Lecture 20: Array Representation & Heaps

CS 62

Spring 2018

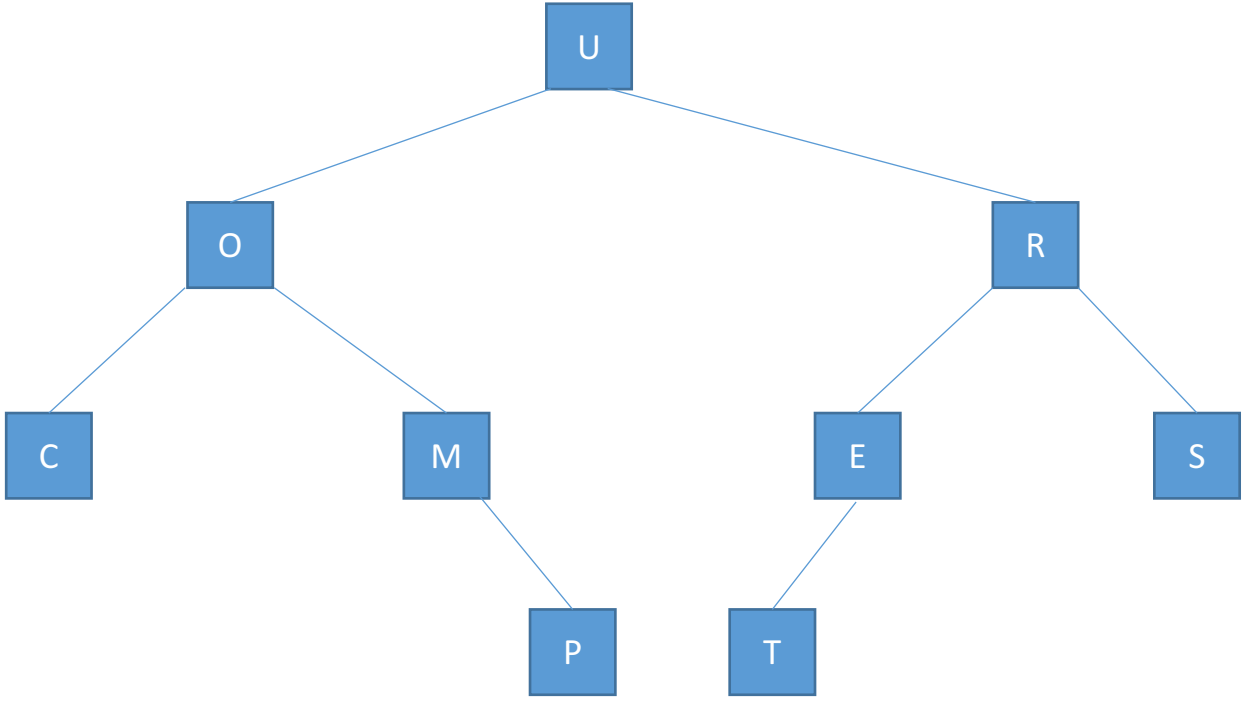
Alexandra Papoutsaki & William Devanny

Array Representation of Binary Trees

Nodes are stored in an array `data`

- Root at `data[0]`
- Left subtree of node `i` in `data[2*i+1]`
- Right subtree of node `i` in `data[2*i+2]`
- Parent of node `i` in `data[(i-1)/2]`

Example



index:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
data[]:	U	O	R	C	M	E	S	--	--	--	P	T	--	--	--

Space usage

Draw different trees and see how much space you'd need for the `data` array

A binary tree T with n nodes requires an array whose length is between n and 2^n-1

Binary Heaps

Complete binary trees with one of the following properties

- Min-heap: the value of each node is greater or equal to its parent, with the minimum element at the root
- Max- heap: the value of each node is smaller or equal to its parent, with the maximum element at the root

When talking about heaps, we will assume min-heaps

Useful when needing to remove object with lowest (or highest priority)

Used to implement priority queues (next lecture)

