Heap

A Heap is a Binary Tree inside an array, so it does not use parent/child pointers. A Heap is sorted based on the "Heap Property" that determines the order of the nodes in the tree.

[18, 12, 11, 10, 7]

We will see later how this array represents a Max-Heap

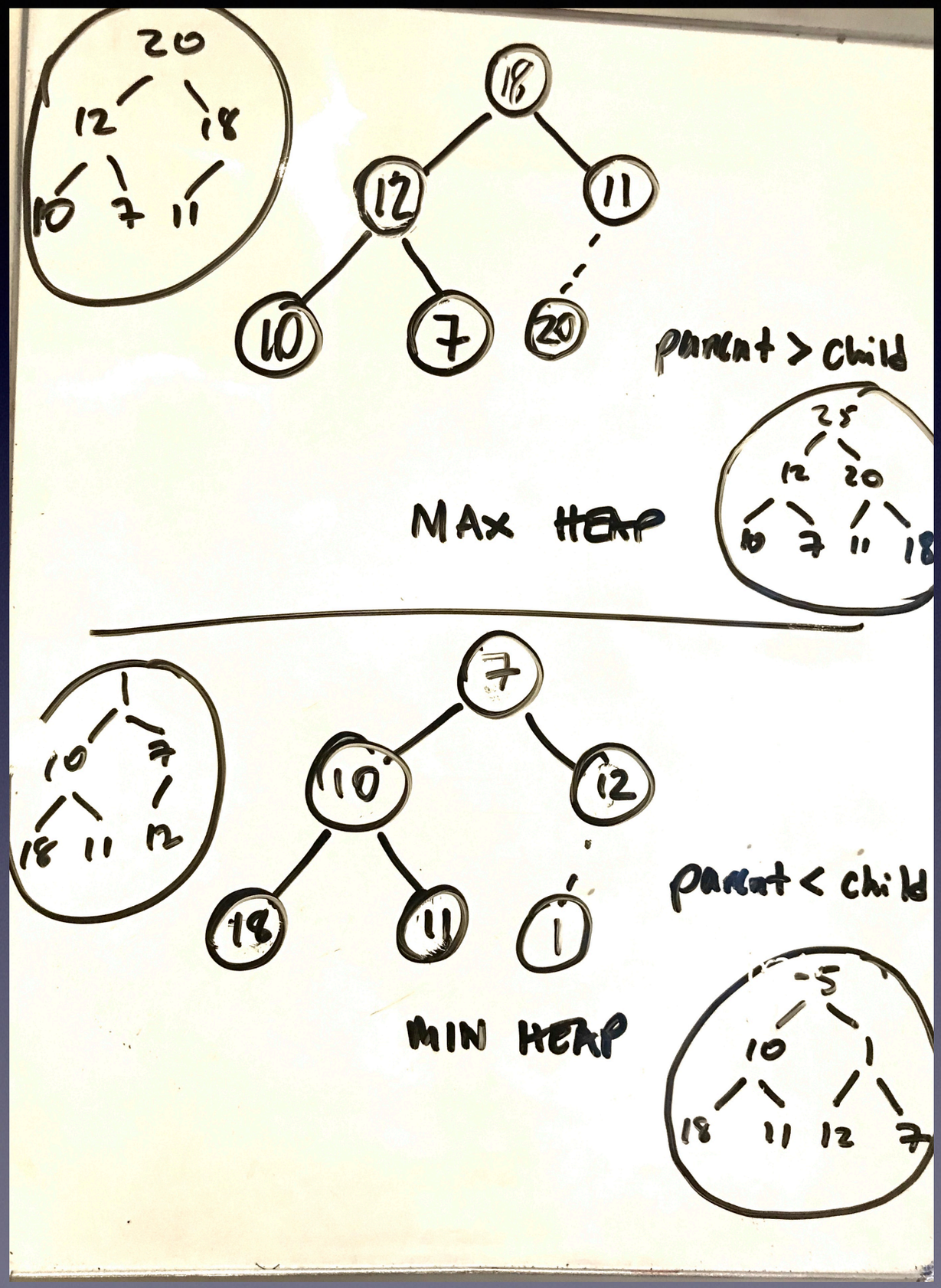
Heap of Stones

Objectives

- Know what a Heap Data Structure is mostly used for
- Be able to diagram a Heap and explain its structure
- Know differences between a min-heap and max-heap
- Be familiar with sifting and keeping the Heap Property validated
- Implement a Heap

