Heap, Priority Queue

Kriengsak Treeprapin

Agenda

- Heap
- Misc

Heap

Heap

Heap is a specialized tree-based data structure that satisfies the heap property:

"if P is a parent node of C, then the key (the value) of node P is greater than the key of node C"

Heap and Priority Queue

A heap can be used as a priority queue:

the highest priority item is at the root and is trivially extracted.
But if the root is deleted, we are left with two sub-trees and we must efficiently re-create a single tree with the heap property.

The value of the heap structure is extract the highest priority item and insert a new one in O(logn) time.

Operations in Heap

Insert : Adding new node to the heap.

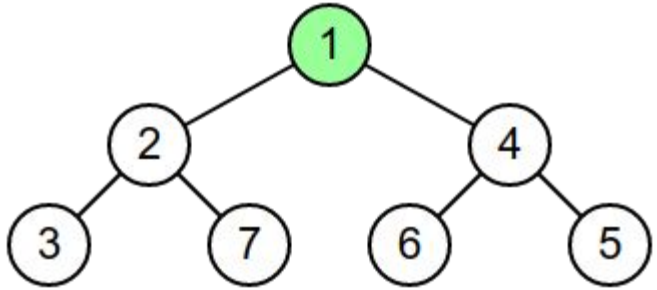
Find-min (max) : finding minimum (or maximum) node.

Delete-min (max) : removing the root node of heap.

Decrease-key : updating a key within a heap, respectively.

Merge : joining two heaps to form a valid new heap containing all the elements of both heaps.

Binary Heap

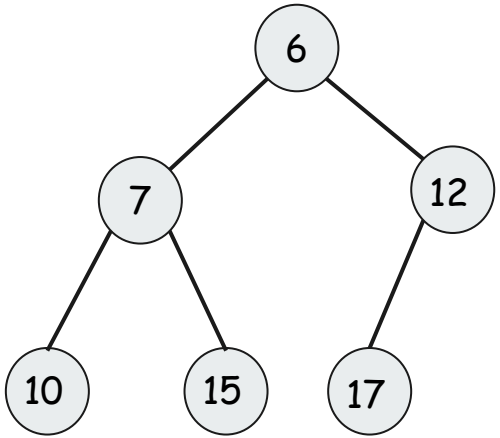


Binary Heap

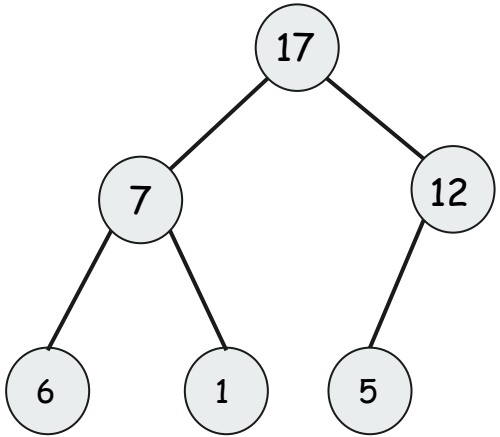
A binary heap is a complete binary tree which satisfies the heap ordering property. The ordering can be one of two types:

- **the min-heap property**: the value of each node is greater than or equal to the value of its parent, with the minimum-value element at the root.
- **the max-heap property**: the value of each node is less than or equal to the value of its parent, with the maximum-value element at the root.

Binary Heap



min-heap



max-heap

Binary Heap

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.

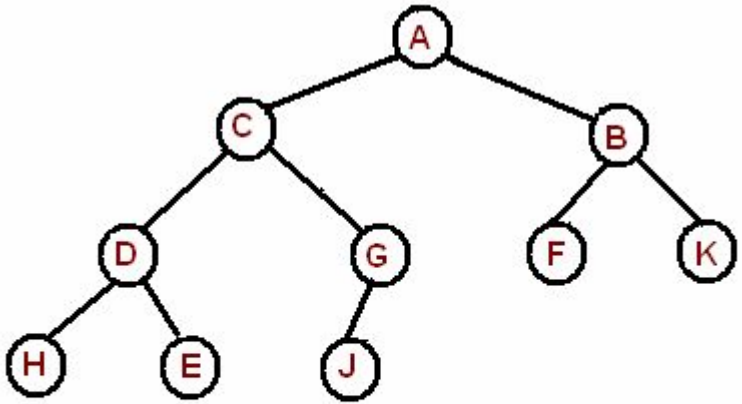
Since a heap is a complete binary tree, it has a smallest possible height - **a heap with N nodes always has O(log N) height**.

Binary Heap in Array Implementation

A complete binary tree can be uniquely represented by storing its level order traversal in an array.

Consider k-th element of the array,

its left child is located at 2*k index
its right child is located at 2*k+1. index
its parent is located at k/2 index



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

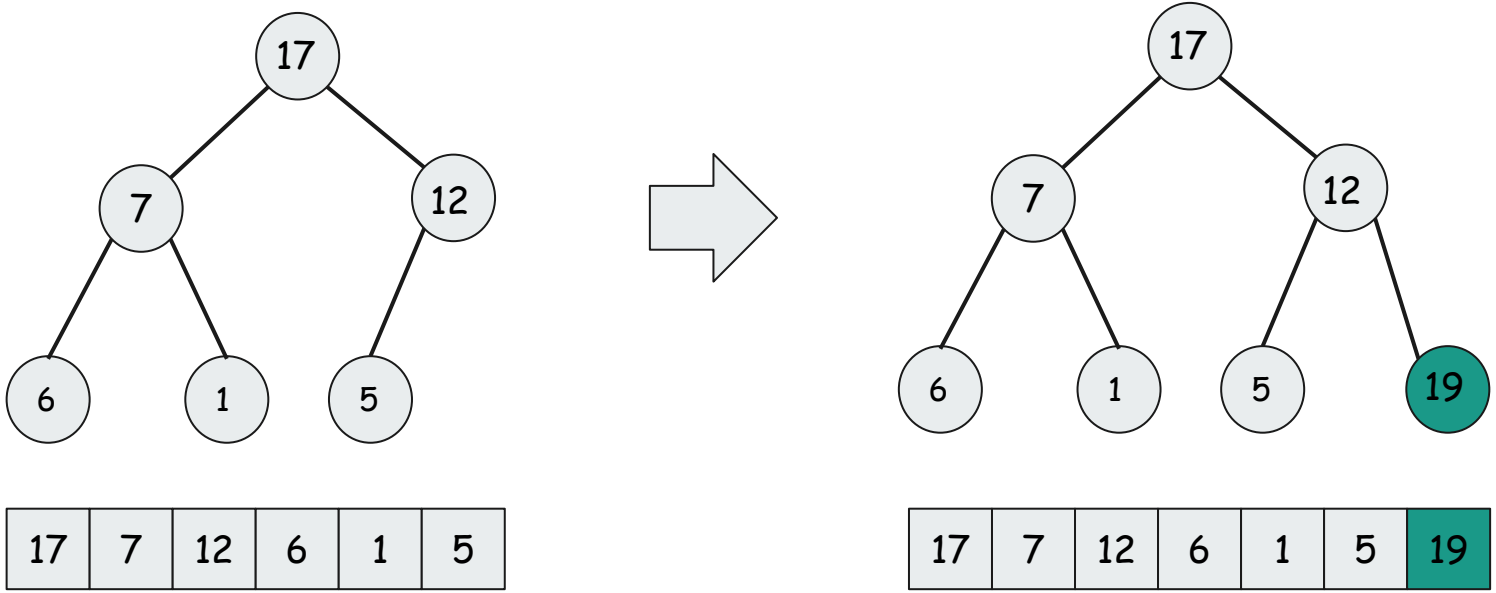
Insert operation in Binary Heap

Algorithm :

1. The new element is initially appended to the end of the heap (as the last element of the array).
2. Repair heap property by comparing the added element with its parent and swapping positions with the parent.
3. The comparison is repeated until the parent is larger than or equal to the element.

