COMP_SCI 214: Data Structures and Algorithms

Priority Queues

PROF. SRUTI BHAGAVATULA

Announcements

- Reminder: Homework 3 was due Tuesday
  - We will release grade reports tomorrow
  - Resubmission is due Tuesday
  - HW3 self-eval to be released tomorrow
  - Self-eval is due Tuesday

Recall: Worklists

- Say you need a program that:
  - Keeps track of “items” you need to handle
  - Allows you to fetch a single piece of “item” to handle next
- You may want to fetch:
  - The last item in **Last-in-first-out (LIFO)**
  - The earliest item in **First-in-first-out (FIFO)**

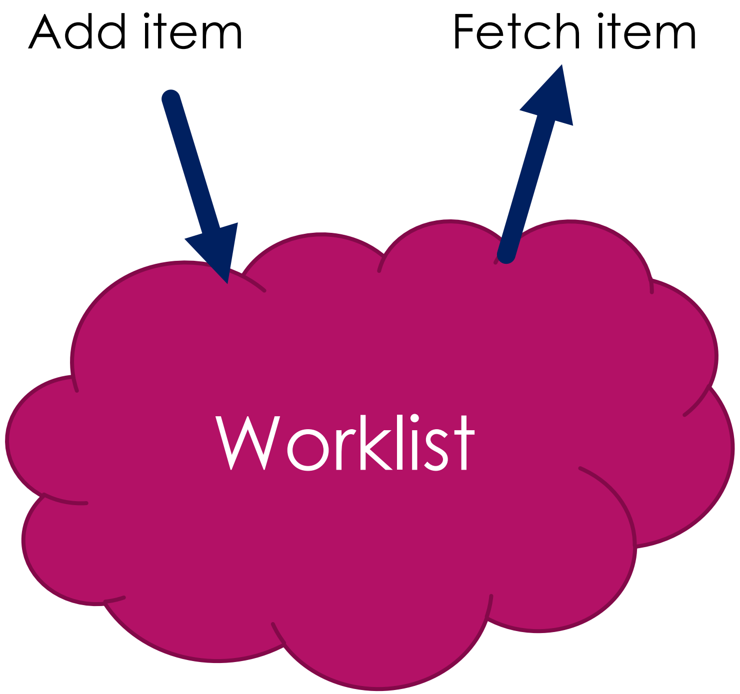
What if you need to…

- Identify which patient to help in a hospital emergency room?
- Identify which tasks are the most pressing in your TODO list?
- Decide which homework assignment you should be working on next?
- Decide which maintenance tasks to handle based on severity?
- Identify and fix the most harmful bugs in your software production system?

- Are stacks and queues good enough worklists for these?

Recall: Worklists

- Say you need a program that:
  - Keeps track of "items" you need to handle
  - Allows you to fetch a single piece of "item" to handle next
- You may want to fetch:
  - ~~The last item in **Last-in-first-out (LIFO)**~~
  - ~~The earliest item in **First-in-first-out (FIFO)**~~
  - The highest priority item

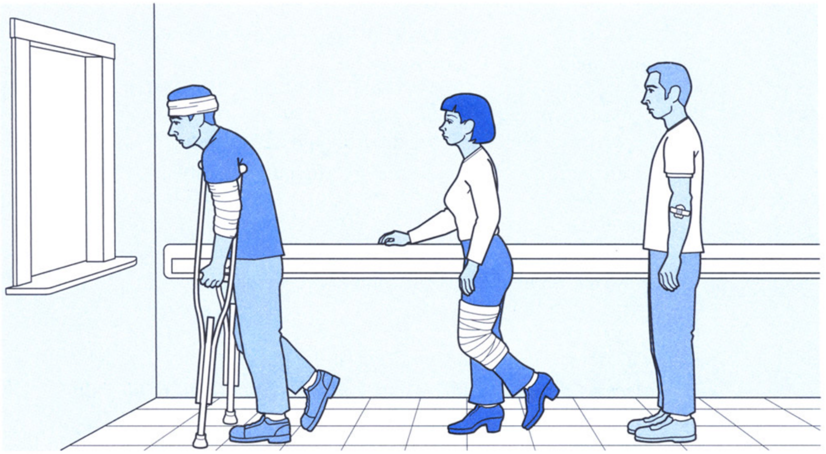


A new ADT: Priority queue

Priority queue (PQ)

- **Data:** set of task or item objects

- **Operations:**
  - Insert an item into the priority queue
  - Fetch the highest priority element
  - Check if there are any items left in the priority queue



Priority queue ADT interface

- Abstract values look like (note the sorting) →

Highest priority

2	Brain damage
5	Heart attack
17	Fever
89	Cold
…	…

```
interface PRIORITY_QUEUE[T]:

  def empty?(self) -> bool?
  def insert(self, priority: num?, value: T) -> NoneC
  def remove_min(self) -> T
```

Properties of a PQ

- Behavior of the PQ:
  - PQ keeps priority-value pairs sorted by priority
  - `remove_min` finds and removes pair with highest priority and returns the value
    - Lowest priority value (or rank) = highest urgency (it's confusing, I know!)
- There can be variants:
  - Separate `find_min` and `remove_min` functions
  - No separate priority value; priority determined from task data itself or combined structure that has both

Other use of priority queues

- PQs are also useful for finding the shortest path between vertices in a graph
- We'll see this one soon – going back to graphs after next week!

Operating on a priority queue

- **Managing a hospital ER:**
  - New patient came in:

```
pq.insert(3, "cardiac arrest")
```

  - Which patient to call in next?

```
pq.remove_min()
```

BD

Highest priority

~~2~~	~~Brain damage~~
5	Heart attack
17	Fever
89	Cold
…	…

3 CA

Operating on a priority queue

- **Managing homework deadline:**
  - Just got assigned a new HW with a deadline of tomorrow

    `pq.insert(1, ”Music HW”)`

  - Which HW should I start?

