Binary Heap with TDD

2020-03-17 RubySG Online Meetup

Who am I?

I am a Software Engineer working at Shopify (Singapore).

I hardly use Social Media, but you can find me on Twitter **@taykangsheng**

My hobby is long distance running! Find me on Strava **Kang Sheng Tay**!

Objective of today’s sharing

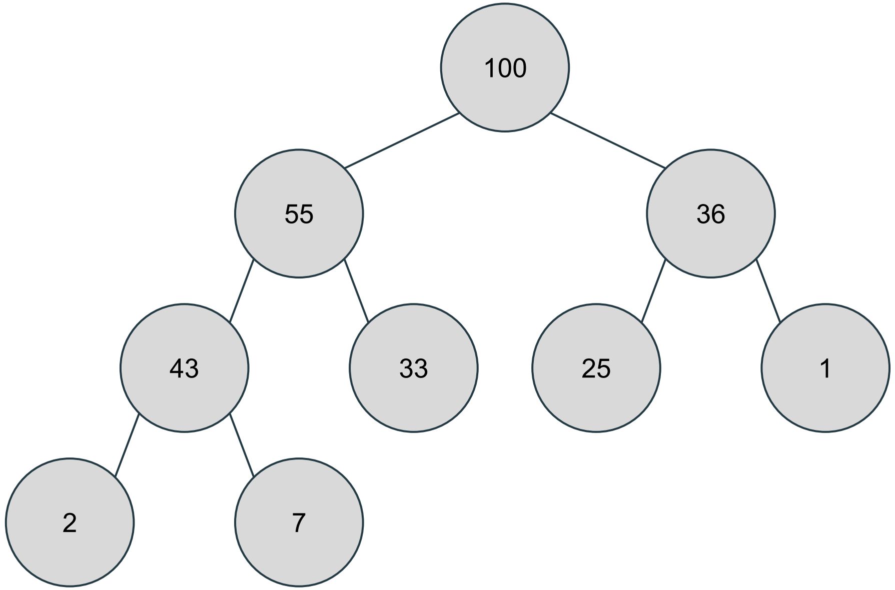
1. 80% Learn / revisit a data structure (Binary Heap) today!
2. 20% Do some Test Driven Development together!

What is Binary Heap used for?

- Commonly used to build Priority Queues
- Introduced by J. W. J. Williams in 1964 as a data structure for heap sort.
- O(log n) time complexity for Insert and Extracting elements from the heap.

What is a Binary Heap?

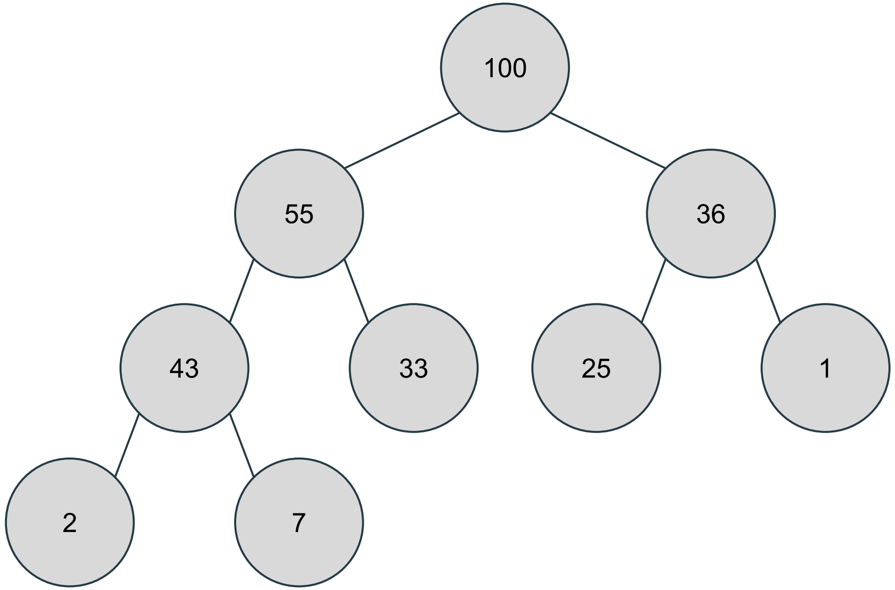
1. Binary Heap is a structure that stores data in the form of a binary tree (and of course with some rules / constraints).



Rules (Constraints) of Binary Heap

Binary Heap have 2 rules (constraints) that makes it more than just a binary tree.

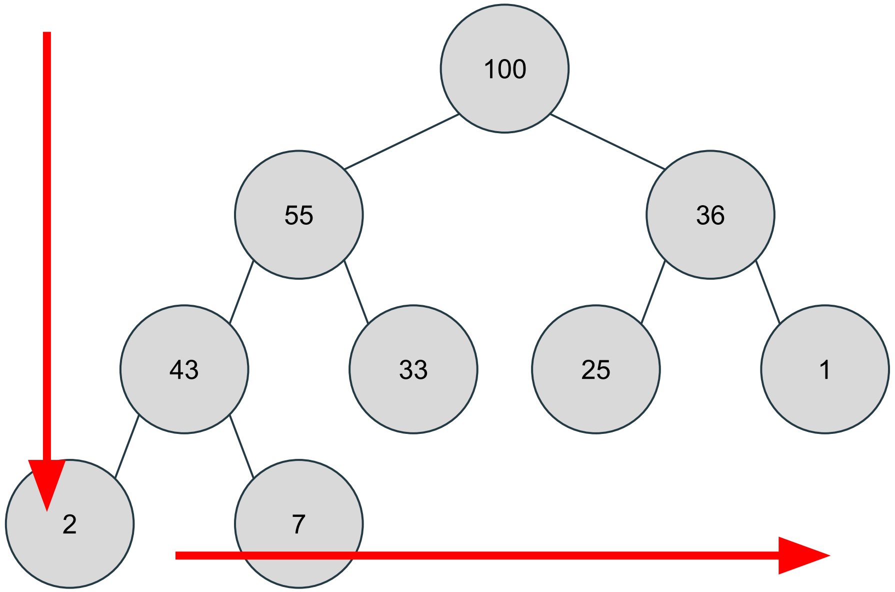
1. Shape property
2. Heap property



Rules (Constraints) of Binary Heap

Shape Property:

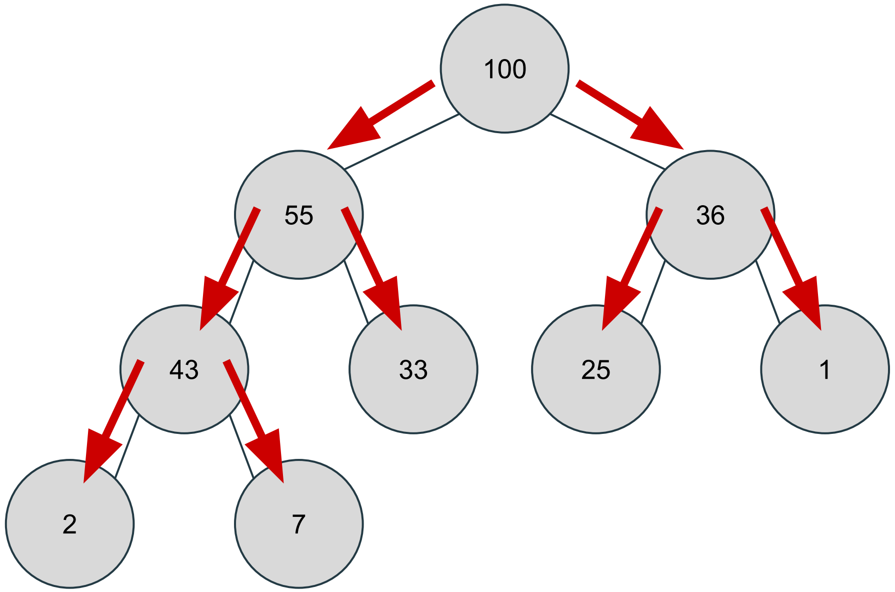
1. All levels of the tree, except possibly the last one (deepest) are fully filled, and,
2. If the last level of the tree is not complete, the nodes of that level are filled from left to right.



Rules (Constraints) of Binary Heap

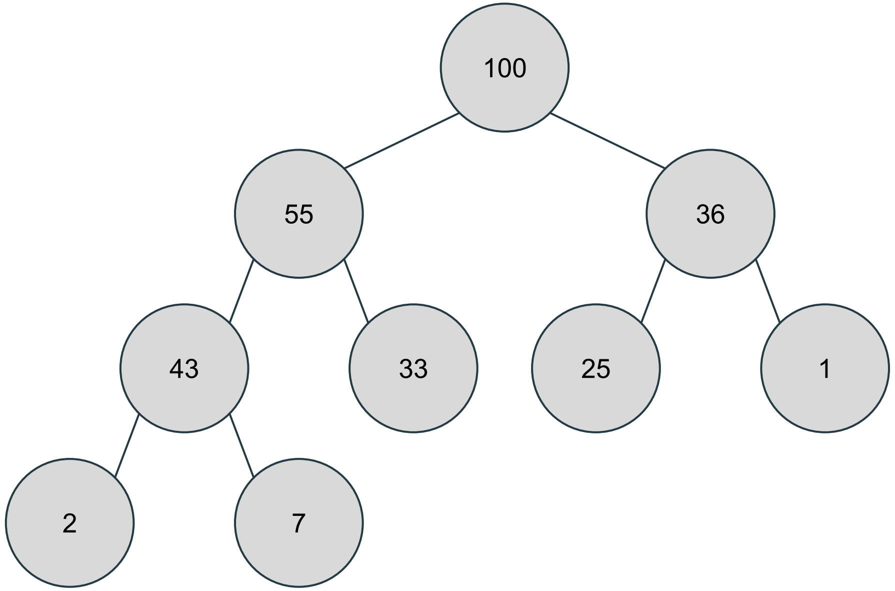
Heap Property:

1. the key stored in each node is either greater than or equal to (≥) or less than or equal to (≤) the keys in the node's children
2. Max-Heap & Min-Heap