Sifting

Max and Min Heap



Vocabulary

- Heap Property (invariant)
- Max-Heap
- Min-Heap
- Shift up (sifting)
- Shift down (sifting)
- Heapify - convert an array to a Heap

Getting Indices

To get the appropriate index of a node use the formulas below:

- Parent Index: (i- 1) / 2
- Left Child Index: 2 * i + 1
- Right Child Index: 2 * I + 2
- e.g to get parent of index 3, (3 - 1)/2 = 1

Common Operations

- insert()
- remove()
- remove(index:)
- replace(index: value:)

NB: All above take O(log n) time

Some Applications

- Priority Queue - we will see this data structure later this week
- Order Statistics - efficiently find the kth smallest or largest element in an array
- Graph Algorithms - Dijkstra's shortest-path algorithm

Resources

- Heap Data Structure: https://github.com/raywenderlich/swift-algorithm-club/tree/master/Heap

- Heap - Wikipedia: https://en.wikipedia.org/wiki/Heap_(data_structure)#Applications

Let us walk through a visual example of insert(_ value: T)

Here is our Heap seen as an Array, so far the Heap Property is satisfied. Let us insert 20.

[18, 12, 11, 10, 7]

1. Append the new value to the Array
2. Shift the node up until the Heap Property is met or you've reached the root

Code

Heap (Continued)