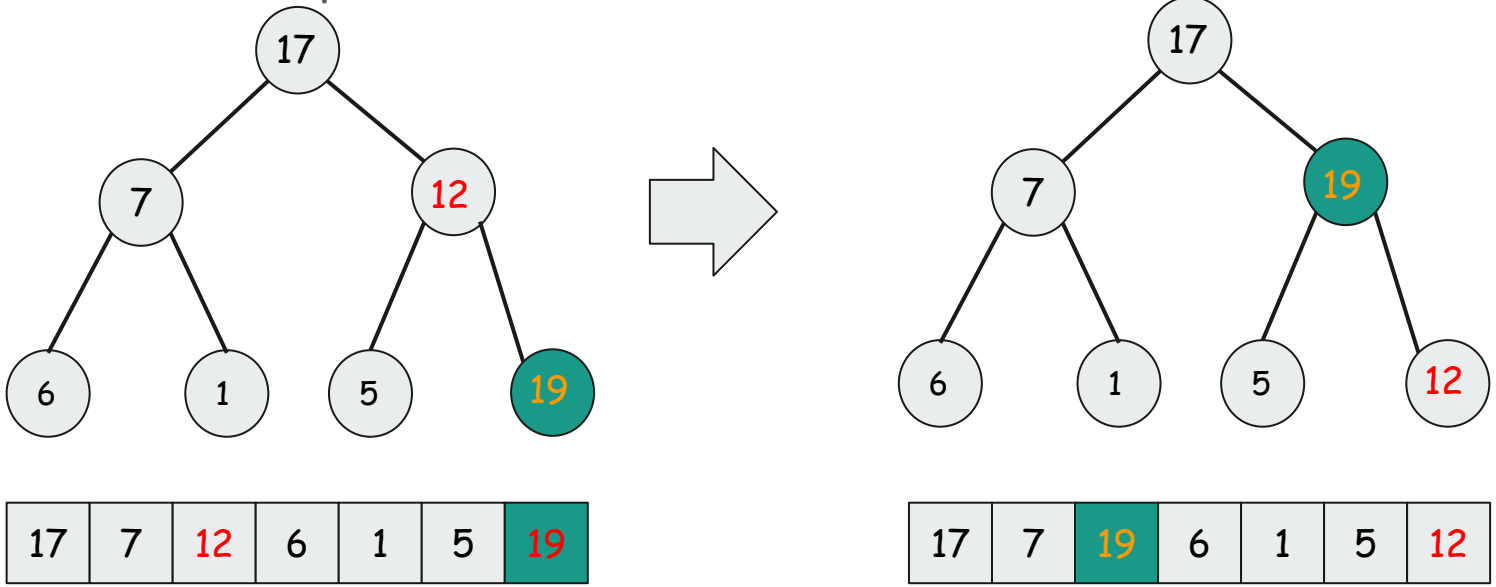
Insert operation in Binary Heap

Ex. Insert 19 : Append new element to the end of the heap.



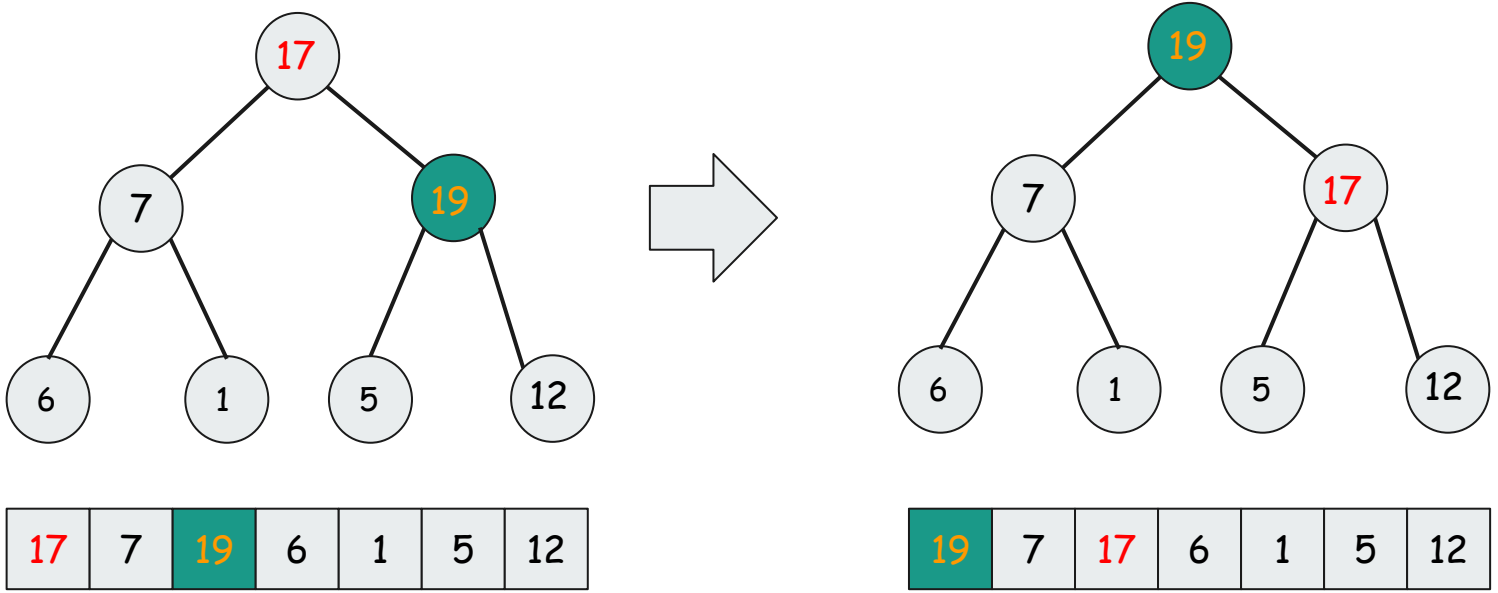
Insert operation in Binary Heap

Ex. Insert 19 : Repair heap property by comparing the added element with its parent.



Insert operation in Binary Heap

Ex. Insert 19 : repeat comparison until the parent is larger than or equal to the element.



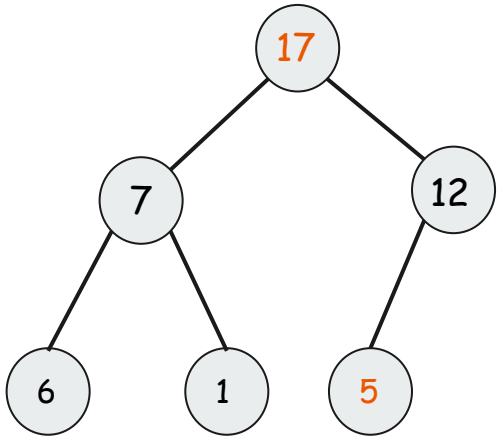
Delete operation in Binary Heap

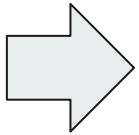
Algorithm :

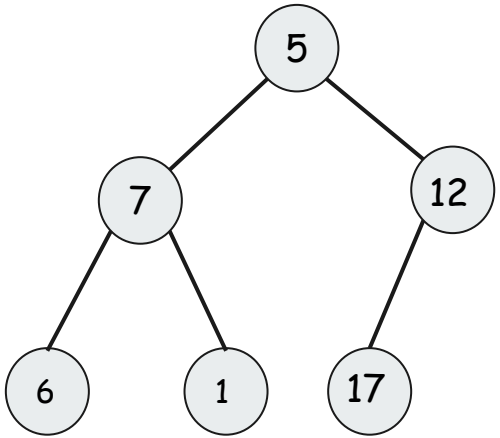
1. Remove the root and replace it with the last element of the heap
2. Restore the heap property by percolating down. Similar to insertion.

Delete operation in Binary Heap

Ex. Delete : Remove the root and replace it with the last element.





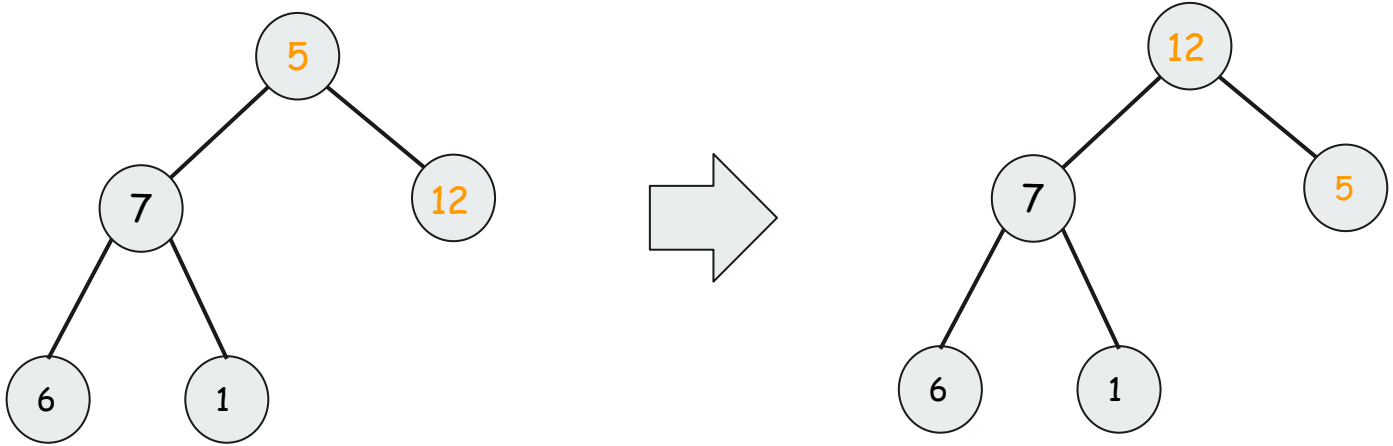


17	7	12	6	1	5

5	7	12	6	1	17

Delete operation in Binary Heap

Ex. Delete : Remove and Restore the heap property.