Operations of Binary Heap

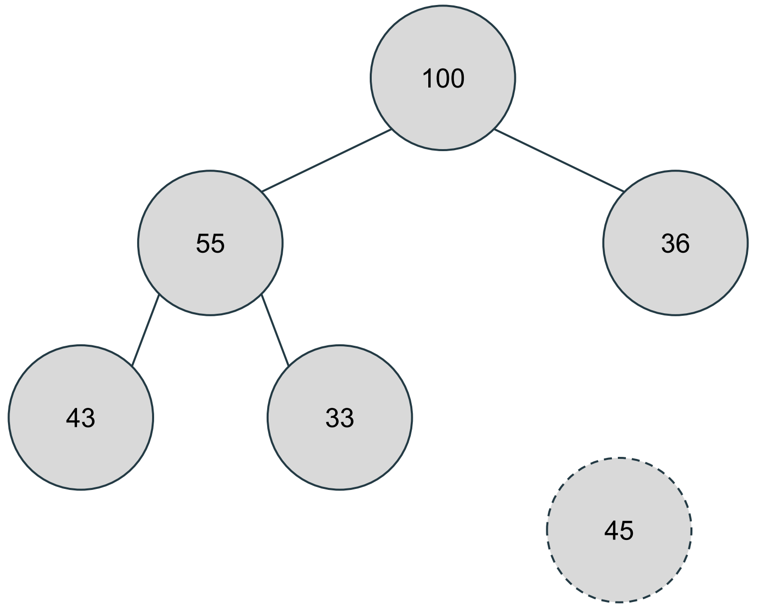
1. **Insert** a new node / element to the Binary Heap.
2. **Extract** the top node / element from the Binary Heap.



Operations of Binary Heap (Insert)

1. Add the node at the end of the tree according to the Shape Property*
2. Move the node up the tree to match the heap property**

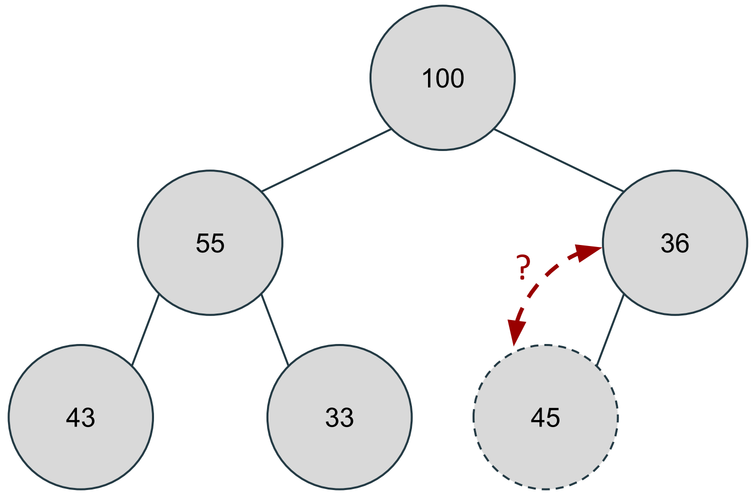
* Shape Property: Fill the tree from top to down, left to right.
** Heap Property: Parent nodes are >= to daughter nodes



Operations of Binary Heap (Insert)

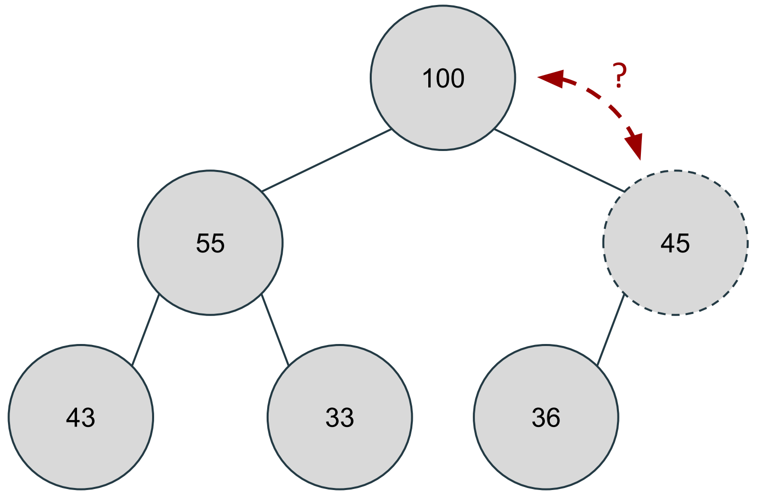
1. Add the node at the end of the tree according to the Shape Property*
2. Move the node up the tree to match the heap property**

* Shape Property: Fill the tree from top to down, left to right.
** Heap Property: Parent nodes are >= to daughter nodes



Operations of Binary Heap (Insert)

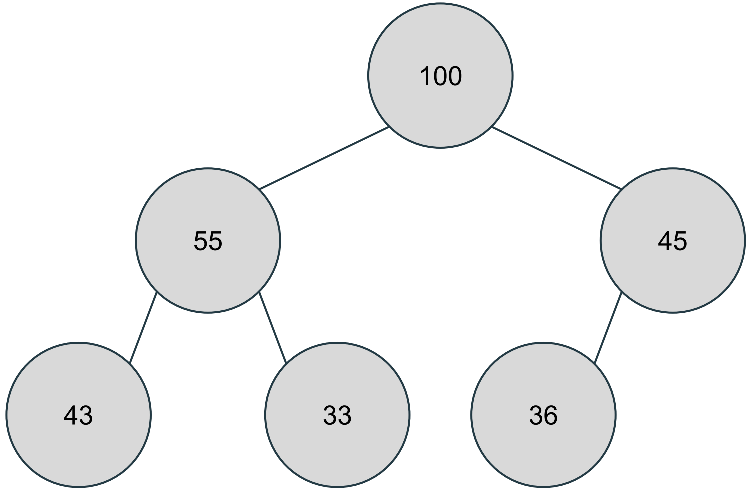
1. Add the node at the end of the tree according to the Shape Property*
2. Move the node up the tree to match the heap property**



* Shape Property: Fill the tree from top to down, left to right.
** Heap Property: Parent nodes are >= to daughter nodes

Operations of Binary Heap (Insert)

1. Add the node at the end of the tree according to the Shape Property*
2. Move the node up the tree to match the heap property**