    `pq.remove_min()`

Highest priority

2	Physics HW
3	214 HW
4	Chemistry HW
5	Math HW
…	…

~~1 music~~

Deciding priorities

- Priorities don't have to always be numbers!
- Can be any data type → just need to specify a comparator function (like with sorting)
  - You'll see this in HW5
  - Allows specifying what's a "priority" given the specific types of data
- These functions can have simple or complex logic to compare two tasks → up to the client!

Priority queue laws

- $\{ pq = \langle\, \langle \}$ $pq.empty?()$ $\Rightarrow$ True $\{\}$
- $\{ pq = \langle\, p_0 : e_0, \ldots, p_n : e_n \,\langle \wedge \forall i\ p_i \leq p_{i+1}\}$ $pq.empty?()$ $\Rightarrow$ False $\{\}$

- $\{ pq = \langle\, p_0 : e_0, \ldots, p_{k-1} : e_{k-1}, p_{k+1} : e_{k+1}, p_n : e_n \,\langle \wedge \forall i\ p_i \leq p_{i+1}\}$
  $pq.insert(p_k, e_k)$ $\Rightarrow$ None
  $\{pq = \langle\, p_0 : e_0, \ldots, p_{k-1} : e_{k-1}, p_k : e_k, p_{k+1} : e_{k+1}, p_n : e_n \,\langle \}$

- $\{ pq = \langle\, p_0 : e_0, p_1 : e_1, \ldots, p_n : e_n \,\langle \wedge \forall i\ p_i \leq p_{i+1}\}$
  $pq.remove\_min()$ $\Rightarrow$ e_0
  $\{pq = \langle\, p_1 : e_1, \ldots, p_n : e_n \,\langle \wedge \forall i\ p_i \leq p_{i+1}\}$

Implementing priority queues

How can we implement a priority queue?

- Any suggestions using data structures we’ve seen so far?
  - Sorted linked list
  - Unsorted linked list

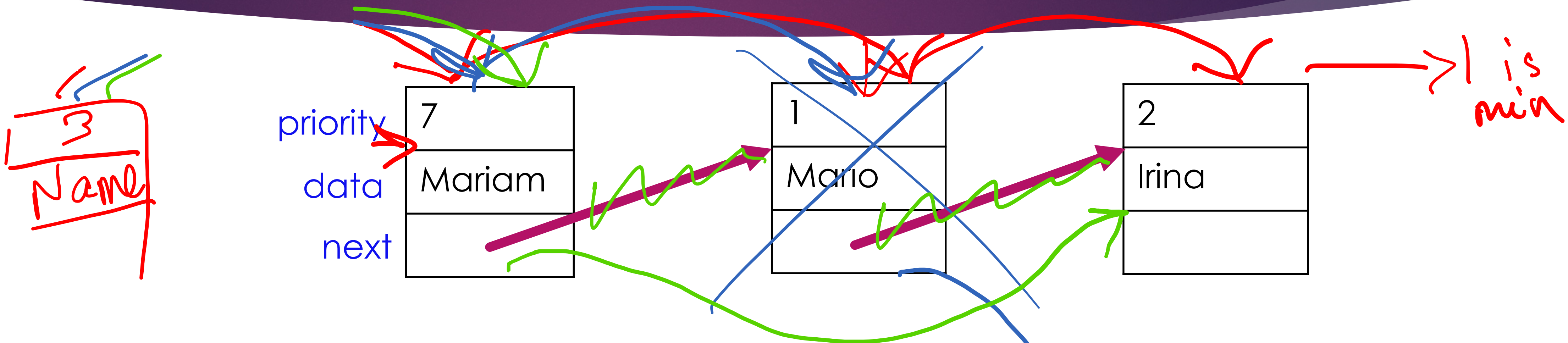
```
interface PRIORITY_QUEUE[T]:

  def empty?(self) -> bool?
  def insert(self, priority: num?, value: T) -> NoneC
  def remove_min(self) -> T
```

Candidates to implement PQ ADT

	Sorted linked list	Unsorted linked list
insert	O(n)	O(1)
remove_min	O(1)	O(n)

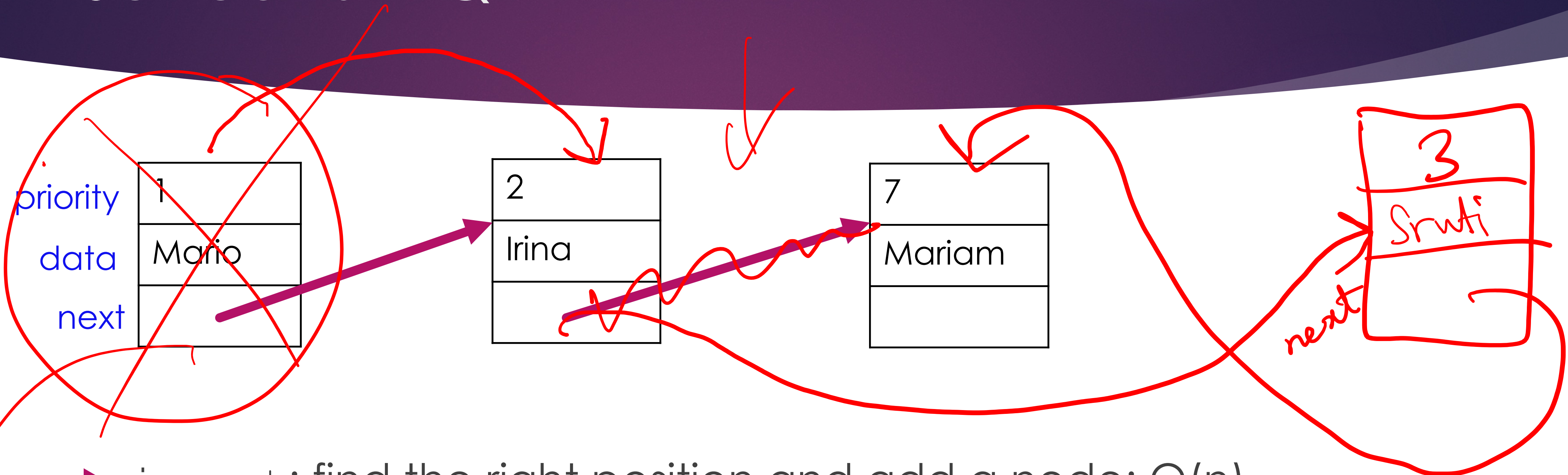
`n` is the number of elements in the priority queue