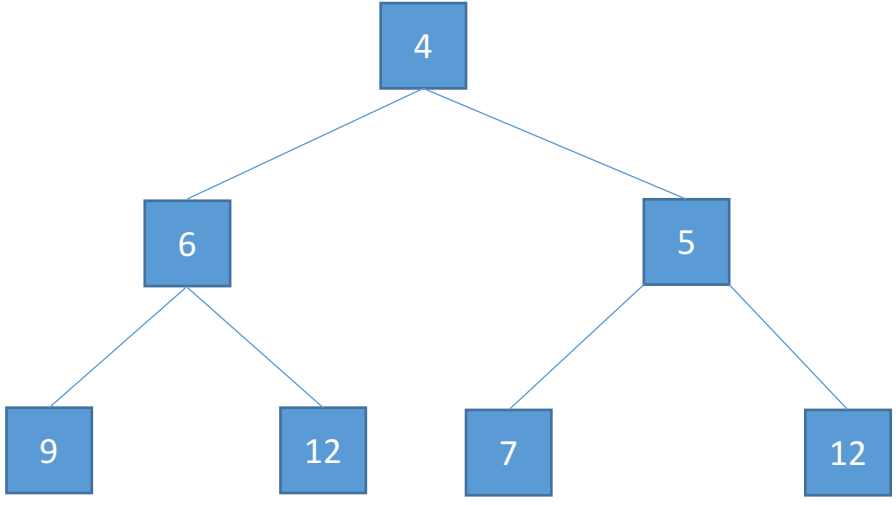
Alternative definition

Min-Heap H is a complete binary tree s.t.

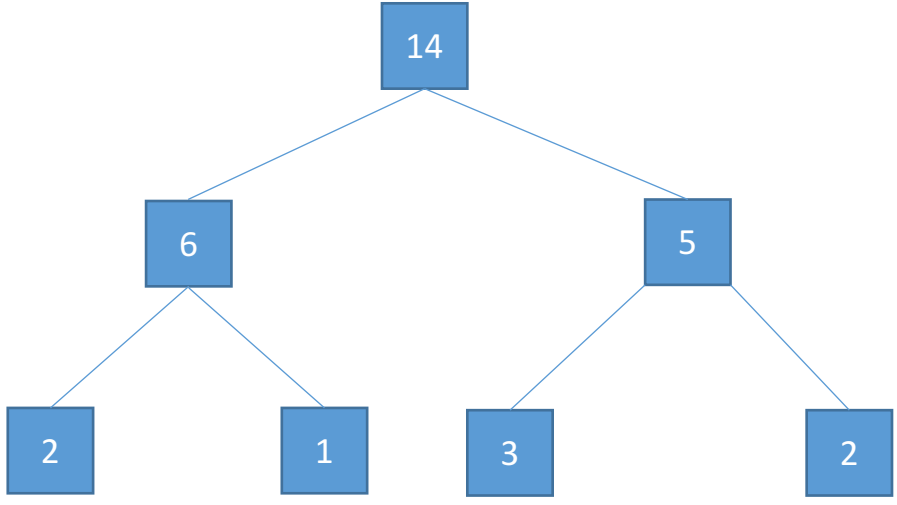
- H is empty, or
- Both of the following hold:
  - The value in root position is smallest value in H
  - The left and right subtrees of H are also heaps. Equivalent to saying parent smaller both than the left and right children

- Since complete tree, smallest possible height $O(\log n)$

Examples



Min-heap



Max-heap

VectorHeap (PriorityQueue in Java)

```
public class VectorHeap<E> {

        protected Vector<E> data;
        public VectorHeap() {
                data = new Vector<E>();

        }
        public VectorHeap(Vector<E> v) {

                int i;

                data = new Vector<E>(v.size());
                for (i = 0; i < v.size(); i++) {
                        add(v.get(i));
                }
        }

        protected static int parent(int i) {
                return (i-1)/2;
        }
        protected static int left(int i) {
                return 2*i+1;
        }
        protected static int right(int i) {
                return 2*(i+1);
        }
}
```