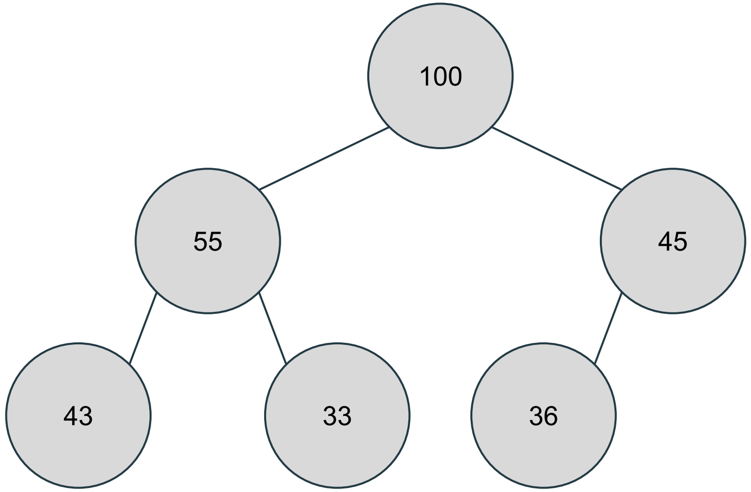
* Shape Property: Fill the tree from top to down, left to right.

** Heap Property: Parent nodes are >= to daughter nodes

Operations of Binary Heap (Extract)

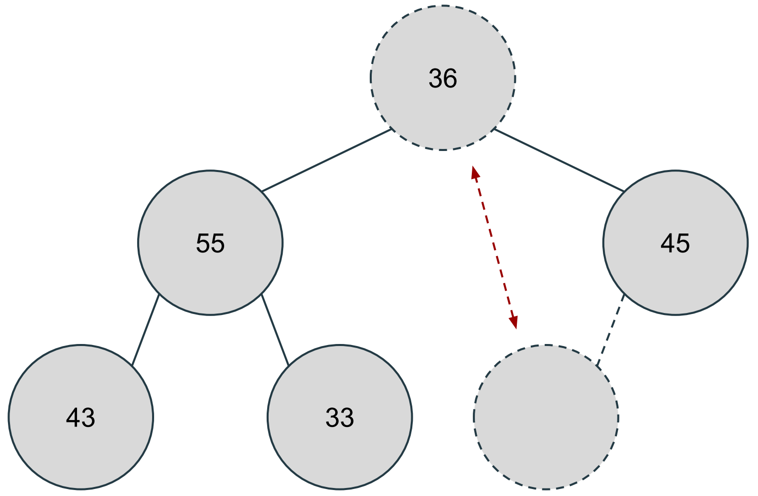
1. Move the last node to the position of the first node.
2. Move the "last" node down until tree fulfils the Heap Property*

** Heap Property: Parent nodes are >= to daughter nodes



Operations of Binary Heap (Extract)

1. Switch the biggest node with the last node.
2. Move the “last” node down until tree fulfils the Heap Property*

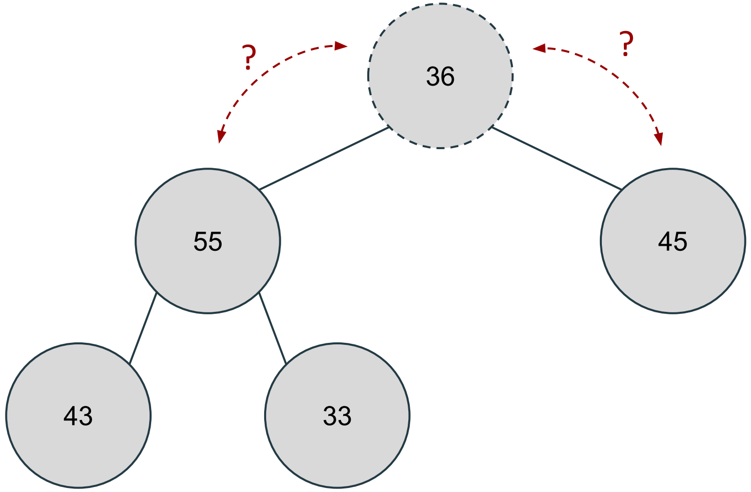


** Heap Property: Parent nodes are >= to daughter nodes

Operations of Binary Heap (Extract)

1. Switch the biggest node with the last node.
2. Move the “last” node down until tree fulfils the Heap Property*

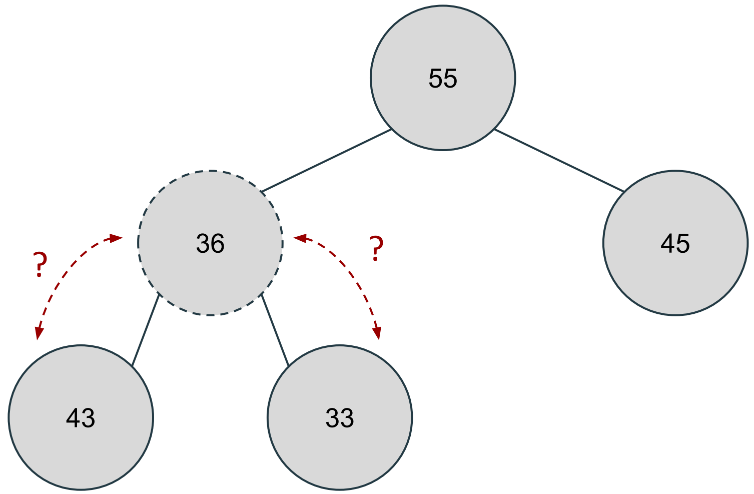
** Heap Property: Parent nodes are >= to daughter nodes