5	7	12	6	1

12	7	5	6	1

Other operations in Binary Heap

Decrease key : update key and restore heap property.

Merge : remake heap from two arrays.

Performance in Binary Heap

find-min (max) : $O(1)$

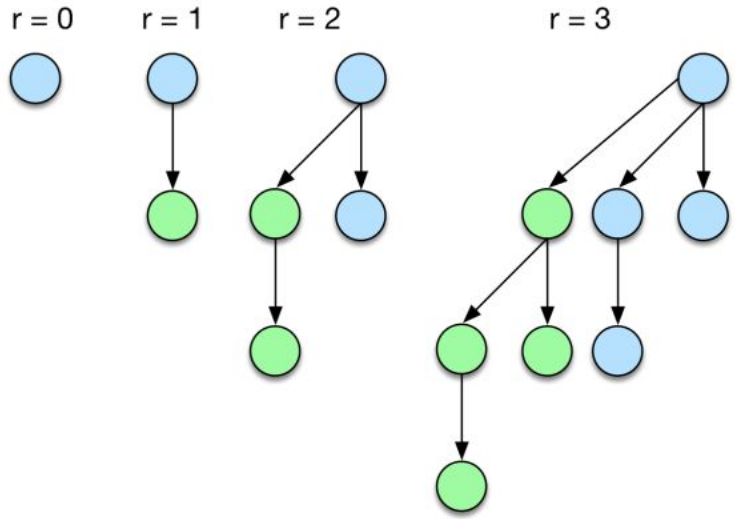
delete-min (max) : $O(\log N)$

insert : $O(\log N)$

decrease-key : $O(\log N)$

merge : $O(N)$

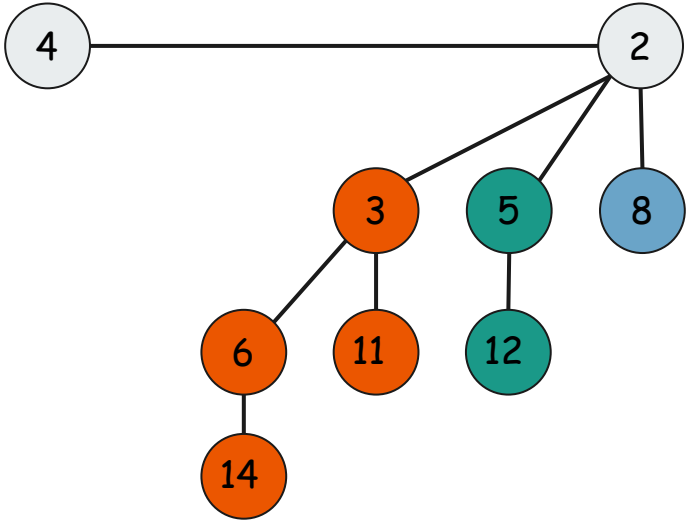
Binomial Heap



Binomial Heap

Provides faster union or merge operation together with other operations provided by Binary Heap.

A Binomial Heap is a collection of **Binomial Trees**.

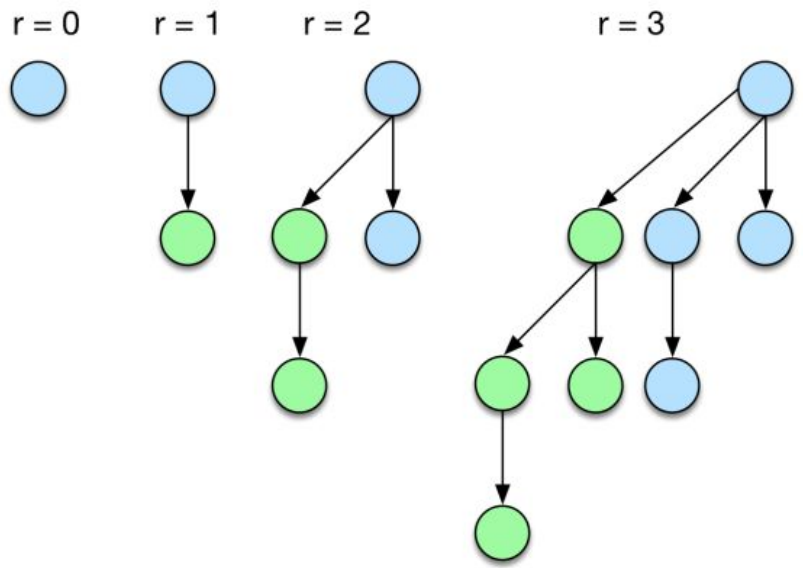


Binomial Tree

A binomial tree of order 0 is a single node

A binomial tree of order k has a root node whose children are roots of binomial trees of orders k–1, k–2, …, 2, 1, 0 (in this order).

A binomial tree of order k has 2^k nodes, height k.



Binomial Tree

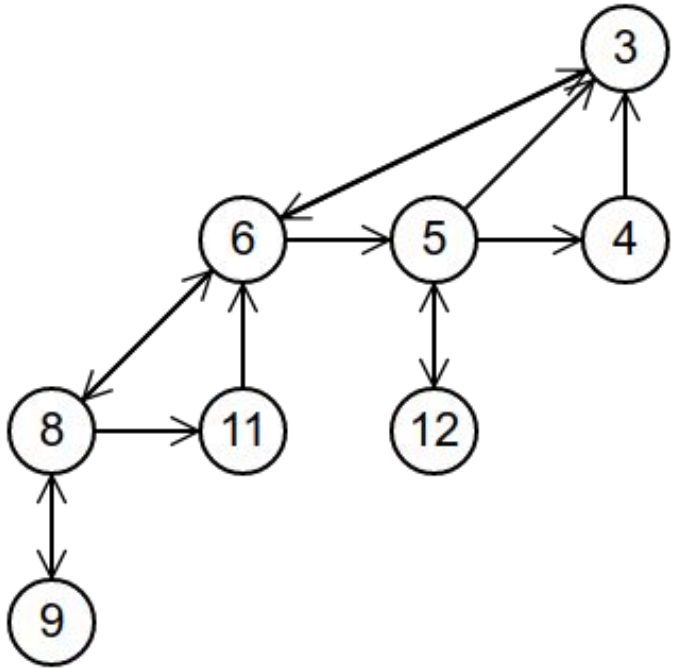
An interesting property of the structure is that it resembles the binary number system.

For example, a binomial heap with 30 elements will have binomial trees of the order 1, 2, 3 and 4, which are in the same positions as the number 30 in binary '11110'.

Links in Binomial Tree

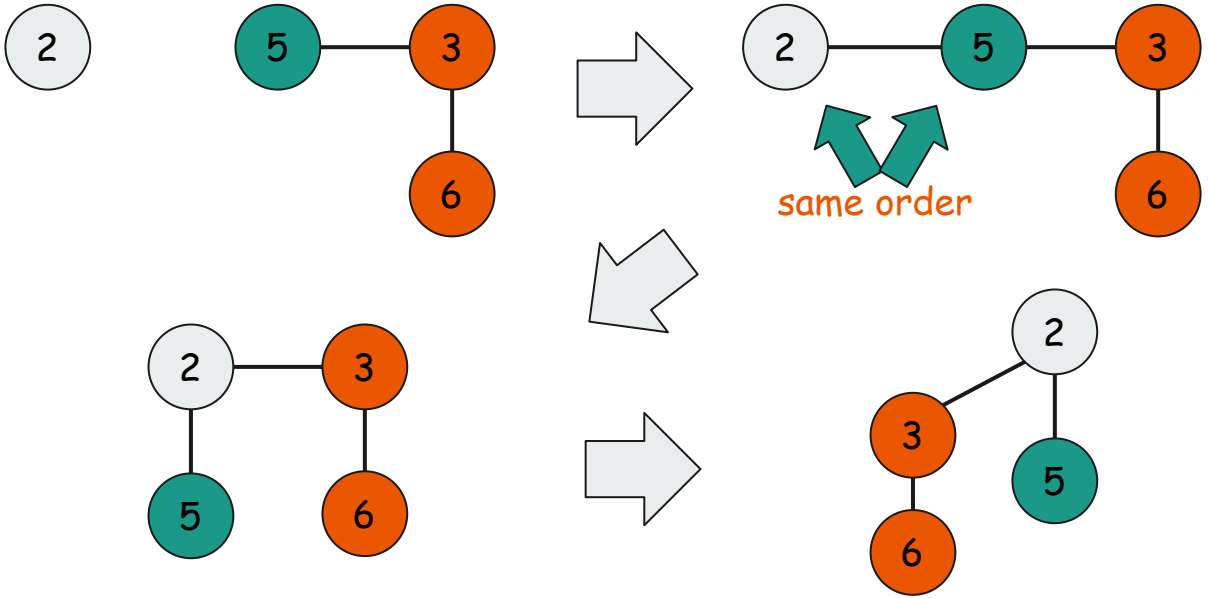
The typical method of implementing the links between nodes is to have pointers to a parent, sibling and child.