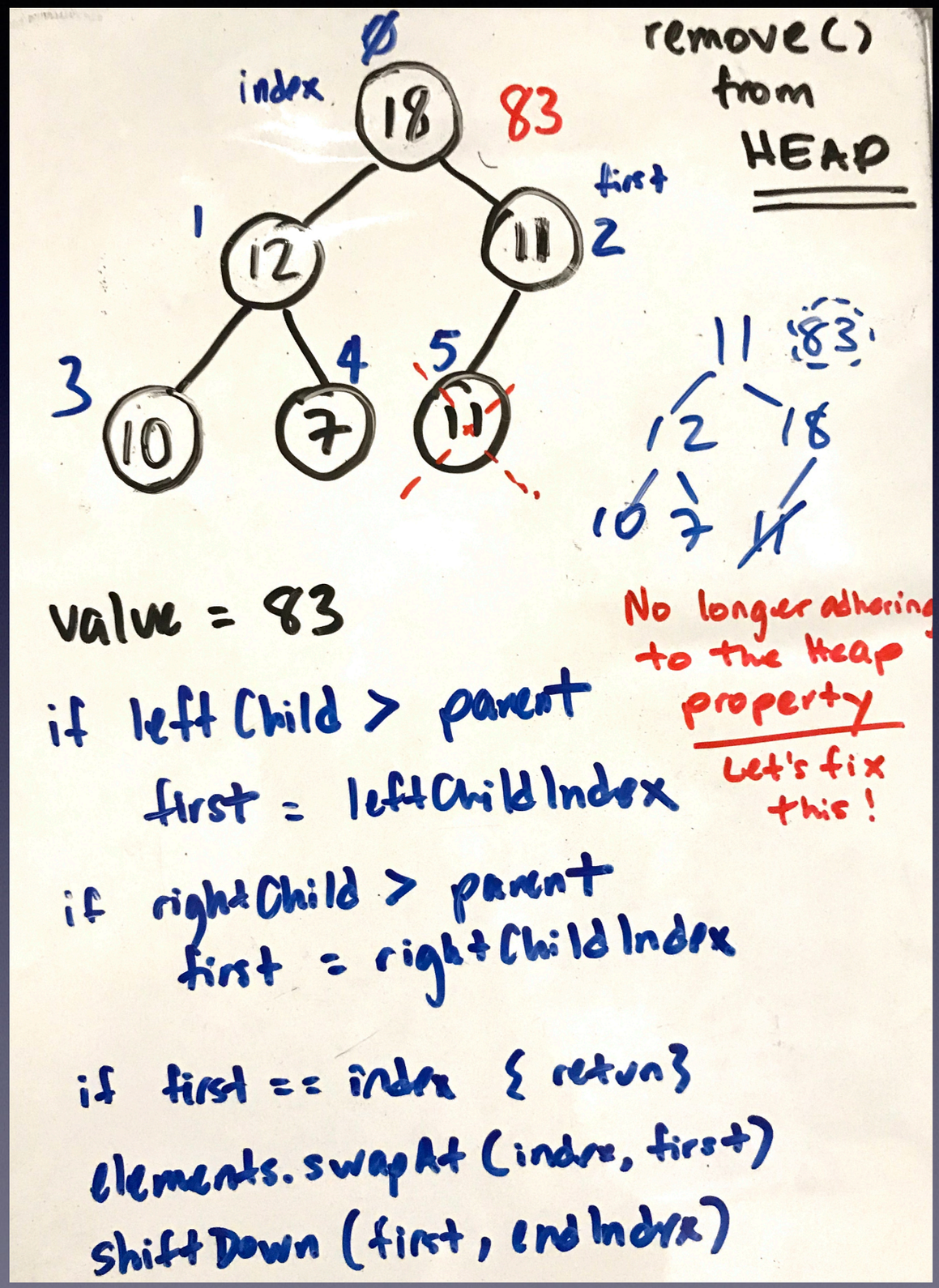
Review Questions

- What is a Heap?
- Difference between a Min-Heap and a Max-Heap?
- How do we keep the Heap Property validated when inserting a new child in the tree

In the last Code Demo we inserted a child node. Now let's review removing.

1. Store root value
2. Overwrite root value with last element
3. Remove last element
4. Fix the Heap Property by shifting the root node down and comparing parent with child
5. Recursively do above until you've reached the end of the array or Heap Property is met.

Visual representation of removing root node
We start off with [83, 12, 18, 10, 7, 11]