Unsorted list PQ



- `insert`: add a node at the start in O(1)
  - Inserting Sven with priority 3
- `remove_min`: search for minimum priority in O(n)
  - Look through list and then remove Mario

Sorted list PQ



- `insert`: find the right position and add a node: O(n)
  - Inserting Sven with priority 3
- `remove_min`: remove first element in O(1)
  - Remove Mario from start of list

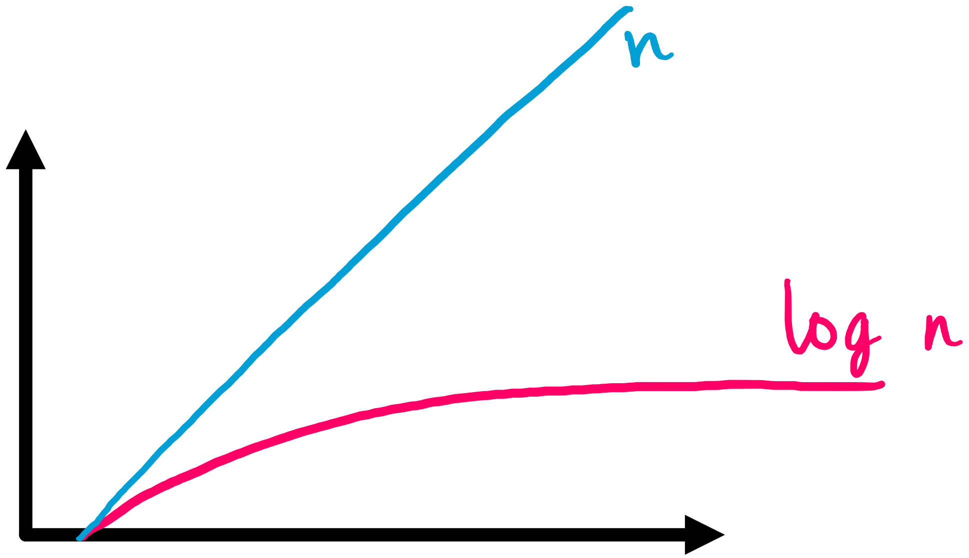
So what do we choose?

- Pick your poison:
  - Which operation do we need more?
  - Make sure that is O(1)
- What if I want both operations?!
  - Neither is great
  - Can we get the best of both worlds?
  - Almost! We can't get O(1) for both **but** we can do O(log n)!

	Sorted list	Unsorted list	??
insert	O(n)	O(1)	O(log n)
remove_min	O(1)	O(n)	O(log n)

Recall: $O(\log n)$ is a big deal

n	10	100	1000	10K	100K	1M	10M	100M
$\log_2 n$	3.3	6.6	10	13.3	16.6	20	23.3	26.6

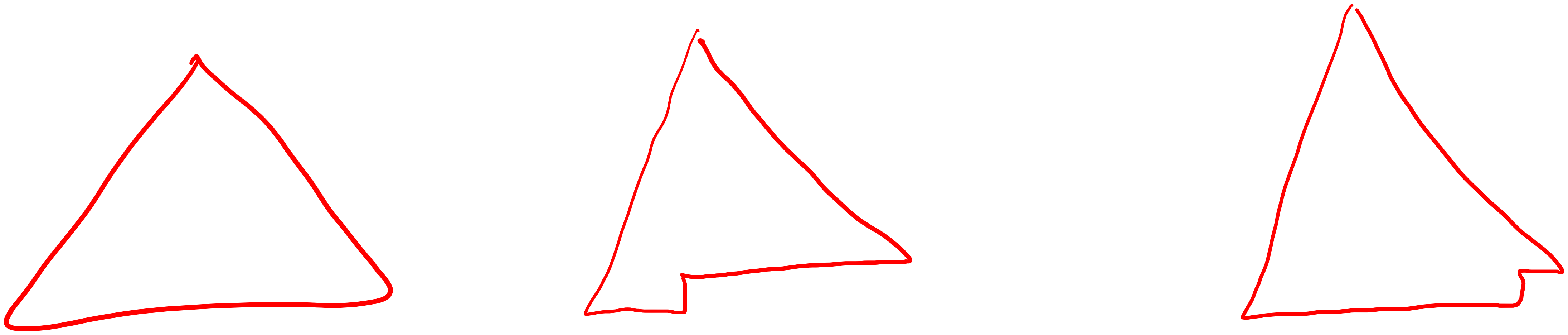


Introducing the **binary heap**

- A new data structure!
- A binary heap is a complete binary tree that is heap-ordered
  - Complete: A k-ary tree is complete if every level is full of nodes. Last level can have nodes missing, but must be filled left to right
  - Heap-ordered: A tree is heap-ordered if each element is less than or equal to its children
- The above two are invariants for a binary heap