A tree does not have a direct link to all it's immediate children, instead it goes to its first child and then iterates through each sibling.



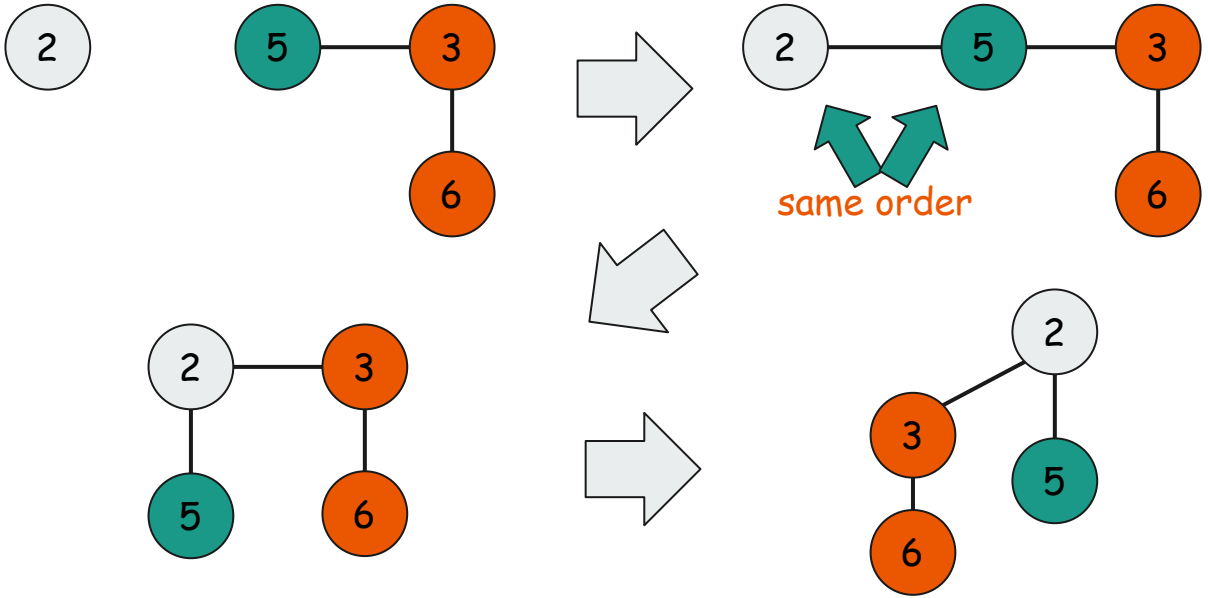
Merge in Binomial Tree

Merges the two heaps together by continually linking trees of the same order until no two trees of the same order exist.



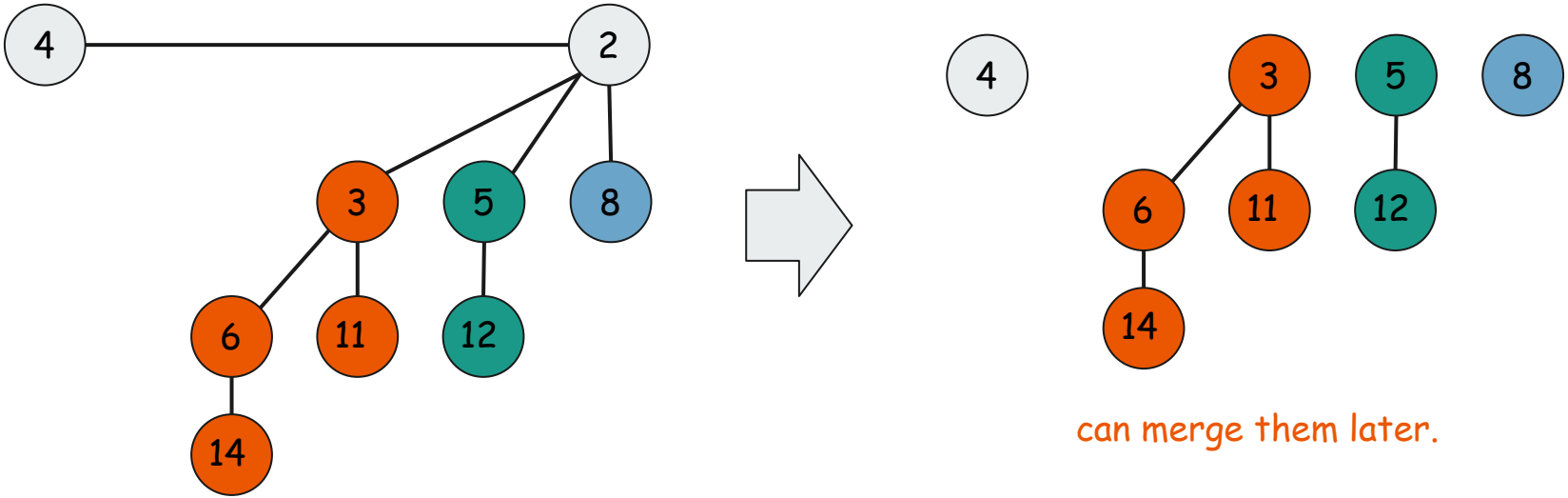
Insert in Binomial Tree

creates a new heap with the inserted element which are then combined using the merge operation.



Delete-min in Binomial Tree

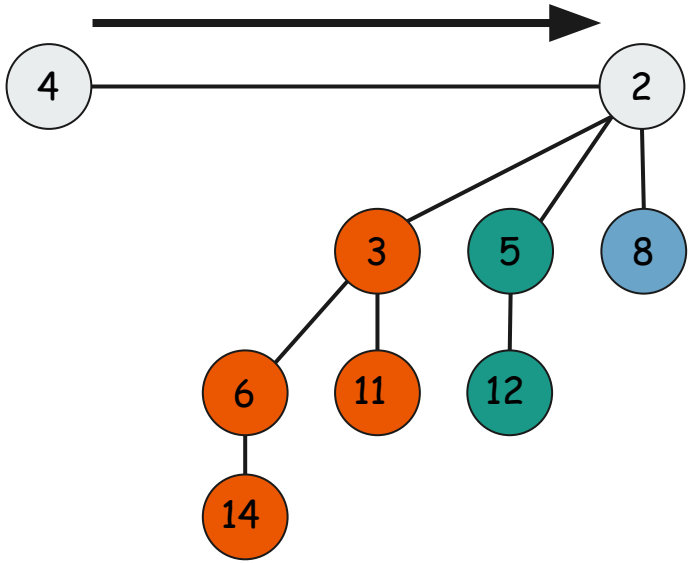
Iterates through the roots of each binomial tree in the heap to find the smallest (largest) node which is removed. The tree fragments are then reversed to form another heap.



can merge them later.

Find minimum in Binomial Tree

iterates through the roots of each binomial tree in the heap.



Performance in Binomial Heap

find-min (max) : $O(\log N)$

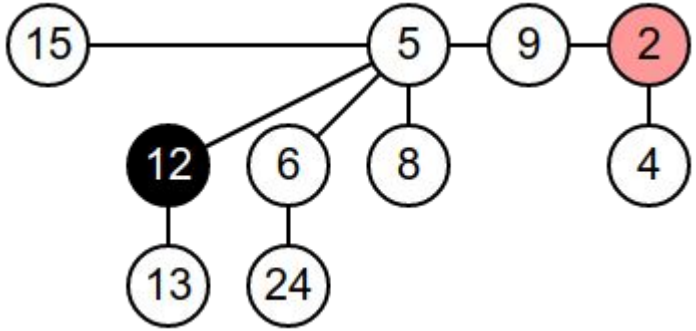
delete-min (max) : $O(\log N)$

insert : $O(1)$

decrease-key : $O(\log N)$

merge : $O(\log N)$

Fibonacci Heap



Fibonacci Heap

A Fibonacci heap is a heap data structure similar to the binomial heap, only with a few modifications and a looser structure.

Binomial heap : **eagerly** consolidate trees after each Insert.

Fibonacci heap : **lazily** defer consolidation until next Delete-min.

Fibonacci Heap

A Fibonacci heap is a collection of heap-ordered trees.