Operations of Binary Heap (Extract)

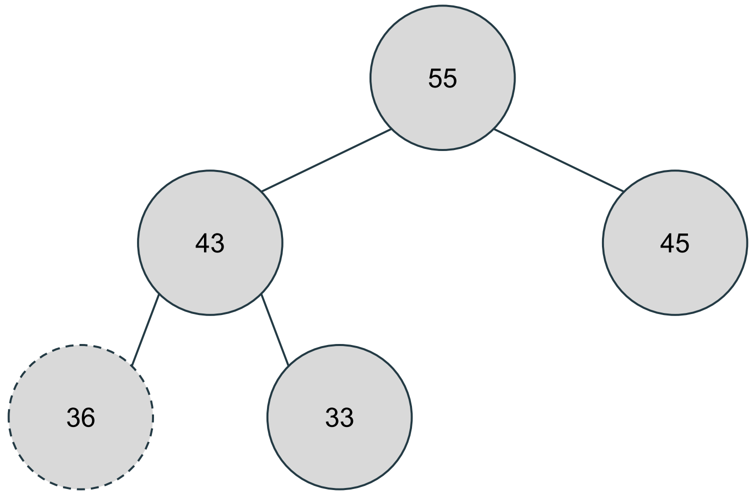
1. Switch the biggest node with the last node.
2. Move the “last” node down until tree fulfils the Heap Property*

** Heap Property: Parent nodes are >= to daughter nodes



Operations of Binary Heap (Extract)

1. Switch the biggest node with the last node.
2. Move the “last” node down until tree fulfils the Heap Property*

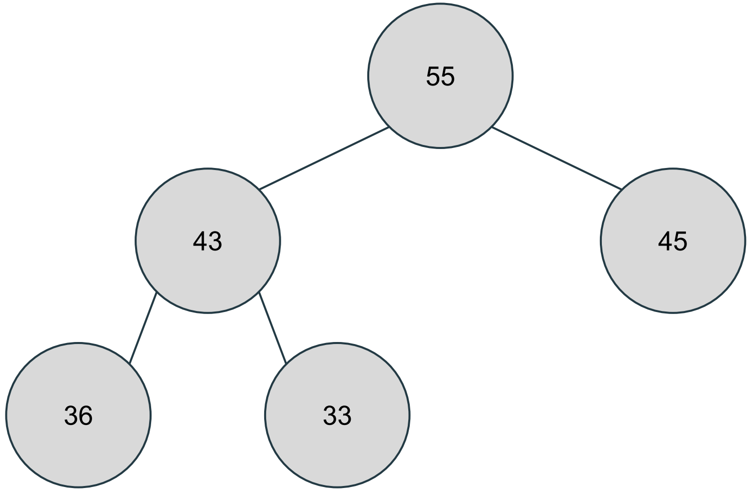


** Heap Property: Parent nodes are >= to daughter nodes

Operations of Binary Heap (Extract)

1. Switch the biggest node with the last node.
2. Move the “last” node down until tree fulfils the Heap Property*

** Heap Property: Parent nodes are >= to daughter nodes



Recap: Binary Heap

Rules / Constraints:

- Shape Property. Top to bottom; right to left.
- Heap Property. Consistent order of values throughout the tree. Parent nodes >= to daughter nodes or Parent nodes <= to daughter nodes.

Operations:

- Insert a new node
- Extract the top node

Writing Tests!

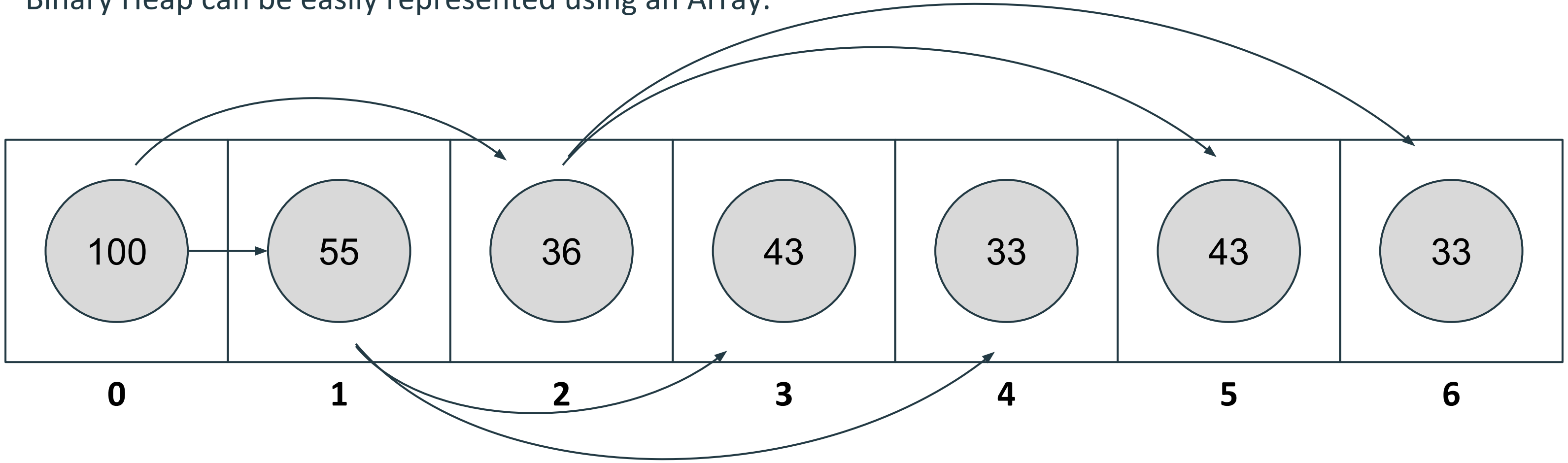
TDD! Why?