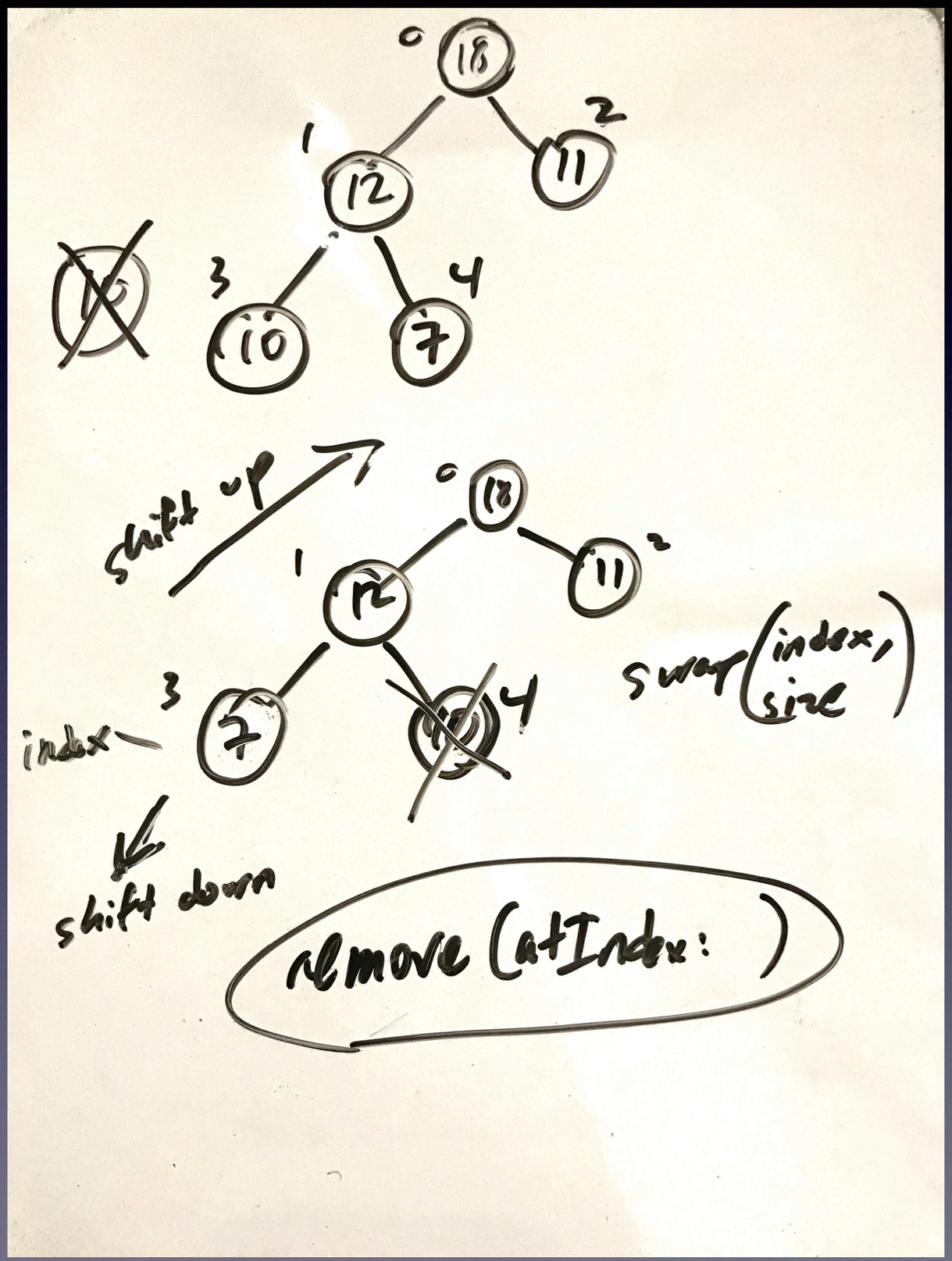


Code

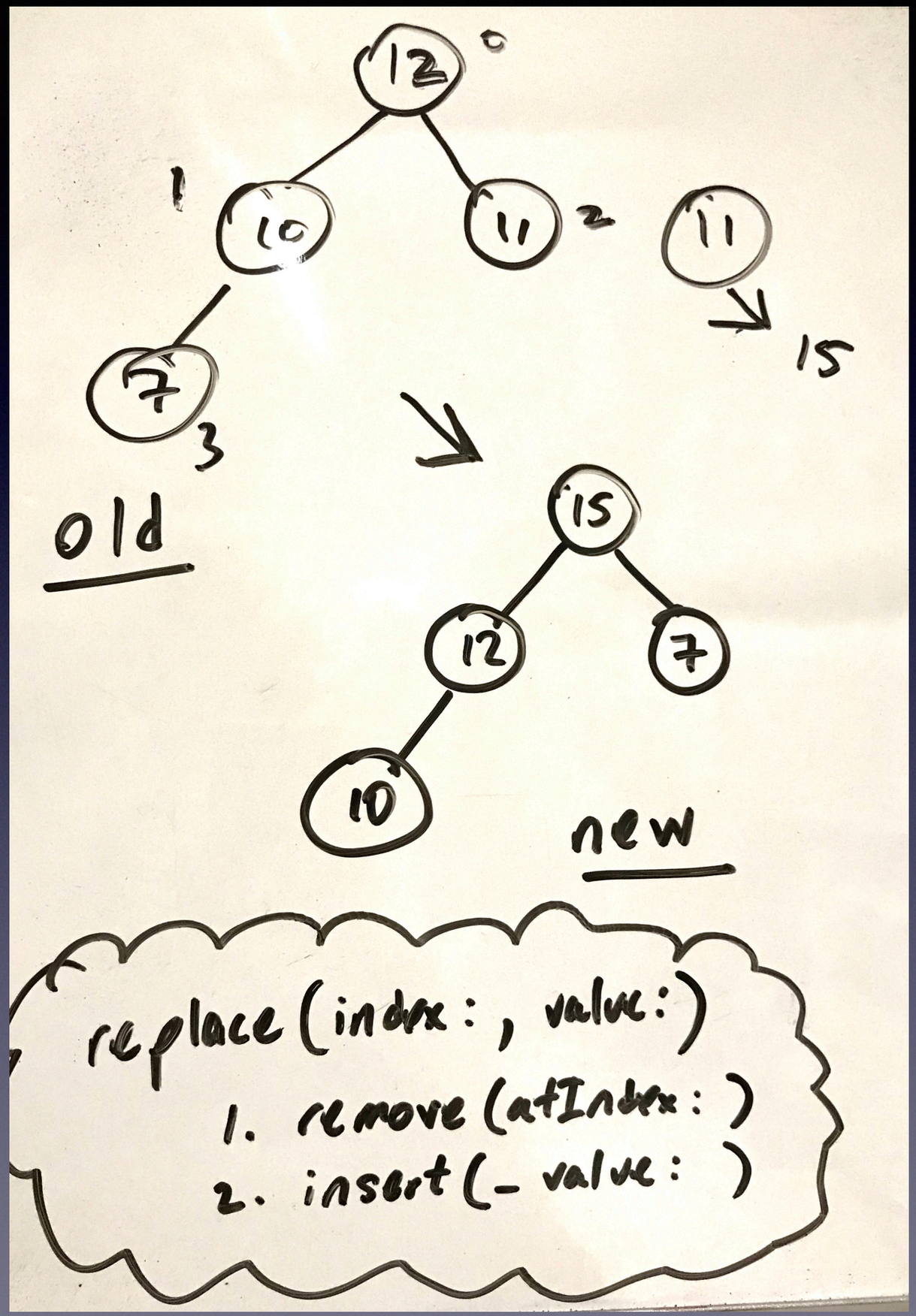
Additional Operations

- remove(atIndex:) : remove an element at a provided index value
- replace(atIndex:) : first removes the element at the given index then adds a new value to the Heap
- heapify(_ arr:) : builds a Heap from a given array