Insertion

Place node in next available position

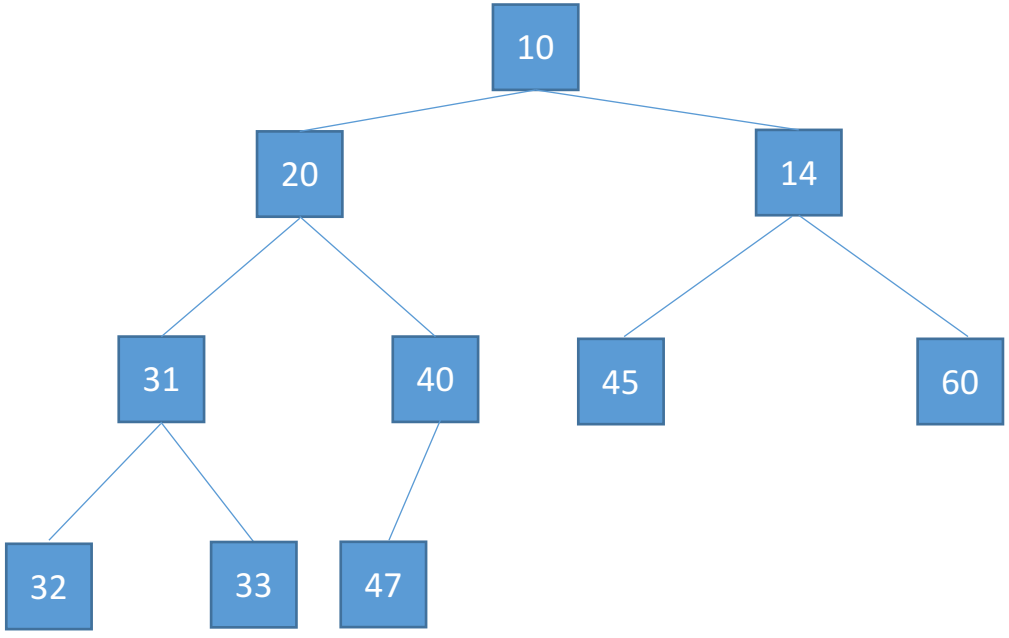
“Percolate” it up

Worst-case: $O(\log n)$

Percolate up

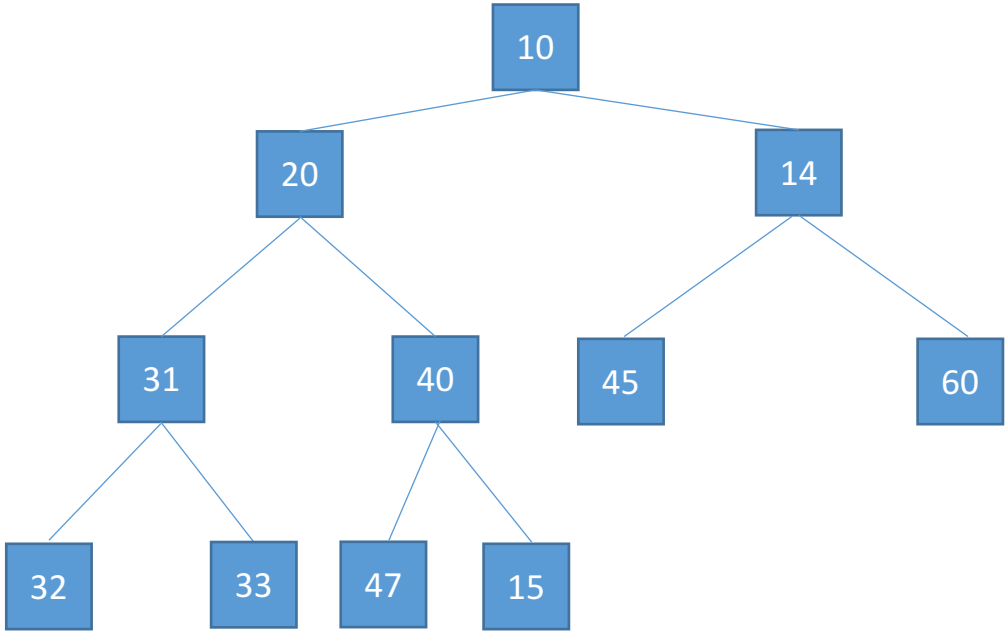
```
/**
* Moves node upward to appropriate position within heap.
* @param leaf Index of the node in the heap.
* @pre 0 <= leaf < size
* @post moves node at index leaf up to appropriate
* position
*/
protected void percolateUp(int leaf) {
	int parent = parent(leaf);
	E value = data.get(leaf);
	while (leaf > 0 && (value.compareTo(data.get(parent)) < 0)) {
		data.set(leaf,data.get(parent));
		leaf = parent;
		parent = parent(leaf);
	}
	data.set(leaf,value);
}
```

