(Priority Queues)

Lab 5: Heaps/PQs

CSCI104

What is a Priority Queue?

- ~~FIFO (first in first out)~~ [queue]
- ~~LIFO (last in first out)~~ [stack]
- PQ: whatever in, “**best**” out

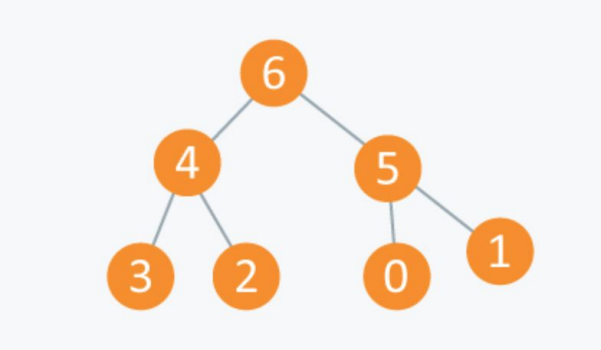
- implementation possibilities:
  - array (sorted or unsorted)
  - linked list
  - binary tree
  - heap** most efficient! → always keeps track of what is best

operations:
- push() - add any node
- pop() - remove the best node
- top() - return the value of the best node

So what is a heap?

- A tree where **every parent is better than its children** ← heap property
- Two representations:

as a complete d-ary tree



as an array

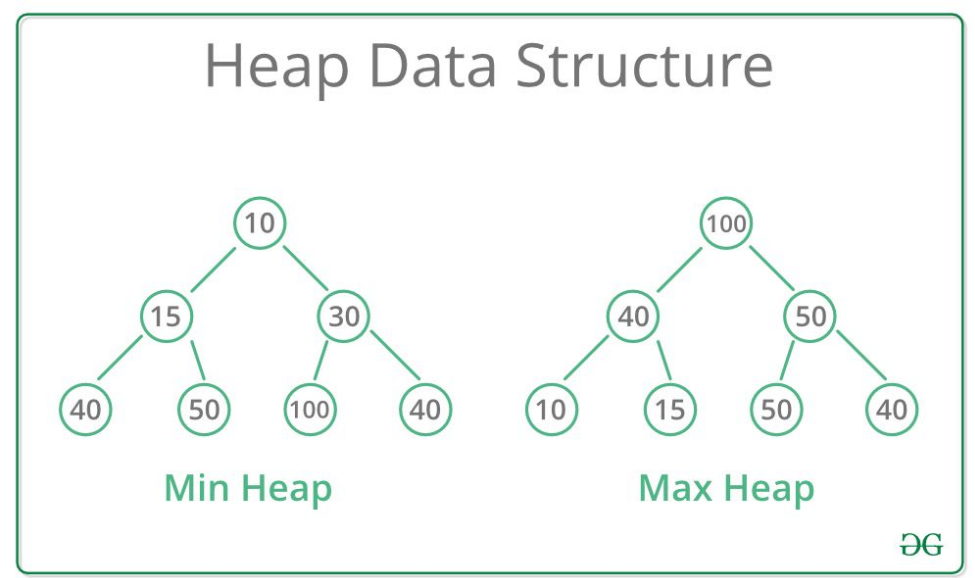
notice the indices: the index of the (parent of node i) = i/2

(this changes if the first element is at index 0!)