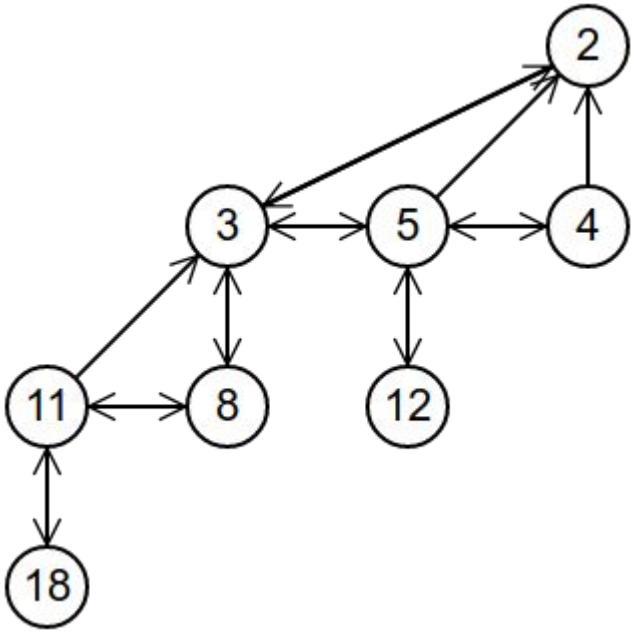
Each tree has an order just like the binomial heap that is based on the number of children. Nodes within a Fibonacci heap can be removed from their tree without restructuring them, so the order does not necessarily indicate the maximum height of the tree or number of nodes it contains.

Links in Fibonacci Heap

Each node stores:

- A pointer to its parent.
- A pointer to any of its children.
- A pointer to its left and right sibilngs.

the child node whose parent links to it is always the node with the smallest value among its siblings.



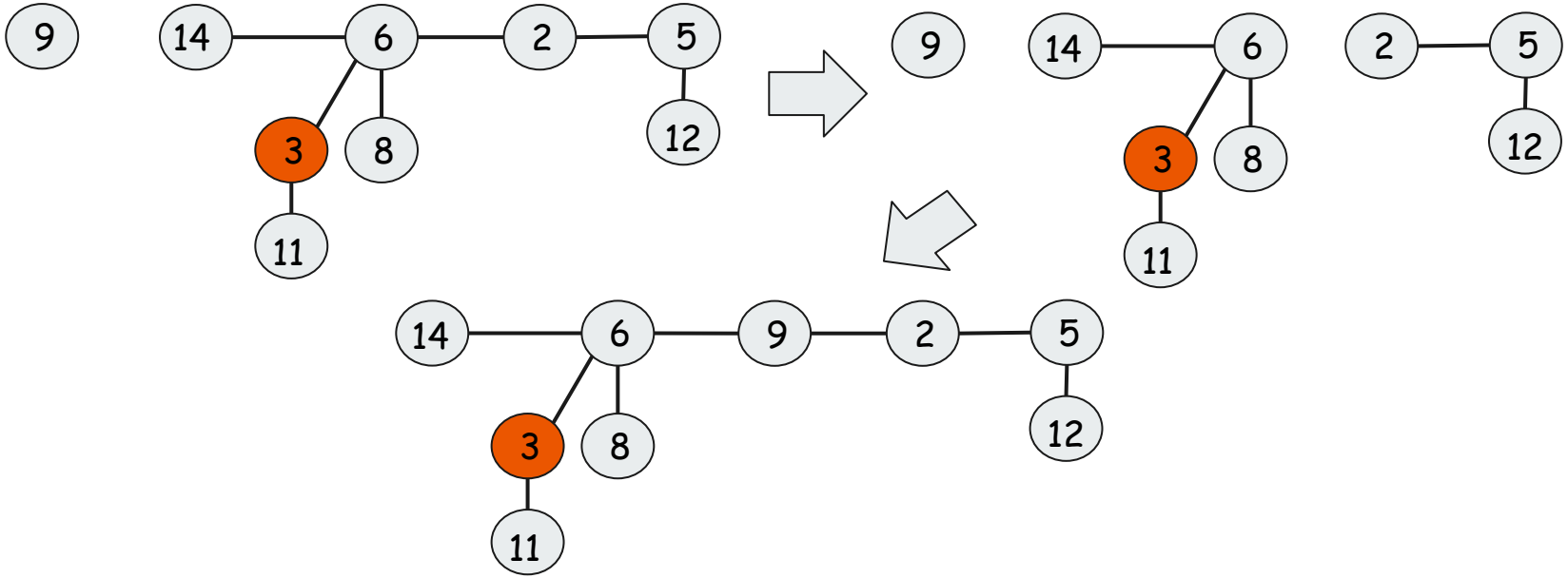
Marked nodes

The decrease key operation marks a node when its child is cut from a tree. Essentially the marking of nodes allows us to track whether:

- The node has had no children cut (unmarked)
- The node has had a single child cut (marked)
- The node is about to have a second child cut (removing a child of a marked node)

Merge in Fibonacci Heap

Concatenates the root lists of two Fibonacci heaps and sets the minimum node to which ever tree's minimum node is smaller.



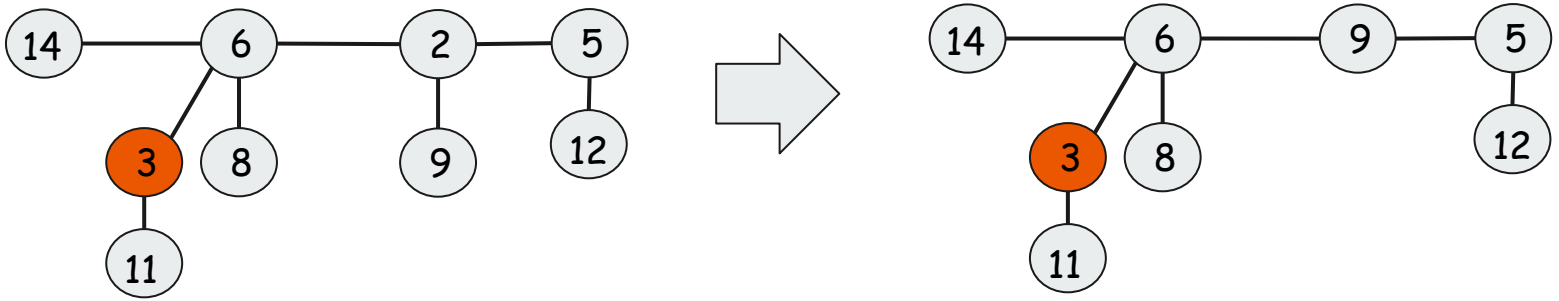
Insert in Fibonacci Heap

creates a new tree containing only the new node which is being added to the heap.

The total number of nodes in the tree is incremented and the pointer to the minimum value is updated if necessary.

Delete-min in Fibonacci Heap

1. Starts by removing the minimum node from the root list and adding its children to the root list.

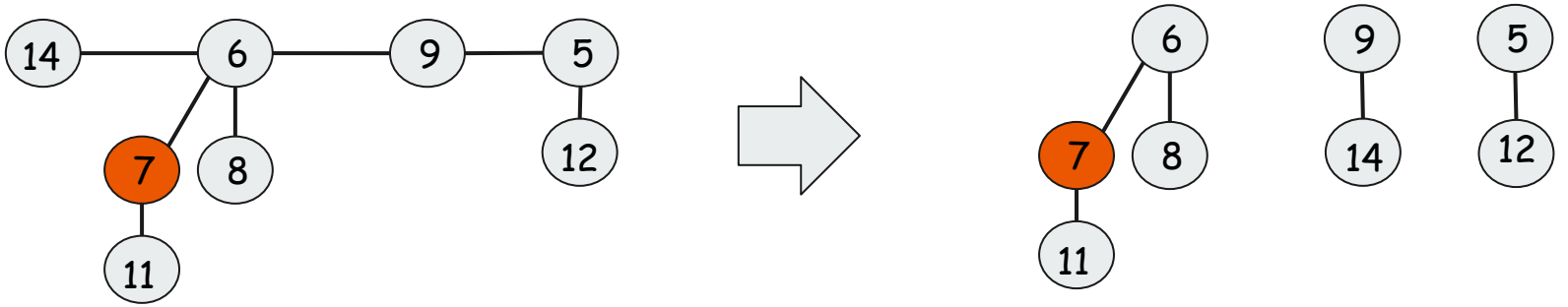


Delete-min in Fibonacci Heap

2. If the minimum was the only node in the root list, the pointer to the minimum node is set to the smallest node in the root list and the operation is completed.