Insert 15



Index: 0 1 2 3 4 5 6 7 8 9 10
data: 10 20 14 31 40 45 60 32 33 47 -

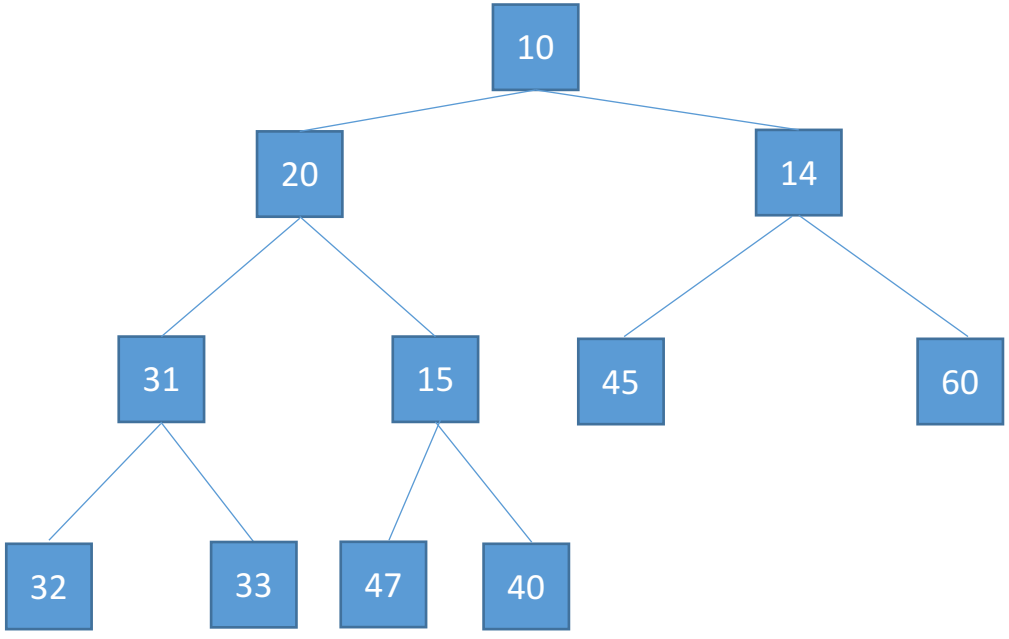
Insert 15



Index: 0 1 2 3 4 5 6 7 8 9 10
data: 10 20 14 31 40 45 60 32 33 47 -

Index: 0 1 2 3 4 5 6 7 8 9 10
data: 10 20 14 31 40 45 60 32 33 47 15

Insert 15

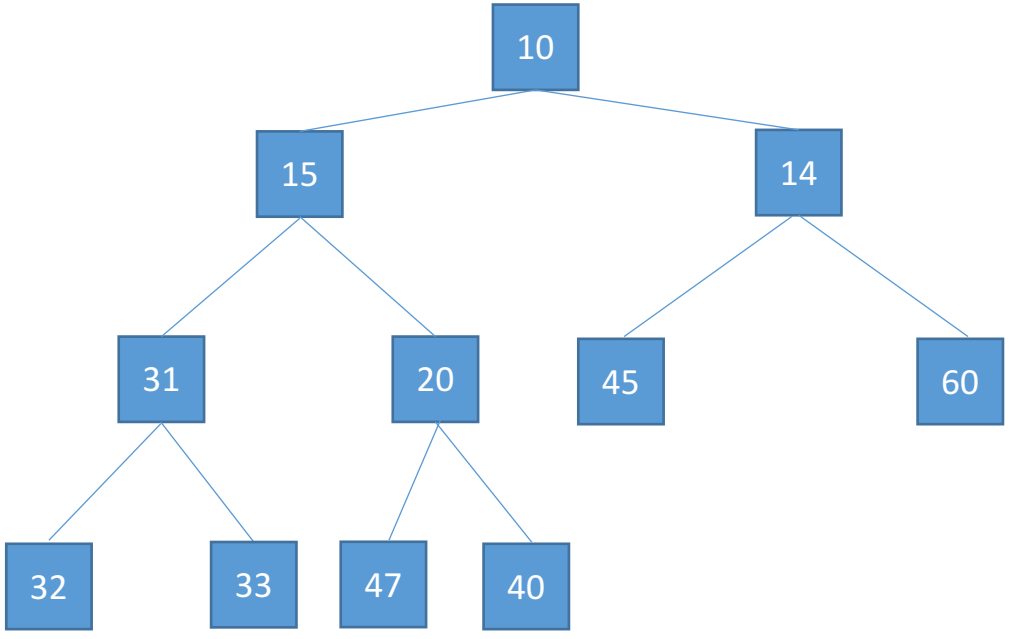


Index:	0	1	2	3	4	5	6	7	8	9	10
data:	10	20	14	31	40	45	60	32	33	47	-

Index:	0	1	2	3	4	5	6	7	8	9	10
data:	10	20	14	31	40	45	60	32	33	47	15

Index:	0	1	2	3	4	5	6	7	8	9	10