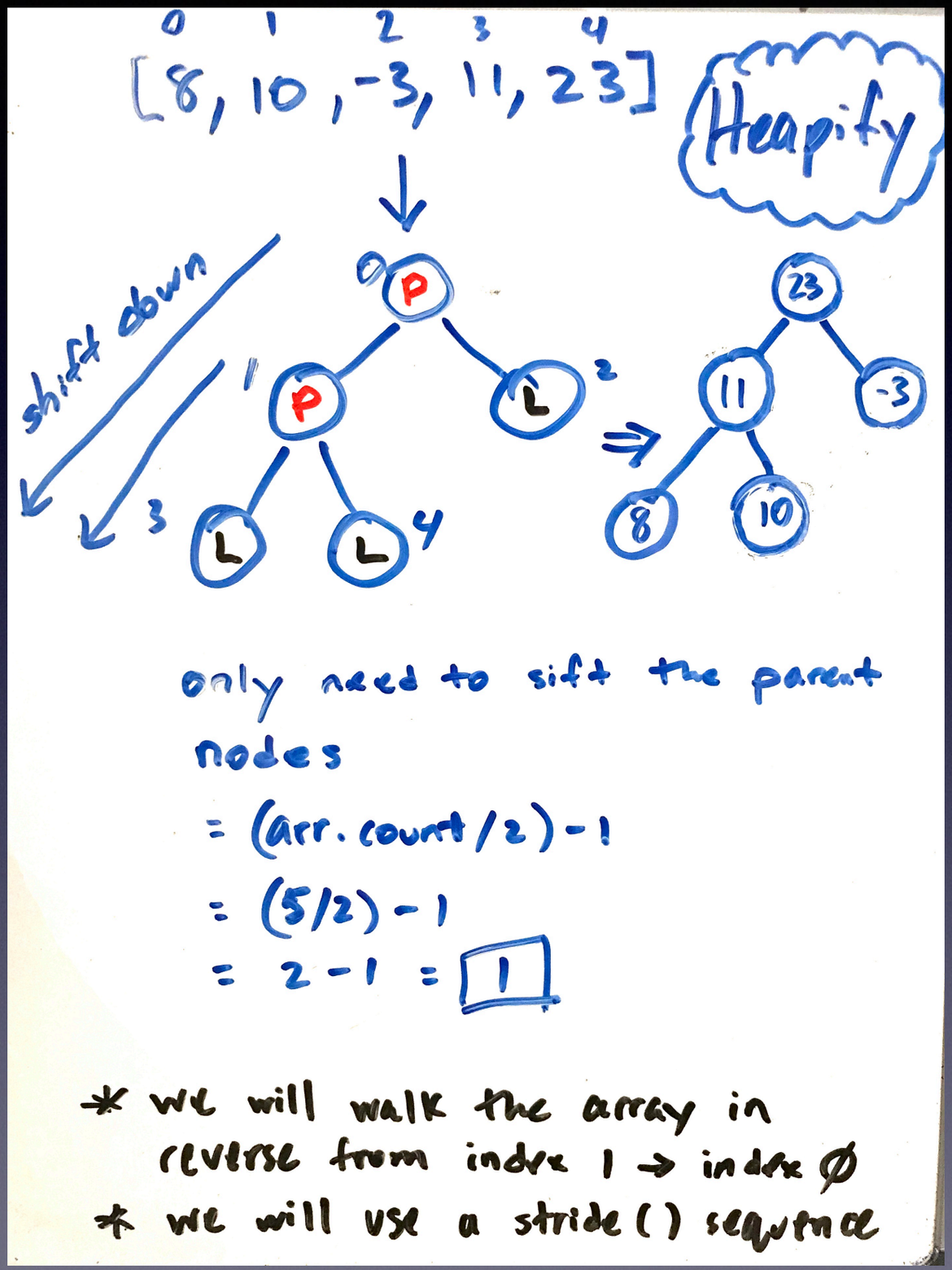
Visual representation for remove(atIndex:) operation



Visual representation for replace(atIndex: value:) operation



Visual representation for heapify(_ arr:) operation



No need to sift down leaf nodes here, only parent nodes