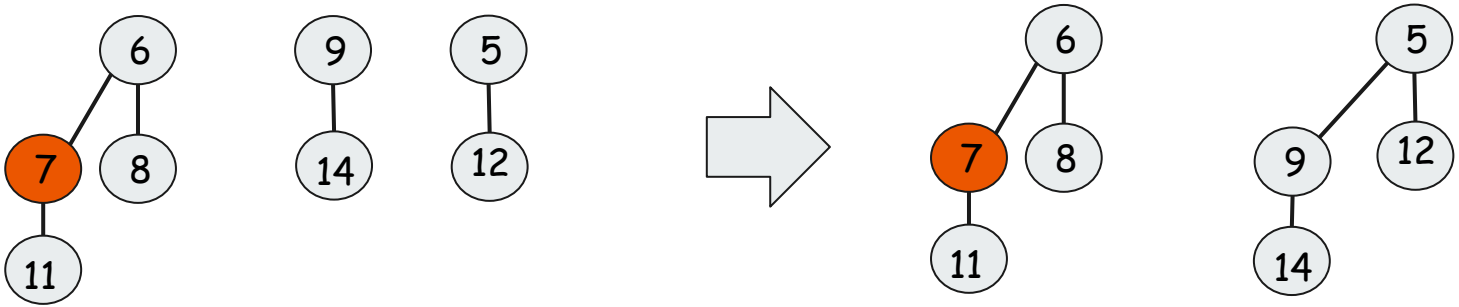
Delete-min in Fibonacci Heap

3. If not, the 'consolidate' operation is performed which merges all trees of the same order together until there are no two trees of the same order. The minimum is then set to the smallest node in the root list.



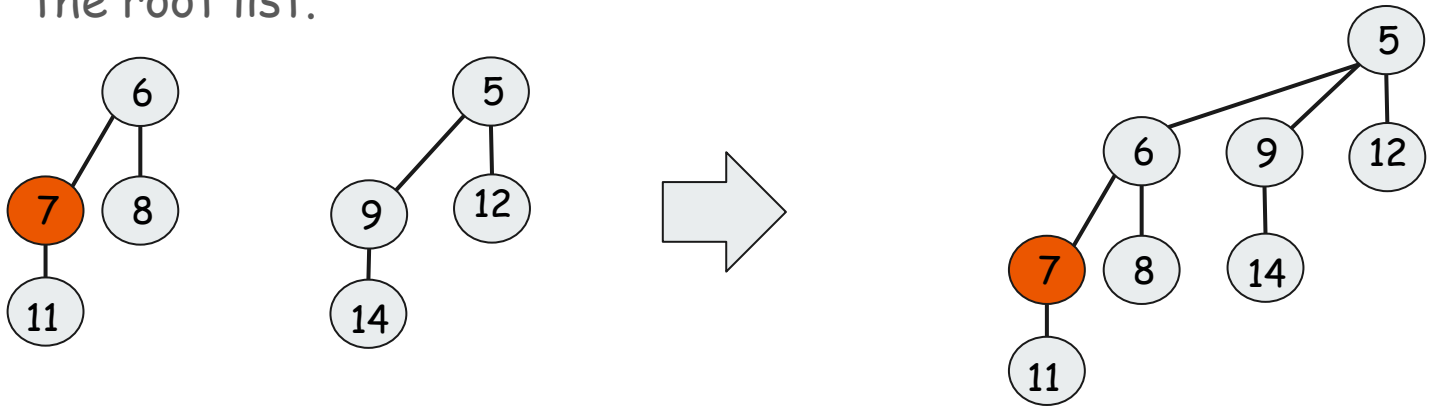
Delete-min in Fibonacci Heap

3. If not, the 'consolidate' operation is performed which merges all trees of the same order together until there are no two trees of the same order. The minimum is then set to the smallest node in the root list.



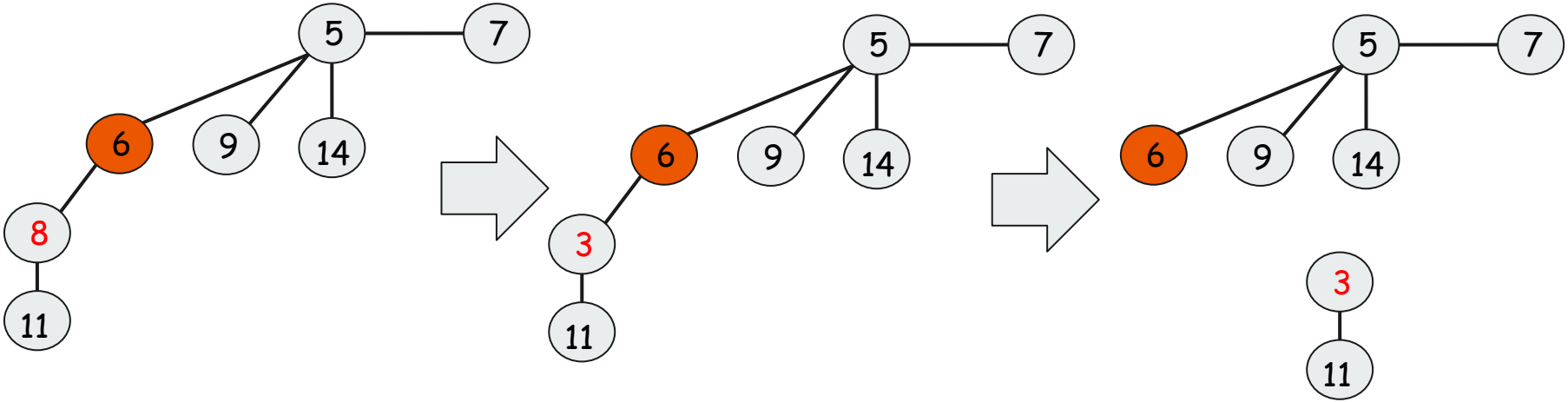
Delete-min in Fibonacci Heap

3. If not, the 'consolidate' operation is performed which merges all trees of the same order together until there are no two trees of the same order. The minimum is then set to the smallest node in the root list.



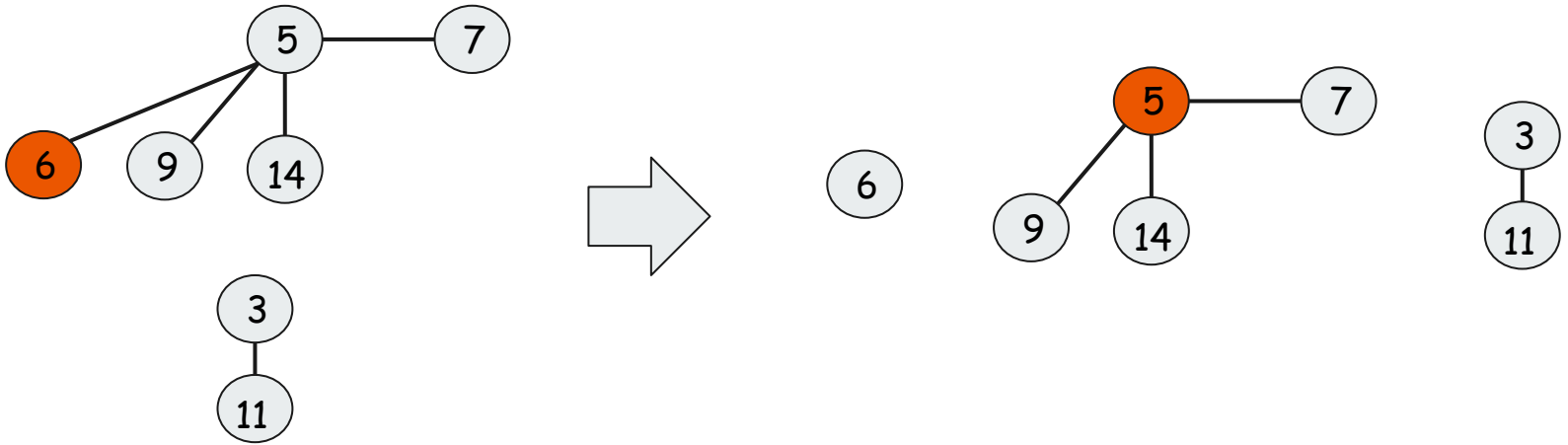
Decrease key in Fibonacci Heap

Decrease key lowers the key of a node. The node is then cut from the tree, joining the root list as its own tree. (example decrease node 8 to 3)



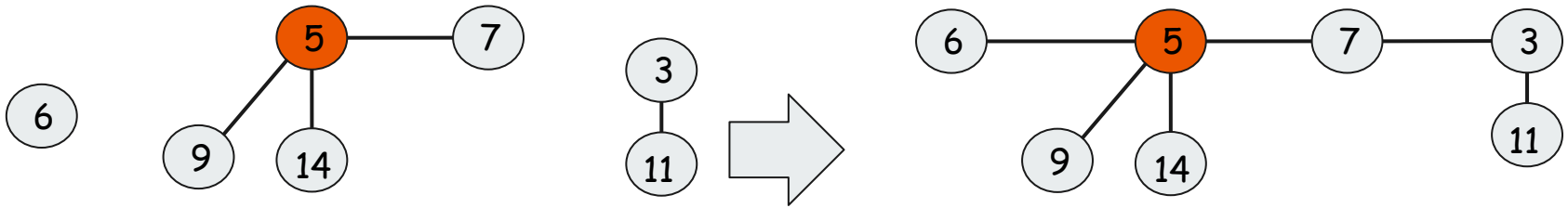
Decrease key in Fibonacci Heap

The parent of the node is then cut if it is marked, this continues for each ancestor until a parent that is not marked is encountered, which is then marked.