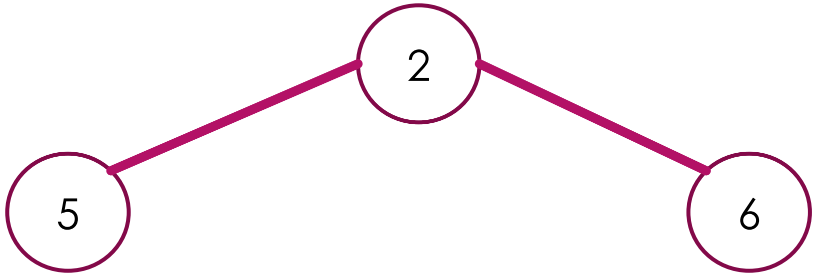
Shape invariant (completeness)

- Possible shapes a binary heap represents



Ordering invariant (heap-ordered)

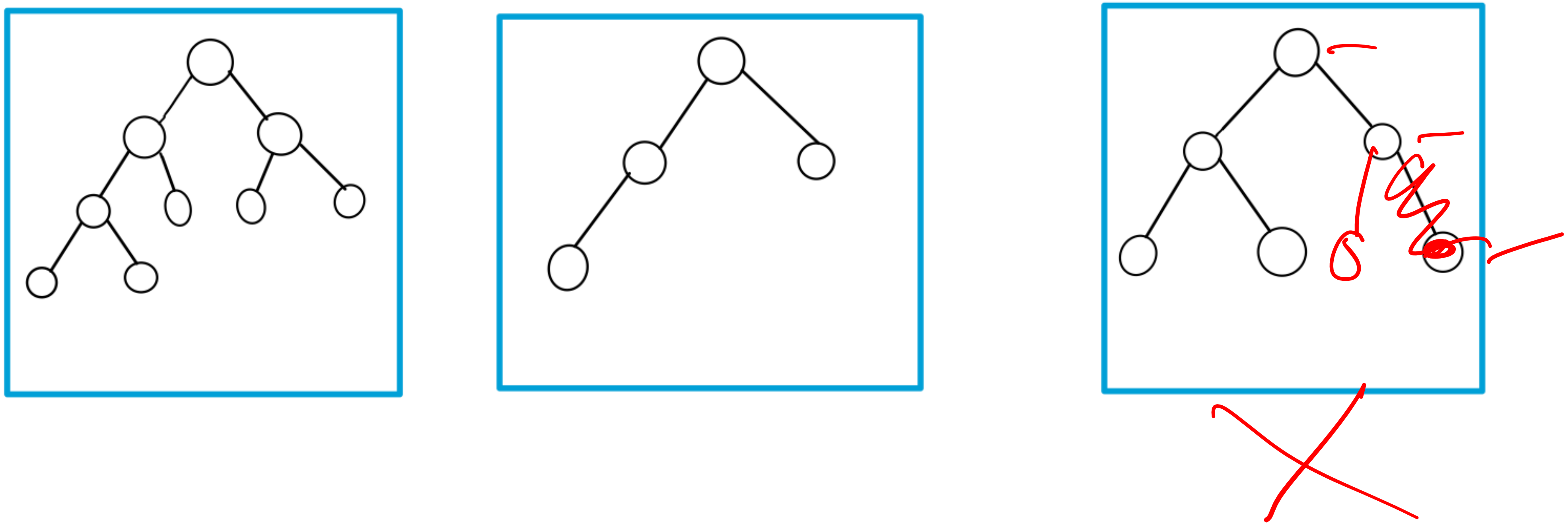
- Our invariant specifies a min-heap



- Another variant: a max-heap where highest value is on top
  - But min-heap is most common
  - With custom comparators, min or max is irrelevant and priority order is up to the client

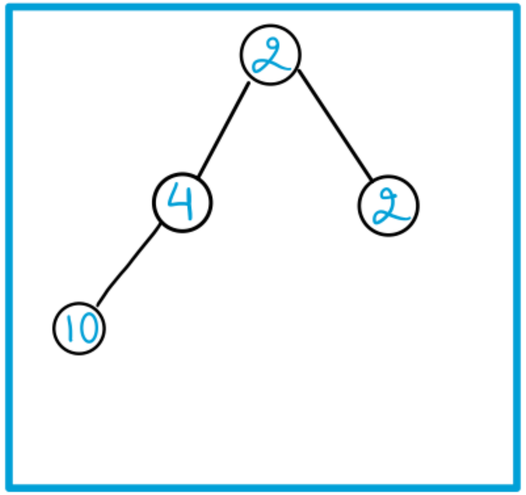
Game: 2 valid heaps and an invalid one?