data:	10	20	14	31	15	45	60	32	33	47	40

Insert 15



Index: 0 1 2 3 4 5 6 7 8 9 10
data: 10 20 14 31 40 45 60 32 33 47 -

Index: 0 1 2 3 4 5 6 7 8 9 10
data: 10 20 14 31 40 45 60 32 33 47 15

Index: 0 1 2 3 4 5 6 7 8 9 10
data: 10 20 14 31 15 45 60 32 33 47 40

Index: 0 1 2 3 4 5 6 7 8 9 10
data: 10 15 14 31 20 45 60 32 33 47 40

Removal of minimum value

More complicated

Remove top, smallest element

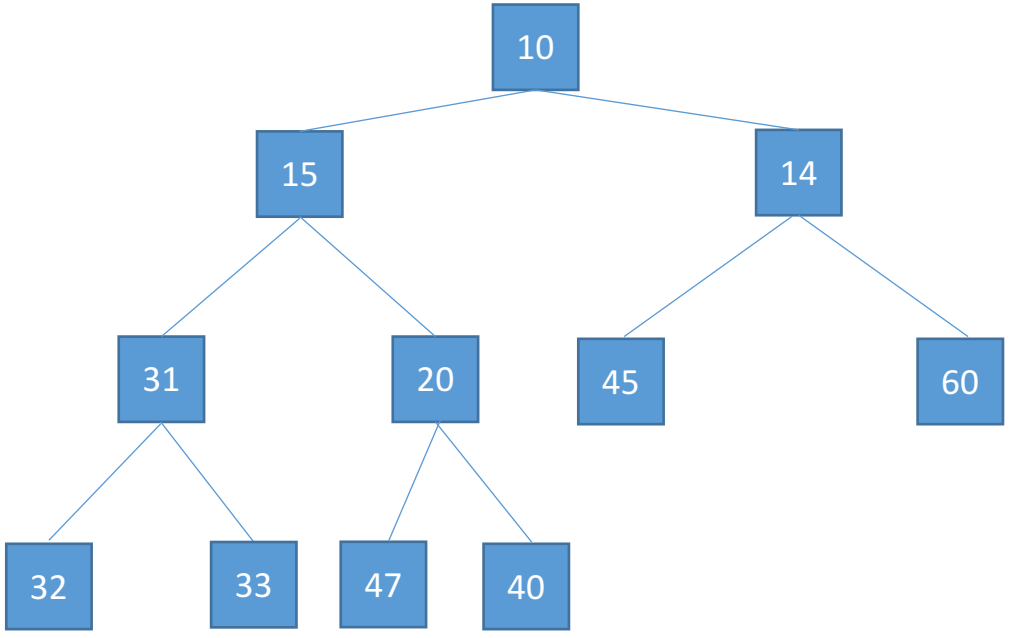
Move last element in array to top so that last node is freed