Decrease key in Fibonacci Heap

The parent of the node is then cut if it is marked, this continues for each ancestor until a parent that is not marked is encountered, which is then marked.



Performance in Fibonacci Heap

find-min (max) : $O(1)$

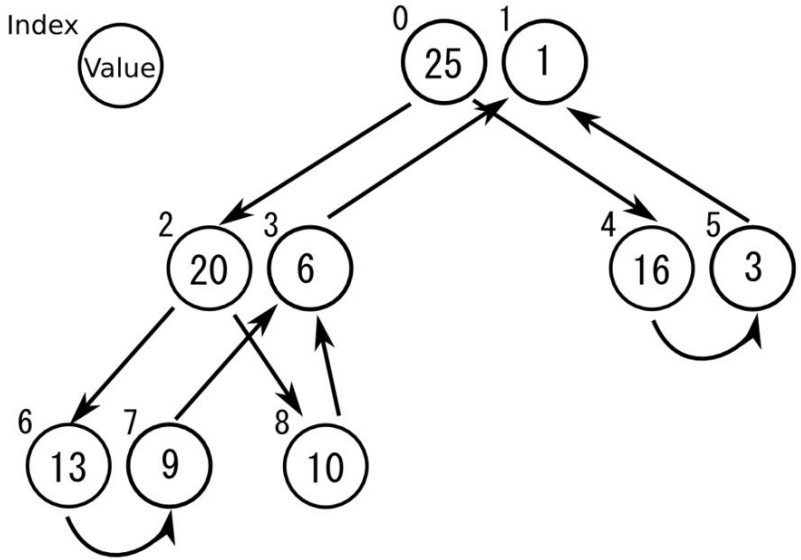
delete-min (max) : $O(\log N)$

insert : $O(1)$

decrease-key : $O(1)$

merge : $O(1)$

Interval Heap



Interval Heap

Use for double-ended priority queue.

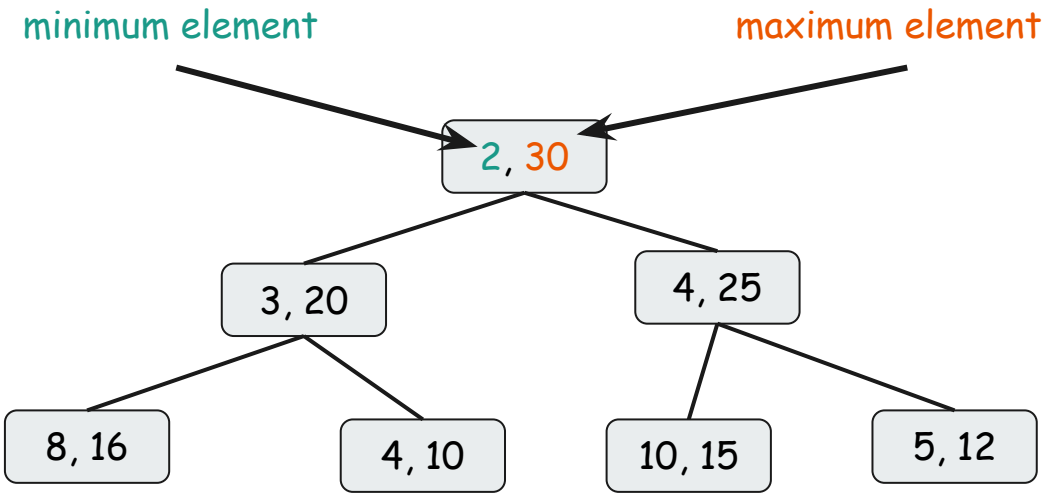
An extension of min heap and max heap that permits us to insert and delete elements in O(log N)

Interval Heap

It is a complete binary tree that satisfies following properties :

- If the total number of node is even, each node has two elements a and b, $a \leq b$. (interval [a, b])
- If the total number of node is odd, the last node has a single element a. (interval [a, a])
- Let $[a_c, b_c]$ be the interval represented by any nonroot node. Let $[a_p, b_p]$ be the interval represented by parent of this node. Then, $[a_c, b_c] \subseteq [a_p, b_p]$. $(a_p \leq a_c \leq b_c \leq b_p)$

Interval Heap



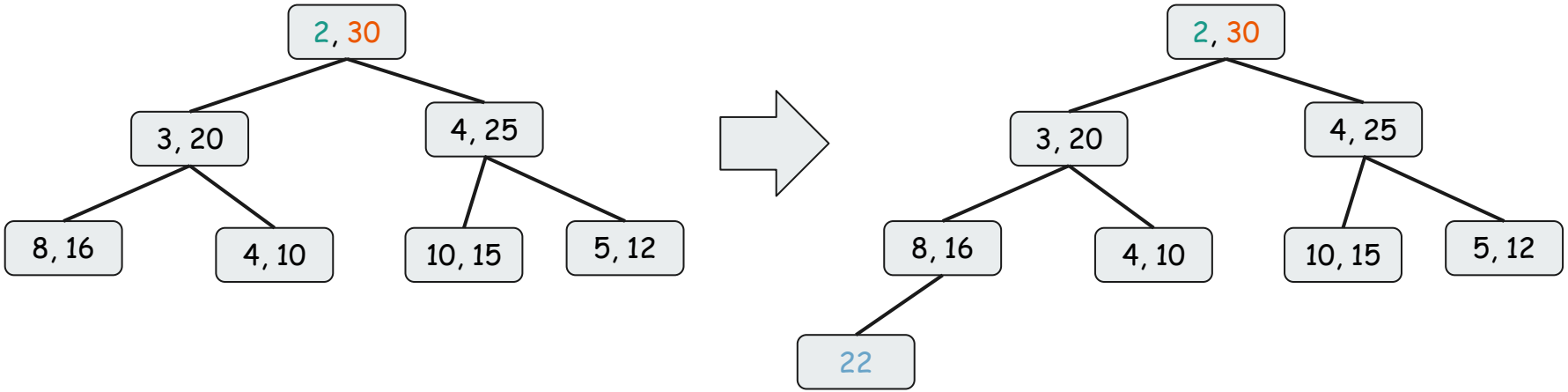
Insertion into Interval Heap

Algorithm :

1. If the last node have one element, accommodate the new element.
2. If the last node have two element, create new node with one new element.
3. Repair heap property by comparing the added element with its parent and swapping positions with the parent.