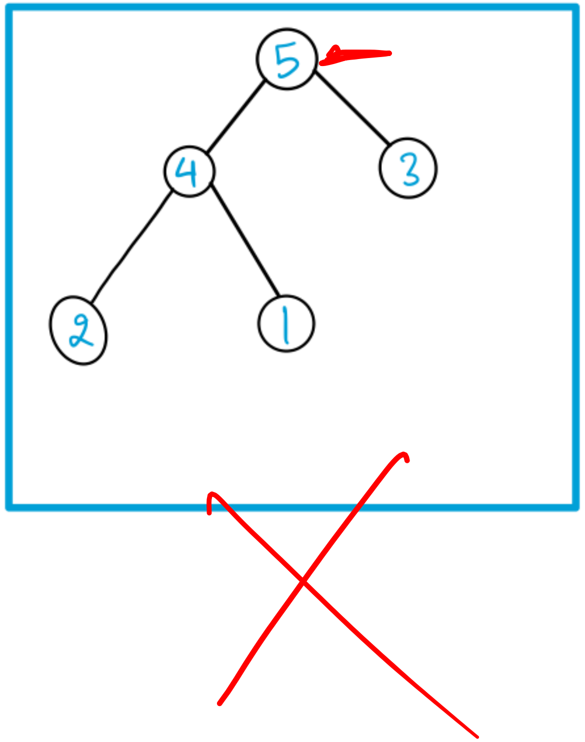
- Based on shape invariant

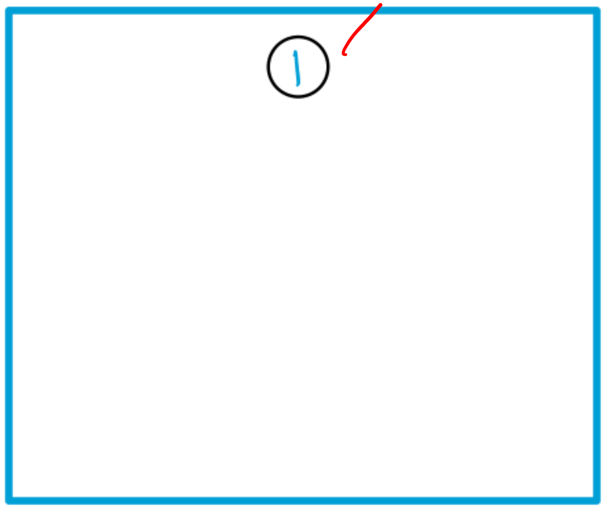


Game: 2 valid heaps and an invalid one?

- Based on ordering invariant







Binary heap for priority queue

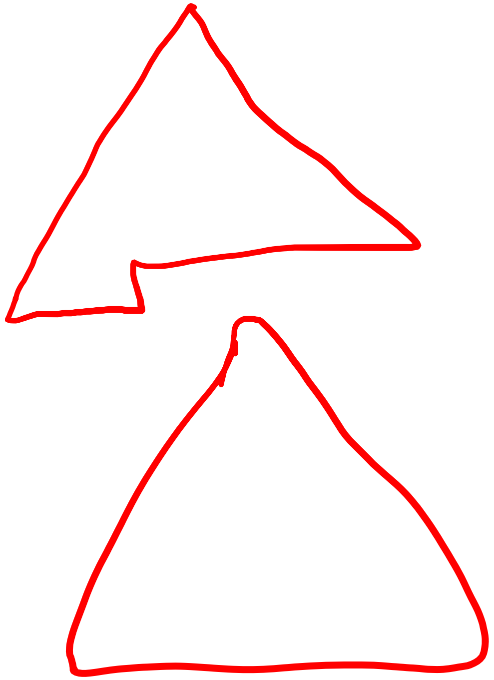
- ADT operations:
  - Let's look at the 2nd two
  - `empty?()` is trivial by maintaining a count

```
interface PRIORITY_QUEUE[T]:

  def empty?(self) -> bool?
  def insert(self, priority: num?, value: T) -> NoneC
  def remove_min(self) -> T
```

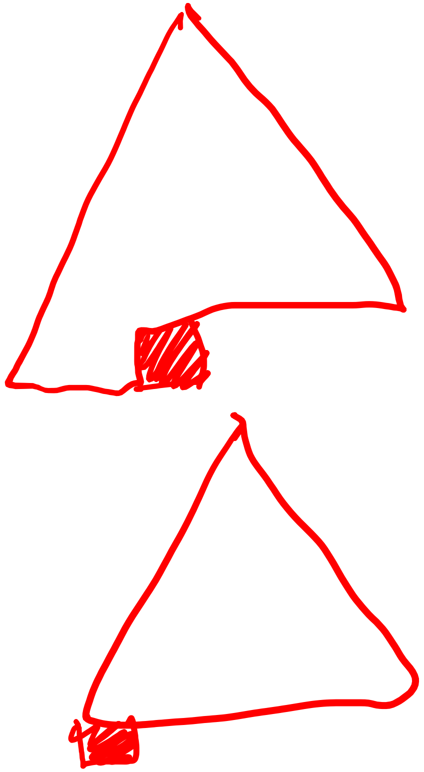
- Both shape and ordering invariant need to hold by the end of each operation

