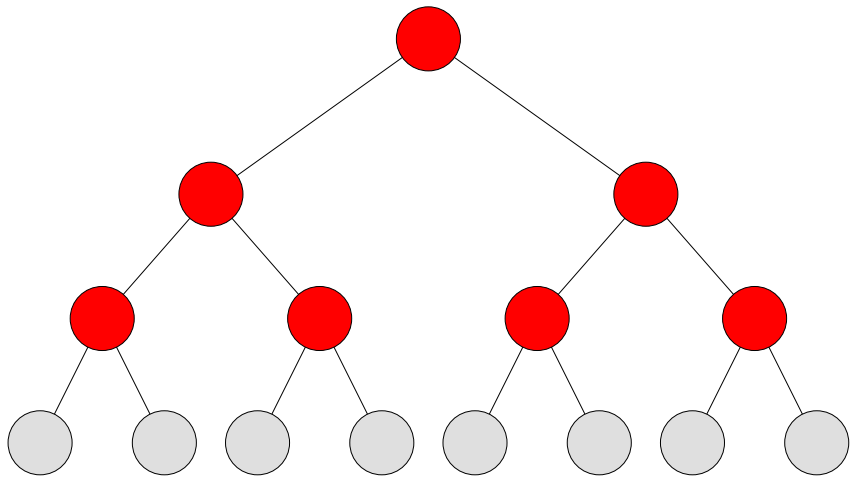


If the two subtrees of a node are already heaps, then one execution of a top-down fix will make the whole tree into a heap.

Build a Binary Heap from an Array

heapify

Consider an array with n elements which we would like to turn into a heap.

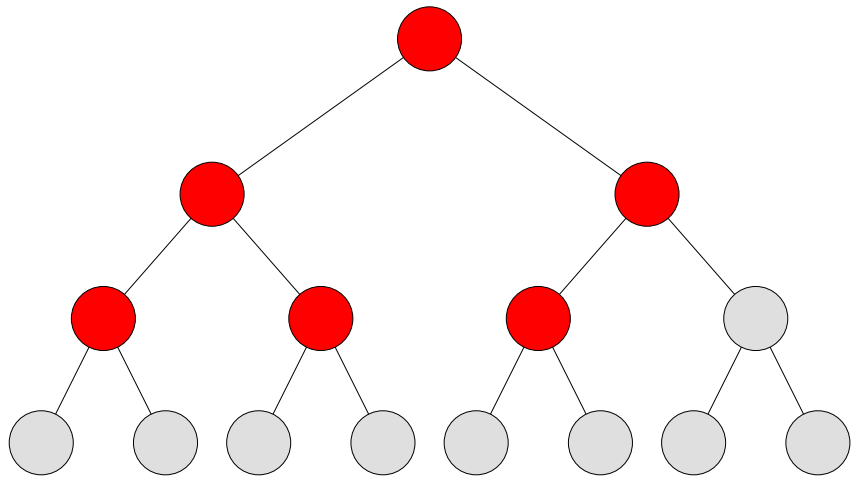


All the leafs are heaps. We can build the rest of the tree by doing "top-down" fixes. Start from the middle of the array and run "top-down" fixes for each element until we reach the root (the first element of the array).

Build a Binary Heap from an Array

heapify

Consider an array with n elements which we would like to turn into a heap.

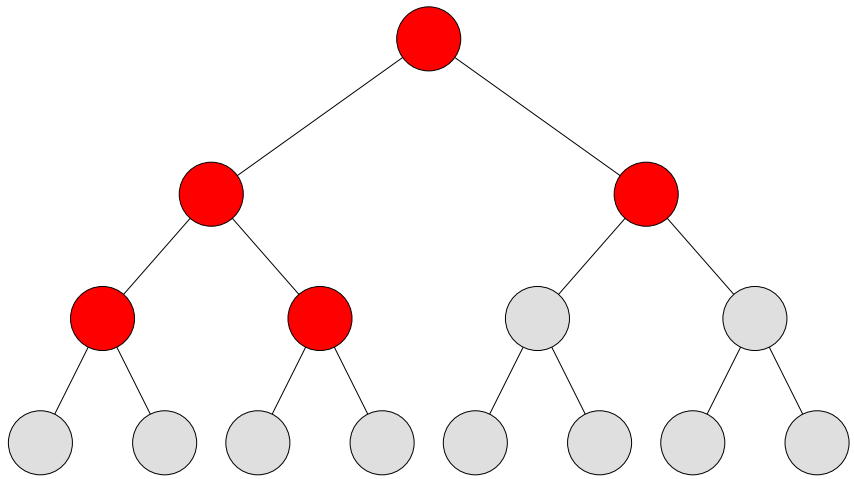


All the leafs are heaps. We can build the rest of the tree by doing "top-down" fixes. Start from the middle of the array and run "top-down" fixes for each element until we reach the root (the first element of the array).