https://www.hackerearth.com/practice/notes/heaps-and-priority-queues/

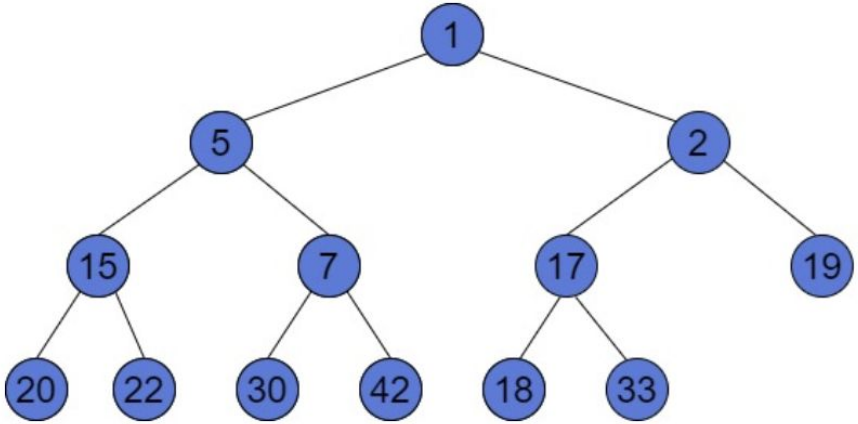
Heap Property

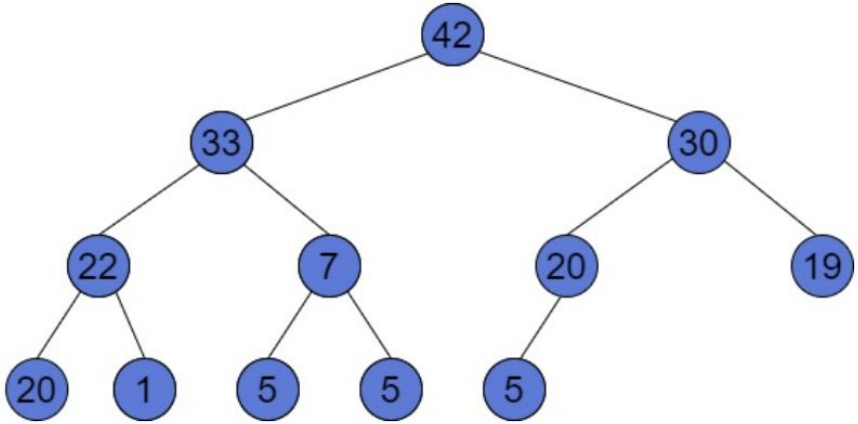
- Min Heap: node is less than or equal to all its children
- Max Heap: node is greater than or equal to all its children



Heaps

Which one is a min heap and which one is a max heap?



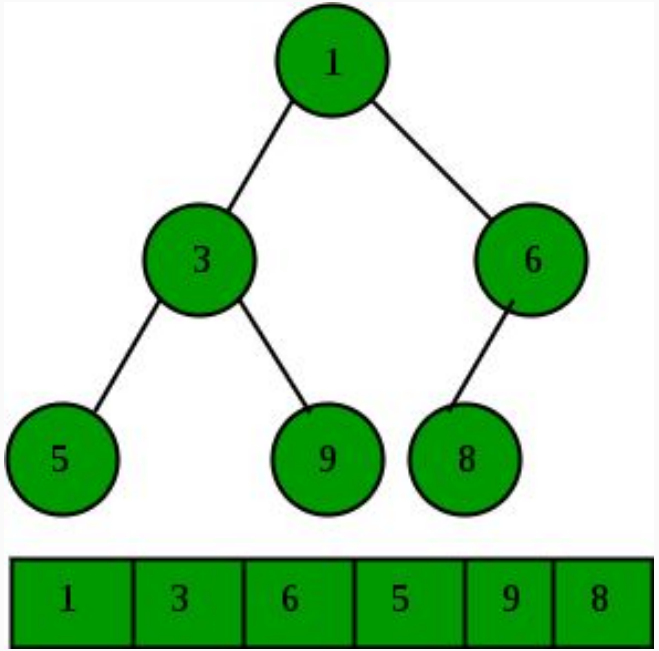


Questions to Think About

- Where in the tree is the “best” item at any given time?
  - THE ROOT
- For a binary heap, does it matter which child is on the left and which child is on the right? Why or why not?
  - NOPE, every parent just needs to be better than children
- In what data structure is there an ordering property between the left and right children and why is it necessary for that data structure and not in heaps?
  - BST, for the search operation

Store Heaps in Array

- Array starting with index 1
  - Parent(i) = i/2
  - Left_child(p) = 2p
  - Right_child(p) = 2p+1
- Array starting with index 0
  - Parent(i) = (i-1)/2
  - Left_child(p) = 2p+1
  - Right_child(p) = 2p+2
- Think about how this changes for 3-ary, 4-ary, or 5-ary heap



Heap operations

Push:

- where should we add the item?
- what index is that?
- how do we make sure the new node is in the right place when we're done?