`insert` operation on binary heap

- To preserve shape invariant, after insertion into

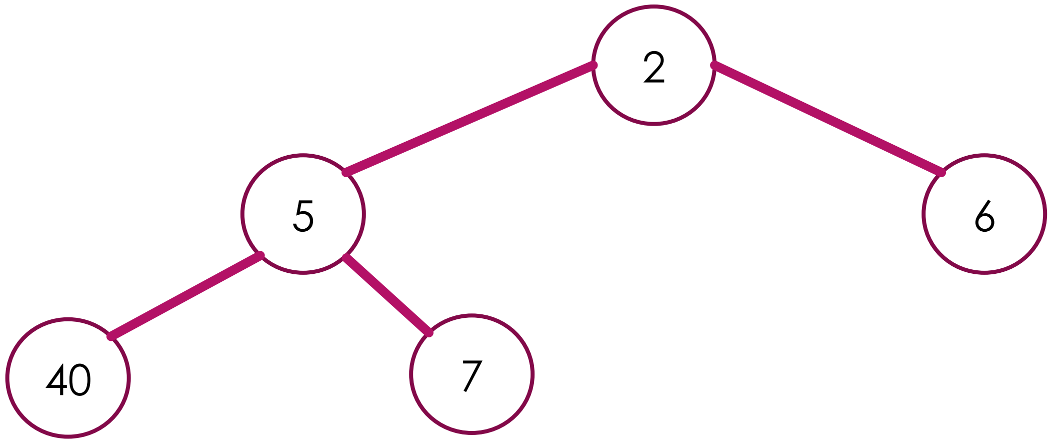


the new shape should look like



`insert` operation on binary heap

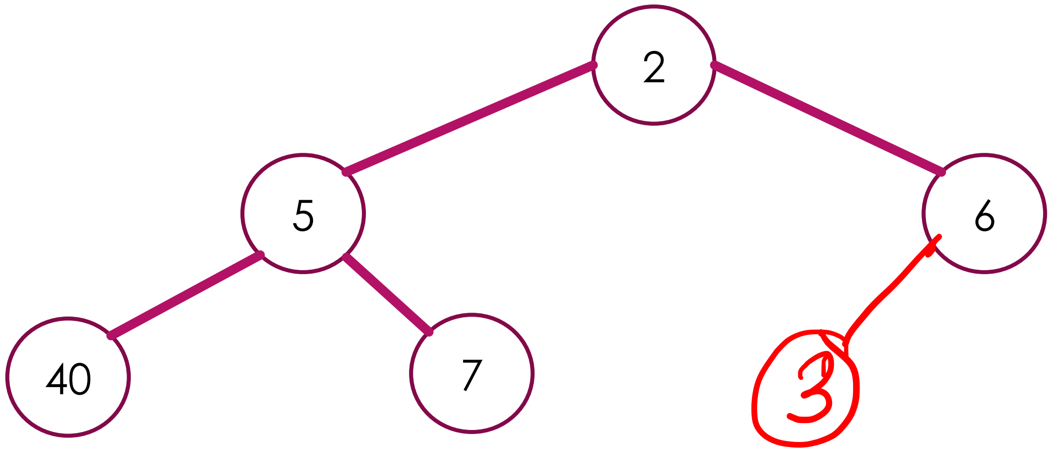
1. First preserve shape invariant
2. Then restore ordering invariant



`insert` operation on binary heap

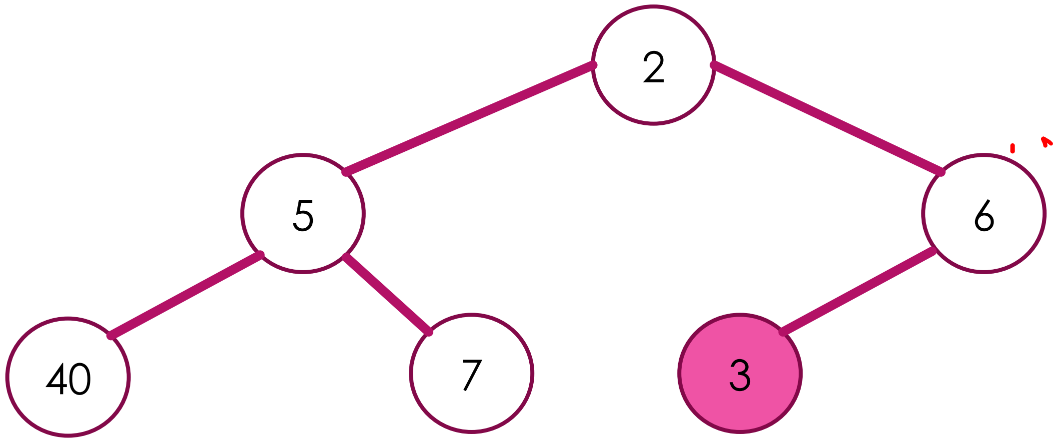
- Insert element with priority 3

1. First preserve shape invariant
   - Where can we insert 3 to preserve this?
2. Then restore ordering invariant



`insert` operation on binary heap

- Insert element with priority 3



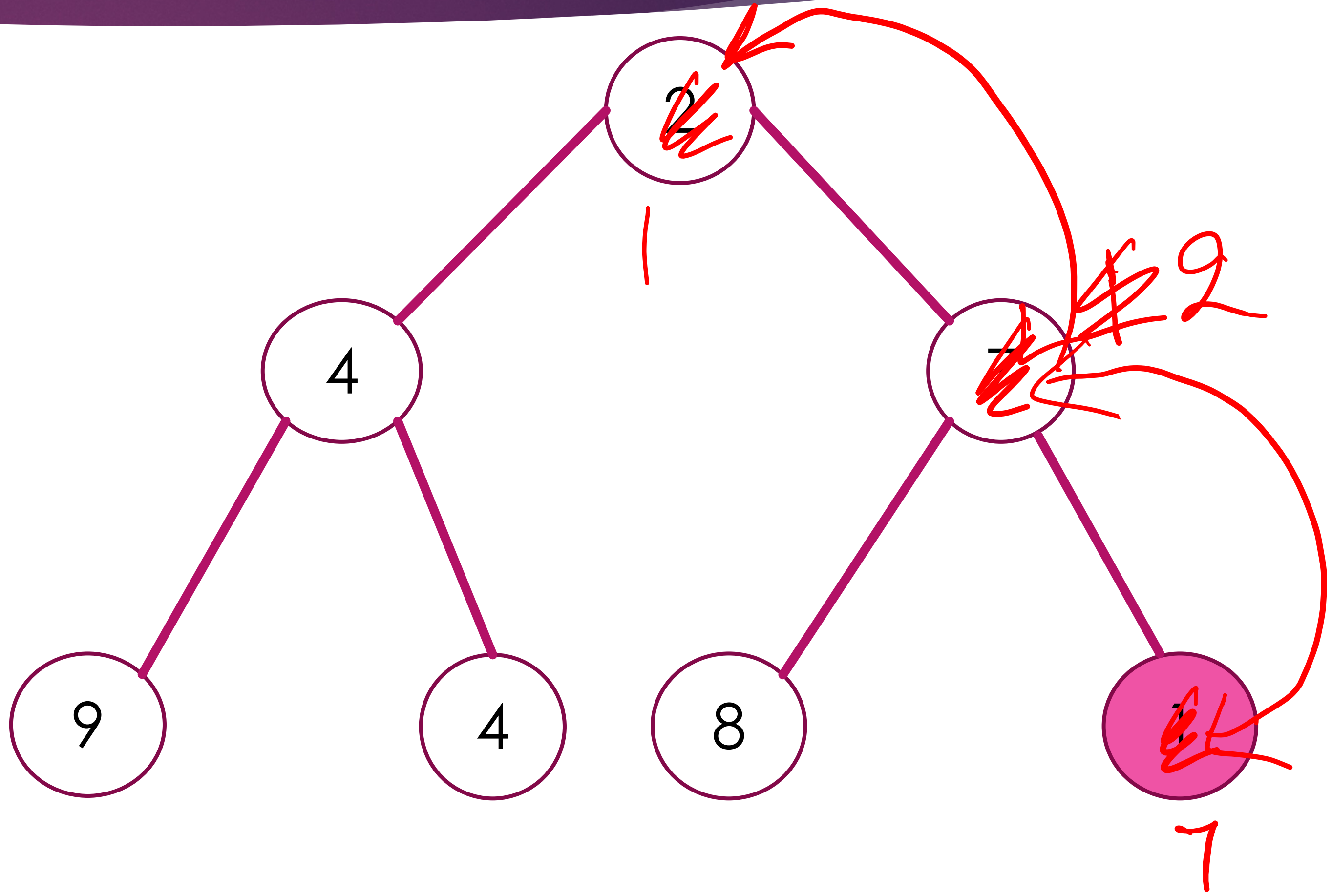
1. First preserve shape invariant
   - Insert element in spot expected by shape invariant
2. Then restore ordering invariant
   - Swap newly inserted node with parent until no longer a violation of ordering invariant **(bubbling up)**