Build a Binary Heap from an Array

heapify

Consider an array with n elements which we would like to turn into a heap.

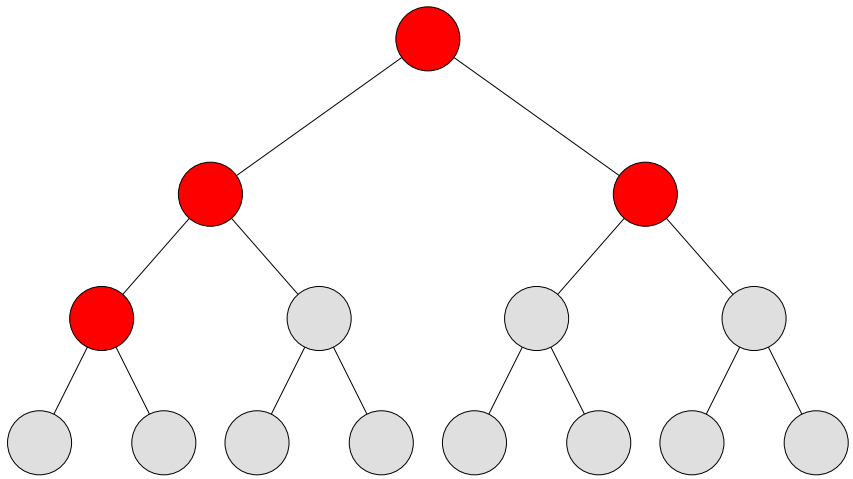


All the leafs are heaps. We can build the rest of the tree by doing "top-down" fixes. Start from the middle of the array and run "top-down" fixes for each element until we reach the root (the first element of the array).

Build a Binary Heap from an Array

heapify

Consider an array with n elements which we would like to turn into a heap.

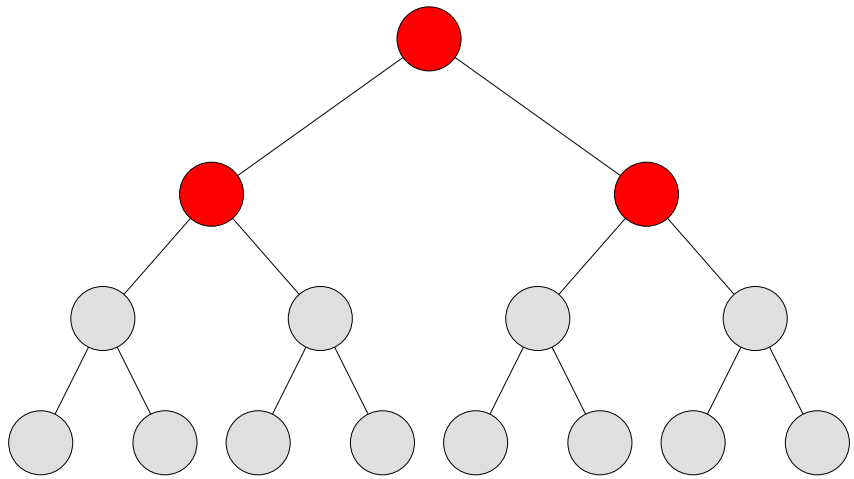


All the leafs are heaps. We can build the rest of the tree by doing "top-down" fixes. Start from the middle of the array and run "top-down" fixes for each element until we reach the root (the first element of the array).

Build a Binary Heap from an Array

heapify

Consider an array with n elements which we would like to turn into a heap.

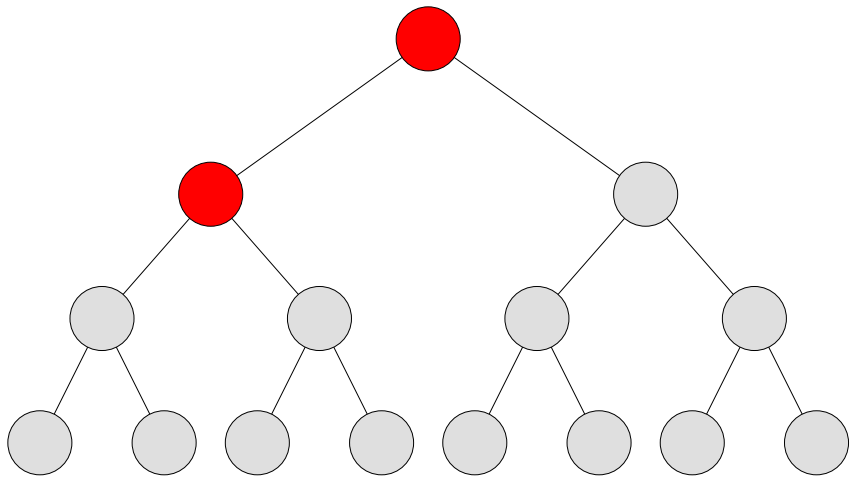


All the leafs are heaps. We can build the rest of the tree by doing "top-down" fixes. Start from the middle of the array and run "top-down" fixes for each element until we reach the root (the first element of the array).