Pop:

- what nodes/indices can we delete?
- how do we swap the "best" node into that node?
- how do we ensure we haven't swapped something into the wrong place?

Top: where does the "best" node live?

Heap operations (contd)

push():

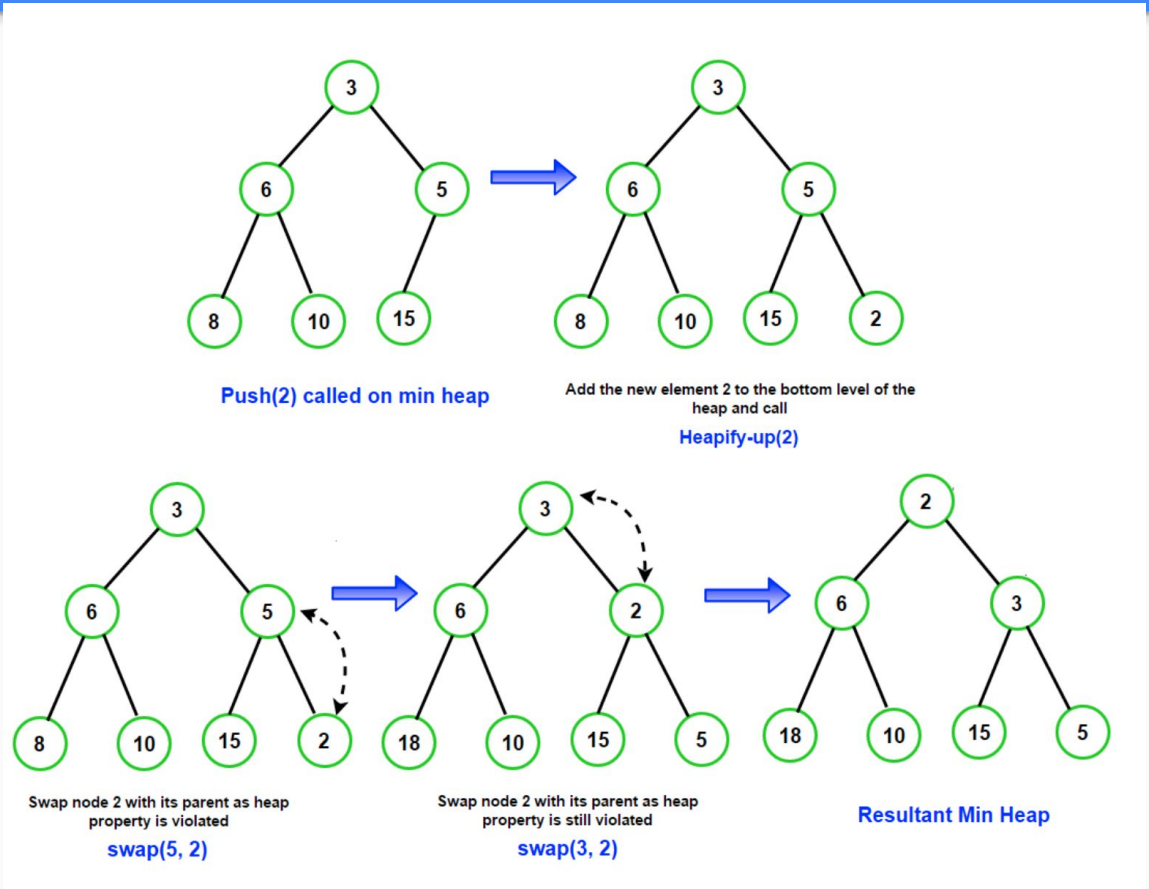
1. add it wherever it's easiest! (the first open index)
2. "trickle up" to find its correct spot (keep going while it's better than its parent)

In general:

1. Do whatever you're trying to do, ignoring the heap property
2. Now re-satisfy the heap property!! **

push

https://www.techiedelight.com/introduction-priority-queues-using-binary-heaps/



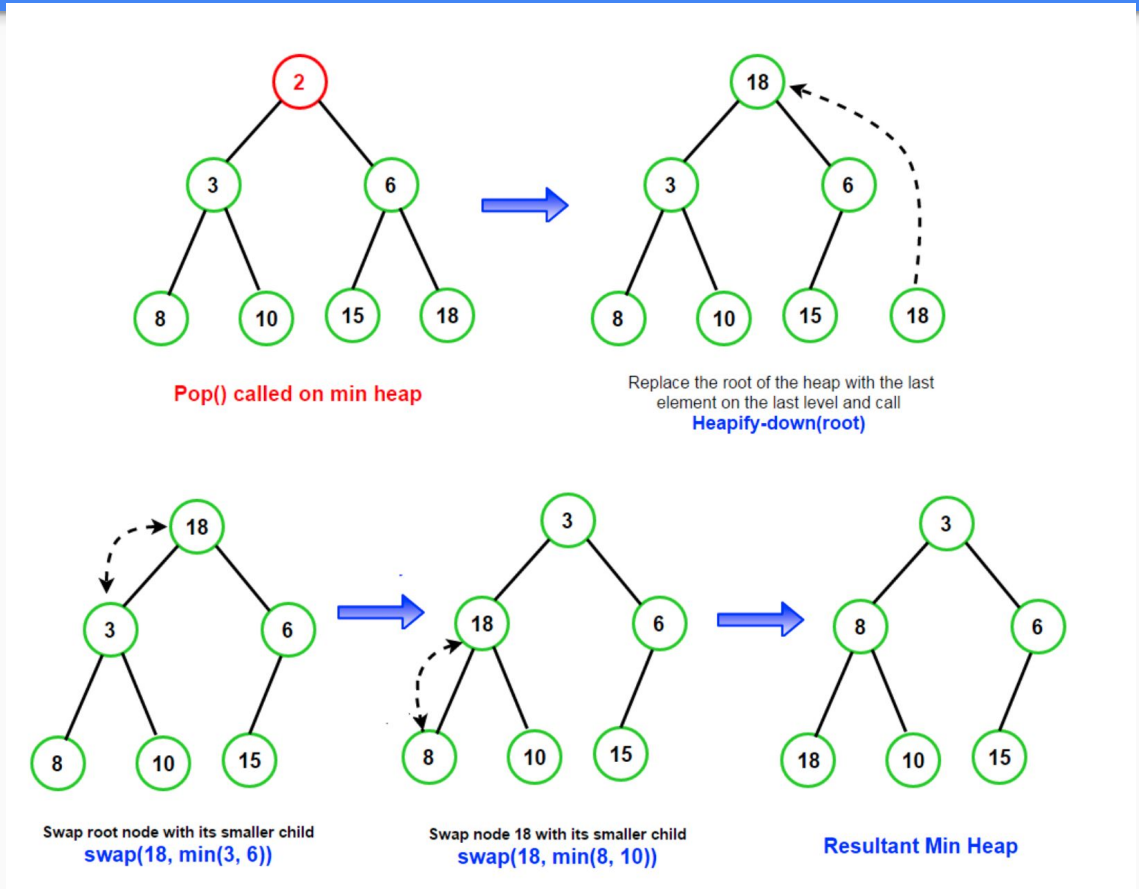
Heap operations (contd)

pop():

1. make it easy to remove (switch it into the last spot)
2. remove original best
3. “trickle down” to find correct spot for whatever you just swapped (keep going while it’s worse than its best child)

what would be the runtimes of these operations?

pop


2
3
6
8
10
15
18
Pop() called on min heap
18
3
6
8
10
15
18
Replace the root of the heap with the last element on the last level and call
Heapify-down(root)
18
3
6
8
10
15
Swap root node with its smaller child
swap(18, min(3, 6))
3
18
6
8
10
15
Swap node 18 with its smaller child
swap(18, min(8, 10))
3
8
6
18
10
15
Resultant Min Heap

The Lab

part 1: implement pop()

- think about indices, the heap property
- use vector operations and swap
- check your implementation with

```
make heap_test
```

`vector.push_back`

`vector.pop_back`

```
std::swap(data[parent], data[left])
```

part 2: a simple game using the heap you just created

- run `make test`

show both for checkoff!