`insert` in action

1. Insert element in spot expected by shape invariant

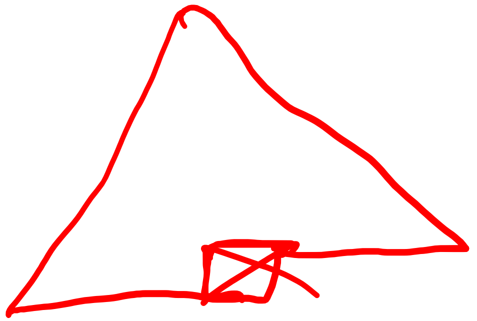
2. **Bubble up** until there is no longer an ordering invariant violation

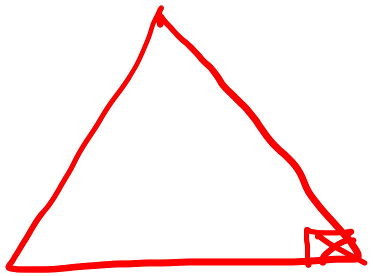
$7 \le 1$? No → Swap
$2 \le 1$? No → Swap



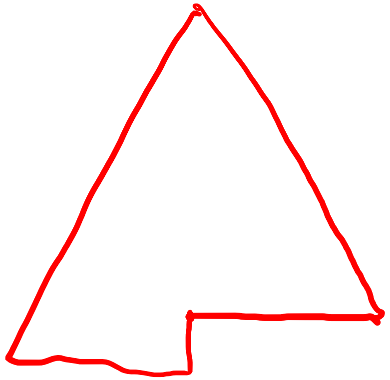
`remove_min` operation on binary heap

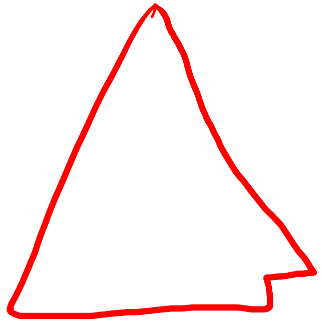
- To preserve shape invariant, after removing the min from