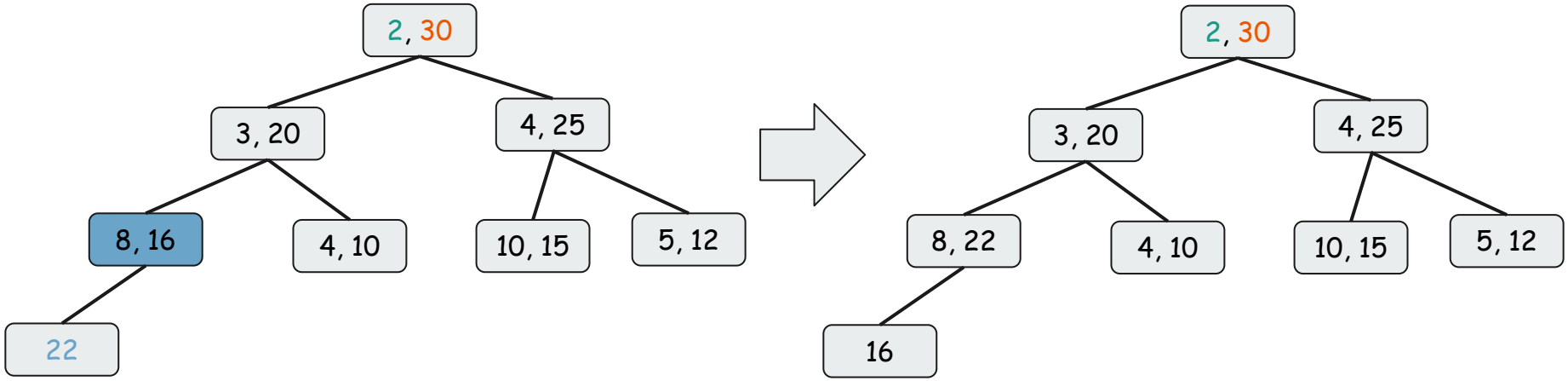
Insertion into Interval Heap

Example : Insert element 22



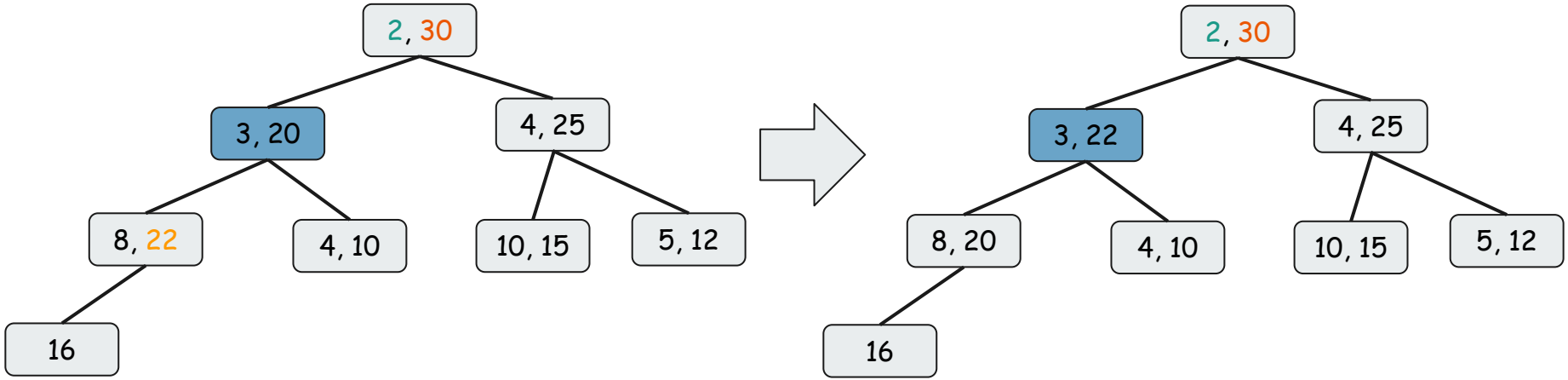
Insertion into Interval Heap

Example : Insert element 22



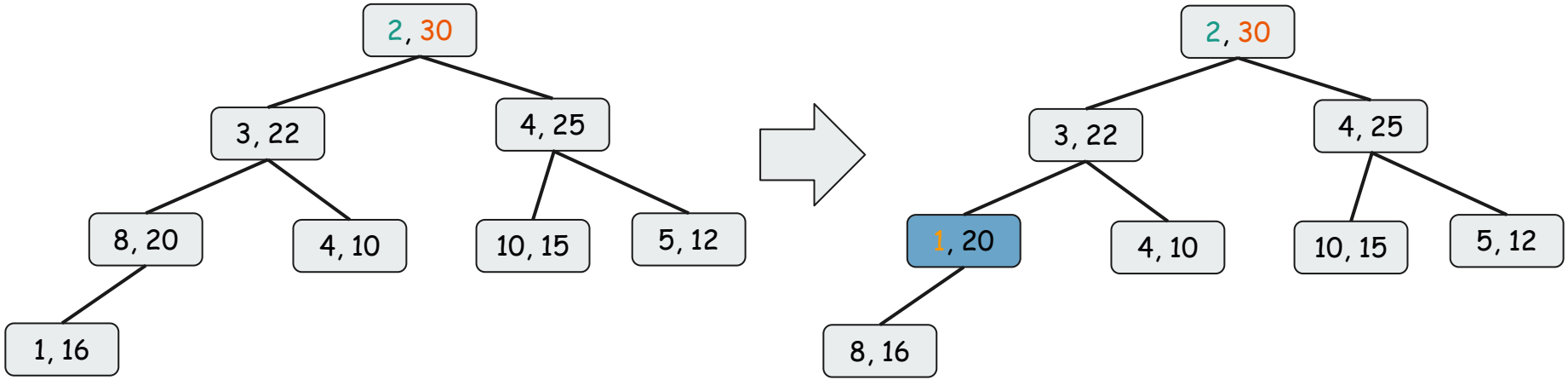
Insertion into Interval Heap

Example : Insert element 22



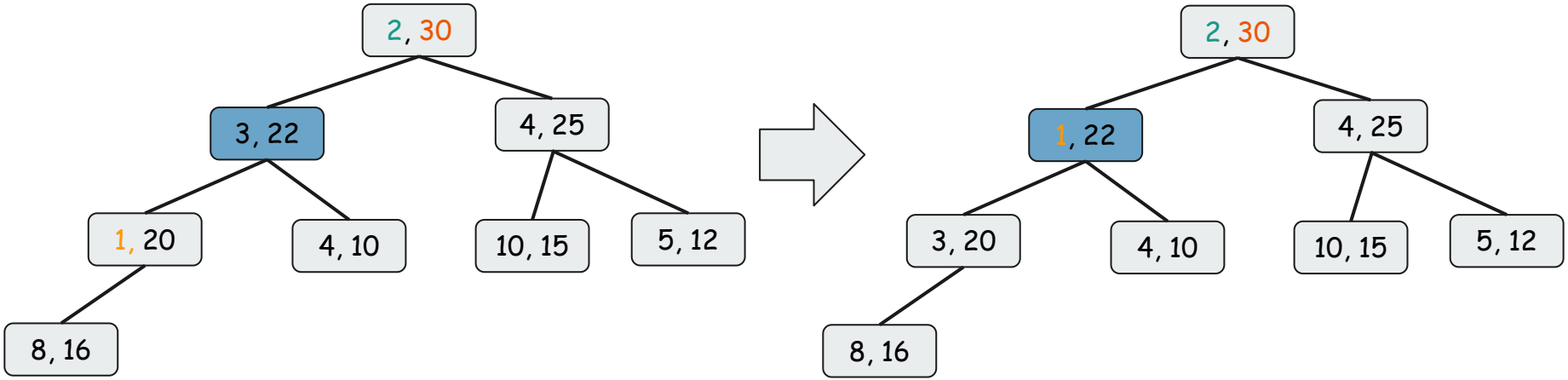
Insertion into Interval Heap

Example : Insert element 1



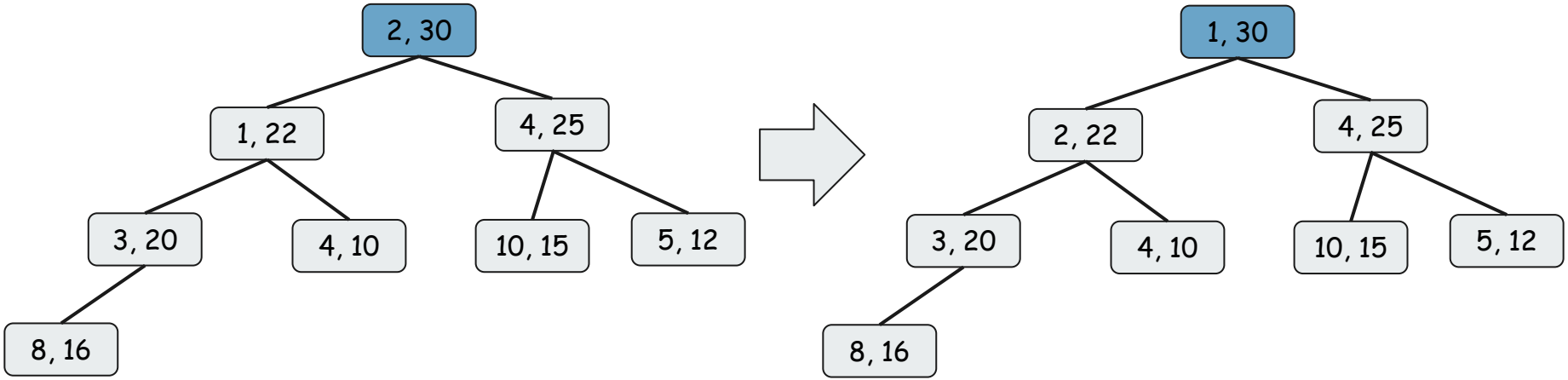
Insertion into Interval Heap

Example : Insert element 1



Insertion into Interval Heap

Example : Insert element 1



Deletion from Interval Heap

Algorithm :

1. Remove the root and replace it with the last element of the heap
2. Restore the heap property by percolating down. Similar to insertion.

Misc.

Binary Heap or Balanced BST?

BSTs have advantages is in-fact bigger compared to binary heap.

- Searching an element.
- We can print all elements of BST in sorted order in O(n) time.
- Floor and ceil can be found in O(Logn) time.
- K'th largest/smallest element be found in O(Logn) time.

Variants of Heap algorithm

Skew heap

Paring heap

skew heap

brodal queue

strict fibonacci heap

etc...