Build a Binary Heap from an Array

heapify

Consider an array with n elements which we would like to turn into a heap.

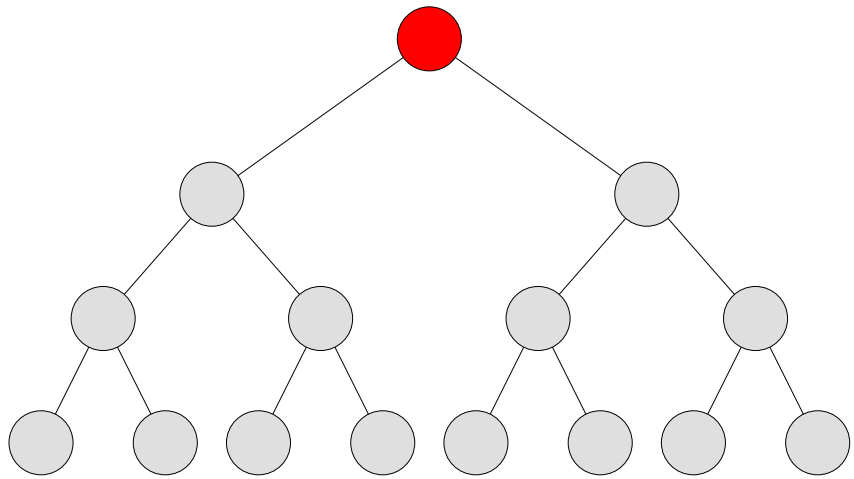


All the leafs are heaps. We can build the rest of the tree by doing "top-down" fixes. Start from the middle of the array and run "top-down" fixes for each element until we reach the root (the first element of the array).

Build a Binary Heap from an Array

heapify

Consider an array with n elements which we would like to turn into a heap.

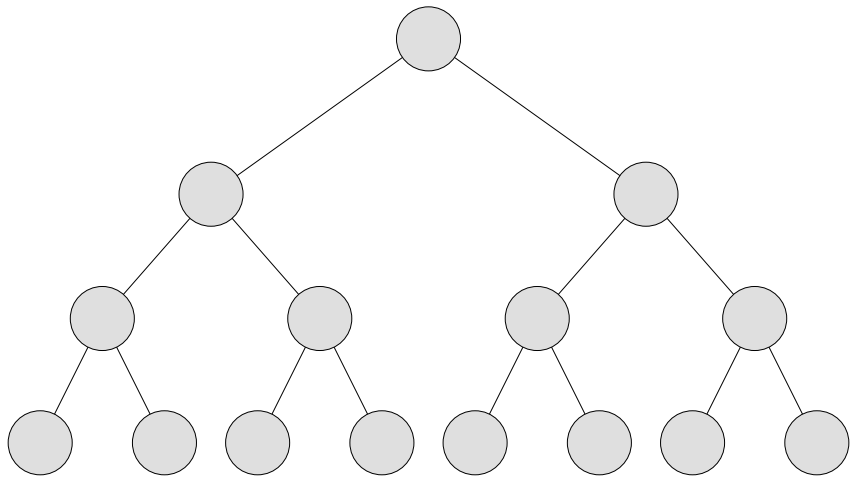


All the leafs are heaps. We can build the rest of the tree by doing "top-down" fixes. Start from the middle of the array and run "top-down" fixes for each element until we reach the root (the first element of the array).

Build a Binary Heap from an Array

heapify

Consider an array with n elements which we would like to turn into a heap.



All the leafs are heaps. We can build the rest of the tree by doing "top-down" fixes. Start from the middle of the array and run "top-down" fixes for each element until we reach the root (the first element of the array).

Build a Binary Heap from an Array

heapify

How much does this procedure cost us?

- The array has n element and we perform "top-down" fixes to at most $k = \lfloor \log n \rfloor$ levels.
- Every level i has at most 2^i nodes
- The up-down fix starting from level i has cost at most $\mathcal{O}(k - i)$.

Adding up:

$$\mathcal{O}\left(\sum_{0 \le i < k} 2^i (k - i)\right) = \mathcal{O}\left(2^k \sum_{0 \le i < k} \frac{k - i}{2^{k-i}}\right) = \mathcal{O}\left(2^k \sum_{j \ge 1} \frac{j}{2^j}\right) = \mathcal{O}(n)$$

Sorting with a Binary Heap

Consider an array with n integers. We would like to output the integers in non-decreasing order.

Algorithms

- create a heap from the array
- while the heap is not empty, remove and print its smallest element

This algorithm is called *Heap-Sort* and puts n numbers in order in time $\mathcal{O}(n \log n)$.