- TDD helps keep us from going Yak Shaving*
- *Yak shaving is what you are doing when you're doing some stupid, fiddly little task that bears no obvious relationship to what you're supposed to be working on, but yet a chain of twelve causal relations links what you're doing to the original meta-task.**

* https://www.hanselman.com/blog/YakShavingDefinedIllGetThatDoneAsSoonAsIShaveThisYak.aspx

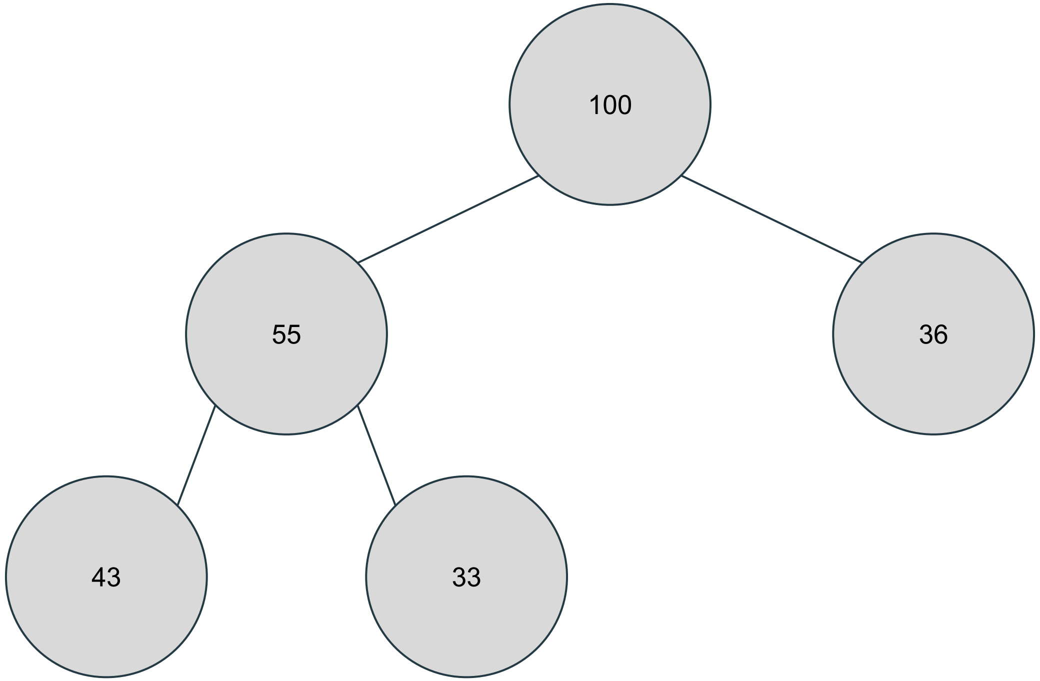
Writing Tests!

Binary Heap can be easily represented using an Array.

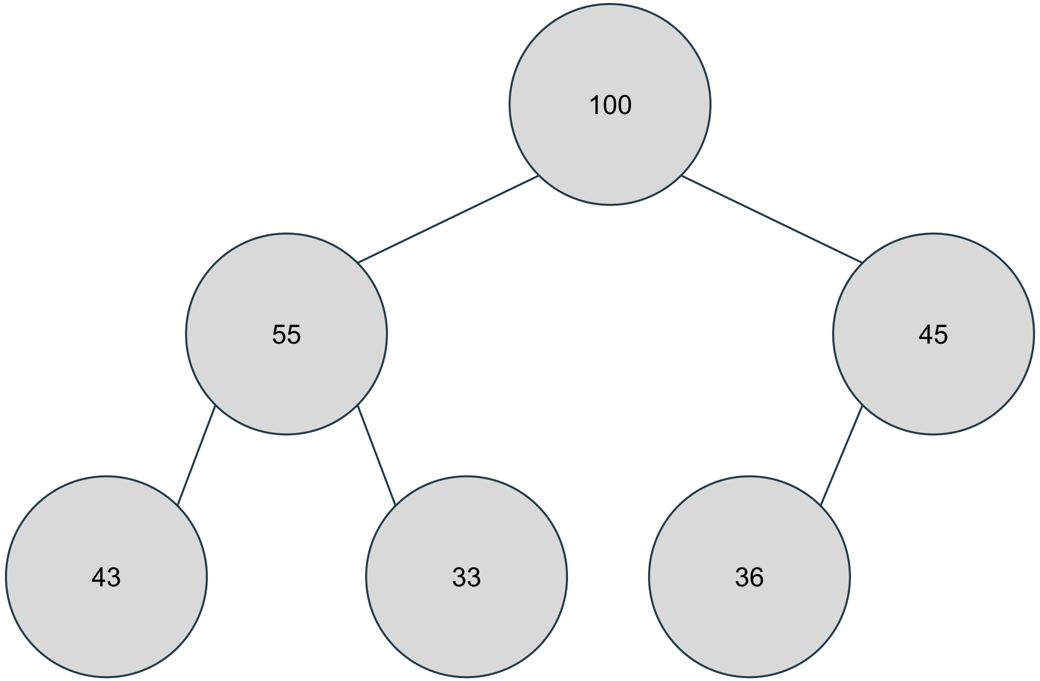


Let’s write some tests for real!