Since this is a large element need to push down while larger than either child

- Swap with smallest child if largest than it

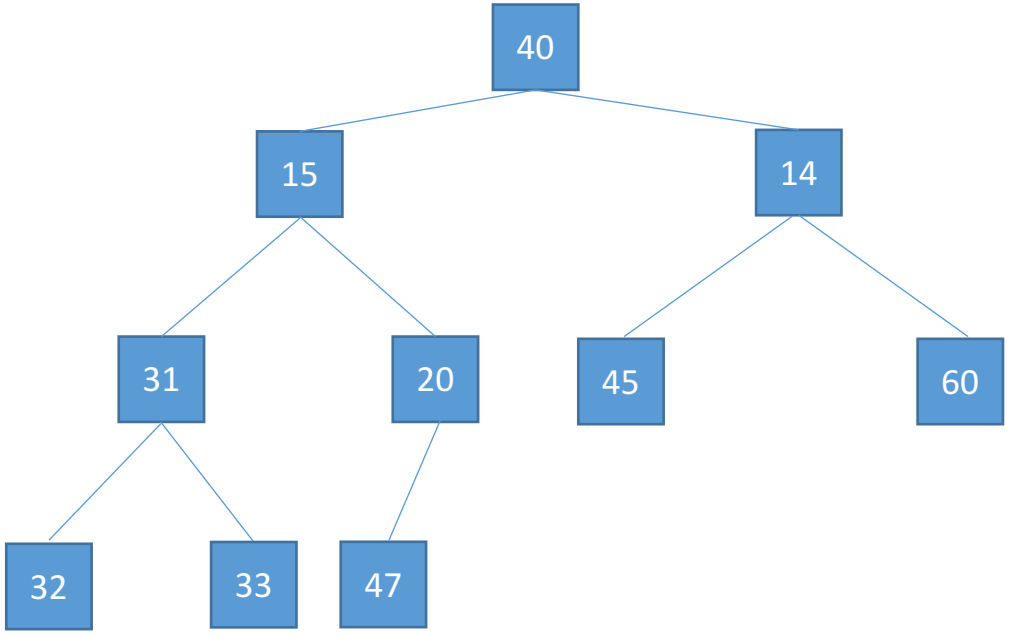
Worst-case: $O(\log n)$

Remove root



Index: 0 1 2 3 4 5 6 7 8 9 10
data: 10 15 14 31 20 45 60 32 33 47 40

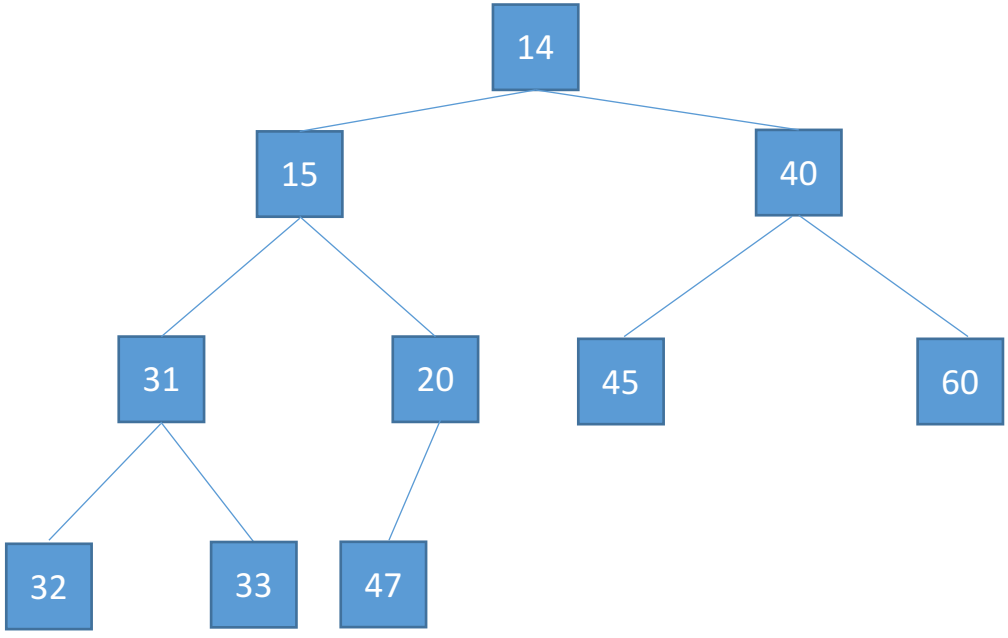
Remove root



Index:	0	1	2	3	4	5	6	7	8	9	10
data:	10	15	14	31	20	45	60	32	33	47	40

Index:	0	1	2	3	4	5	6	7	8	9	10
data:	40	15	14	31	20	45	60	32	33	47	--

Remove root



Index:	0	1	2	3	4	5	6	7	8	9	10
data:	10	15	14	31	20	45	60	32	33	47	40

Index:	0	1	2	3	4	5	6	7	8	9	10
data:	40	15	14	31	20	45	60	32	33	47	--

Index:	0	1	2	3	4	5	6	7	8	9	10
data:	14	15	40	31	20	45	60	32	33	47	--

Removal of any node

Similarly to removing root node

Exchange with rightmost node of the last level

If this exchanged node is smaller than its parent percolate up
If it is larger push down