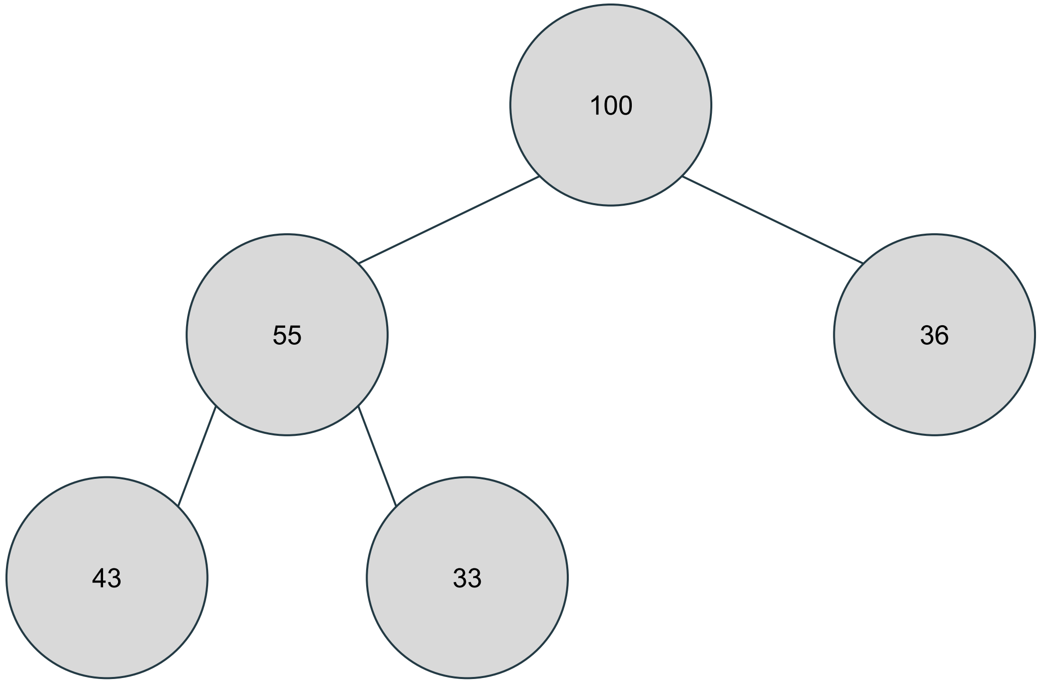
Test Data



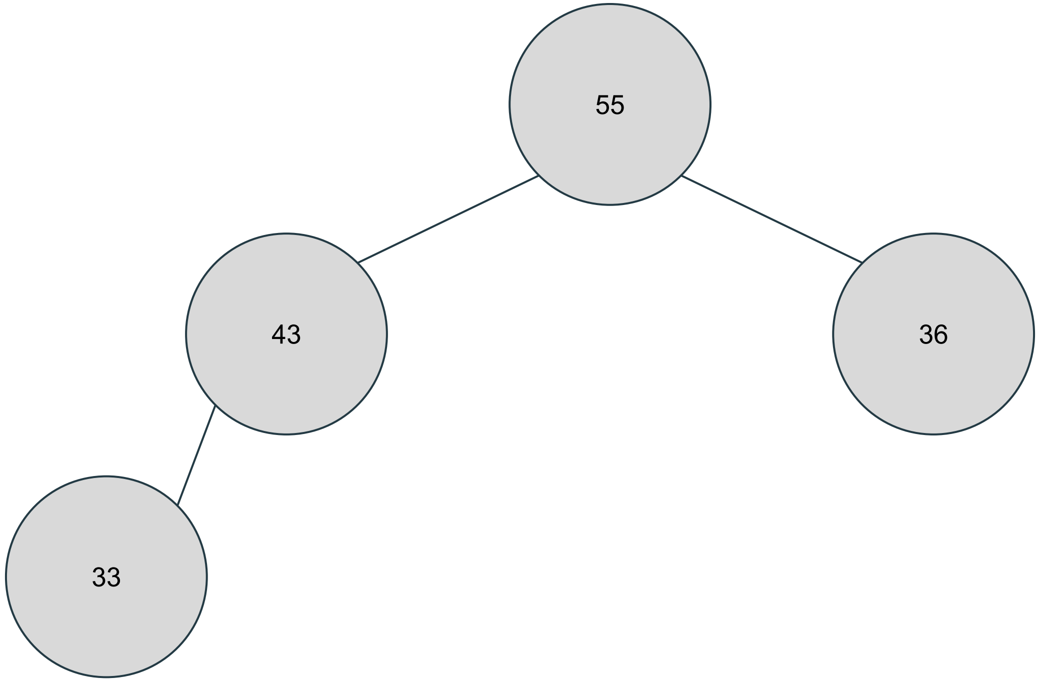
Insert(45)



Test Data



Extract



Solution

https://github.com/TayKangSheng/ruby-algorithms

What we have today: https://github.com/TayKangSheng/ruby-algorithms/blob/master/binary_heap.rb

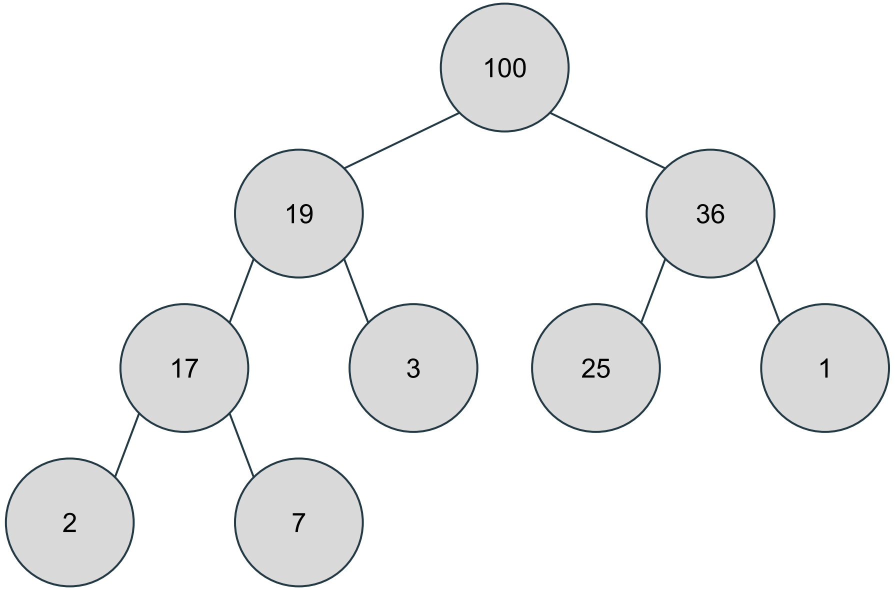
Solution: https://github.com/TayKangSheng/ruby-algorithms/blob/master/binary_heap_solution.rb

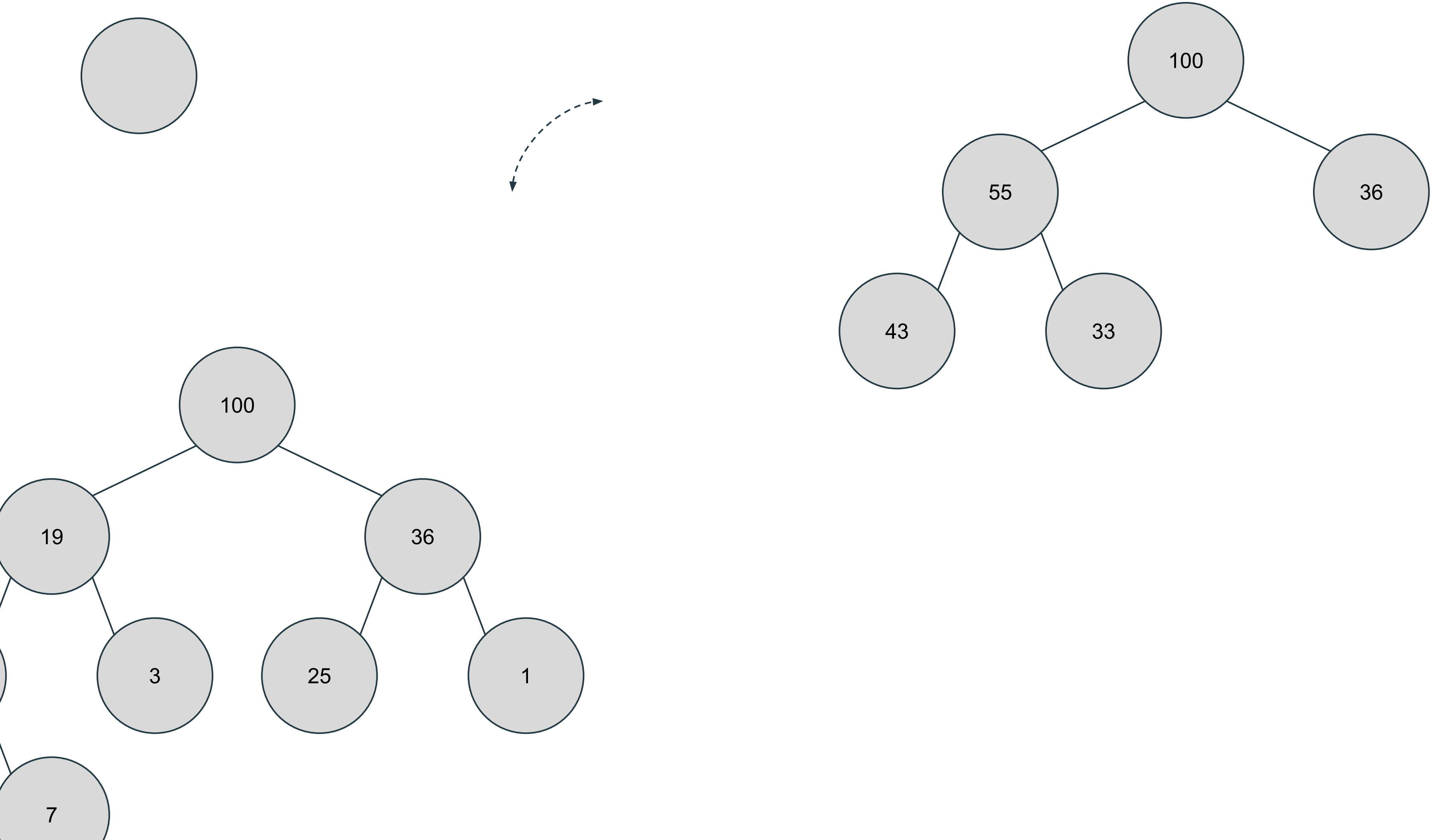
Feedbacks are welcome!

Twitter: @taykangsheng
GitHub: @taykangsheng

Title

Body




100
55
36
43
33
100
19
36
3
25
1
7