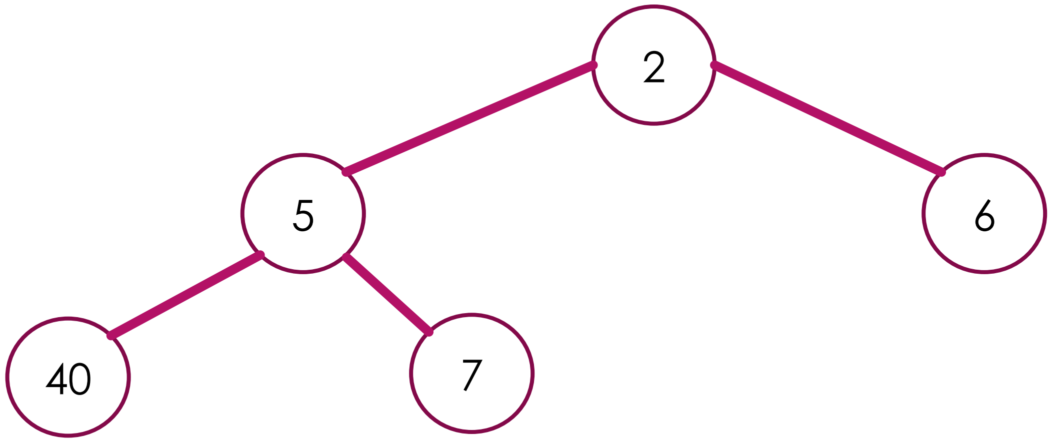
the new shape should look like





`remove_min` operation on binary heap

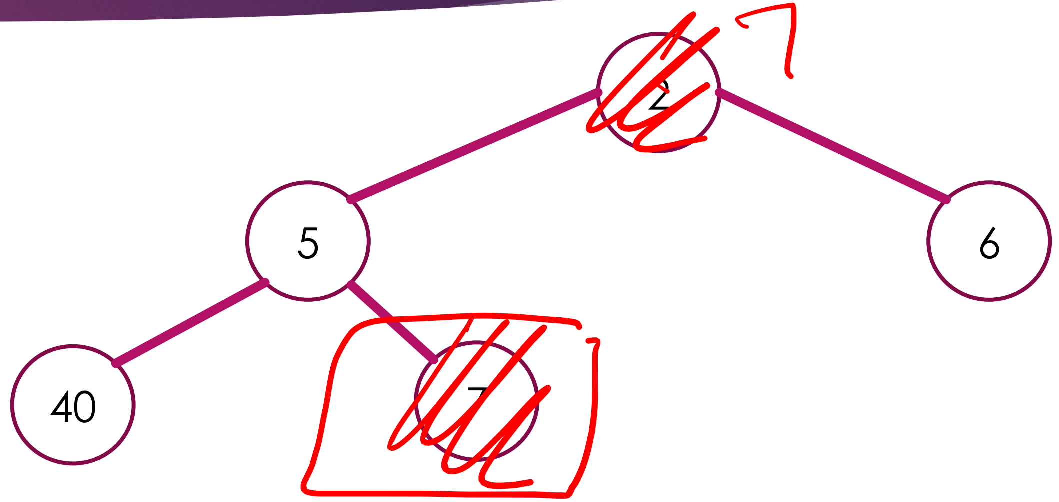
1. First preserve shape invariant
2. Then restore ordering invariant



`remove_min` operation on binary heap

Return 2

- Remove root

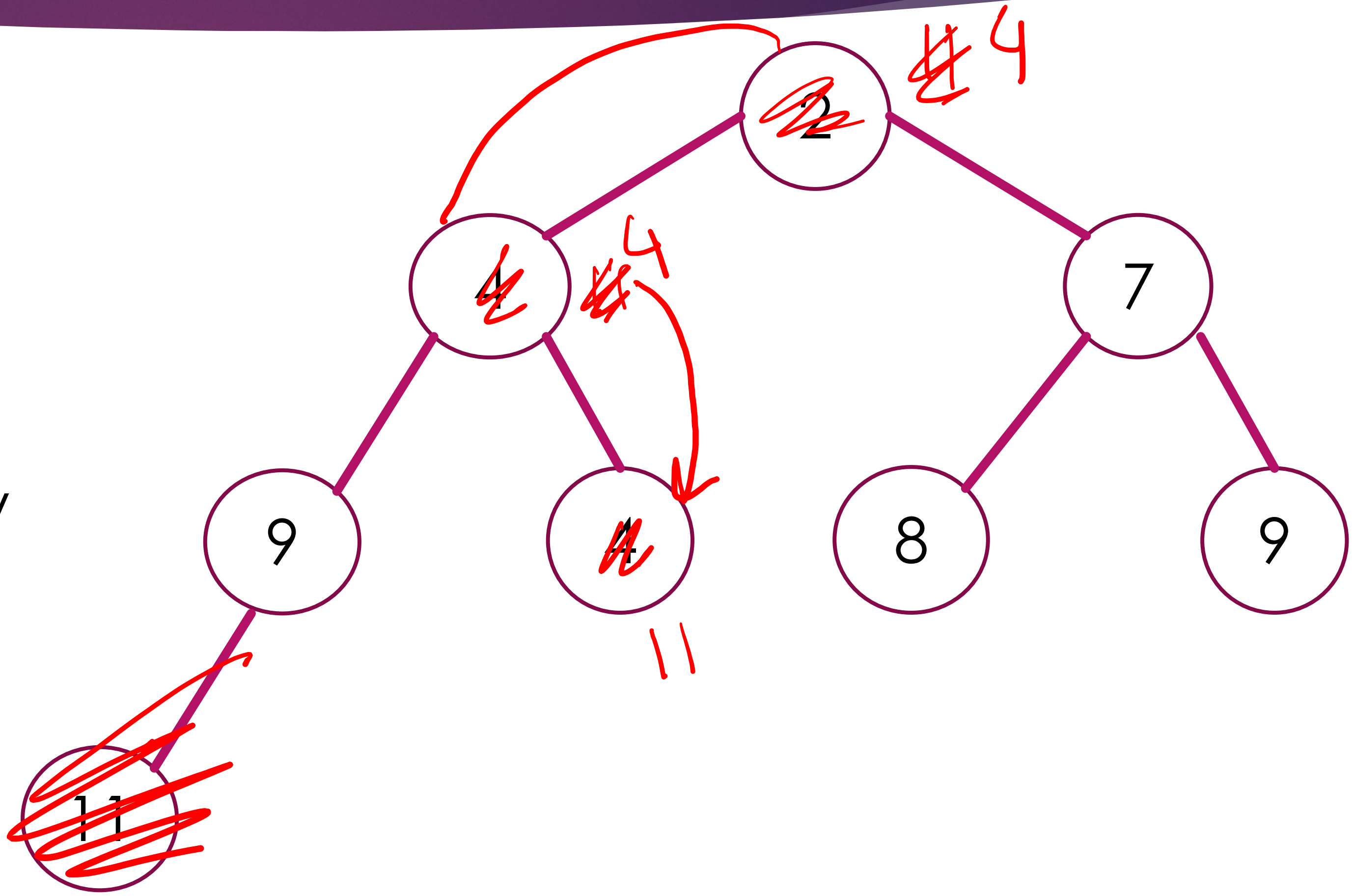


1. First preserve shape invariant
    - Remove "last" element in lowest level and put in root's place
2. Then restore ordering invariant
    - Swap new root with children until there is no longer an ordering invariant violation **(percolating down)**

`remove_min` in action

1. Remove "last" element in lowest level and put in root's place

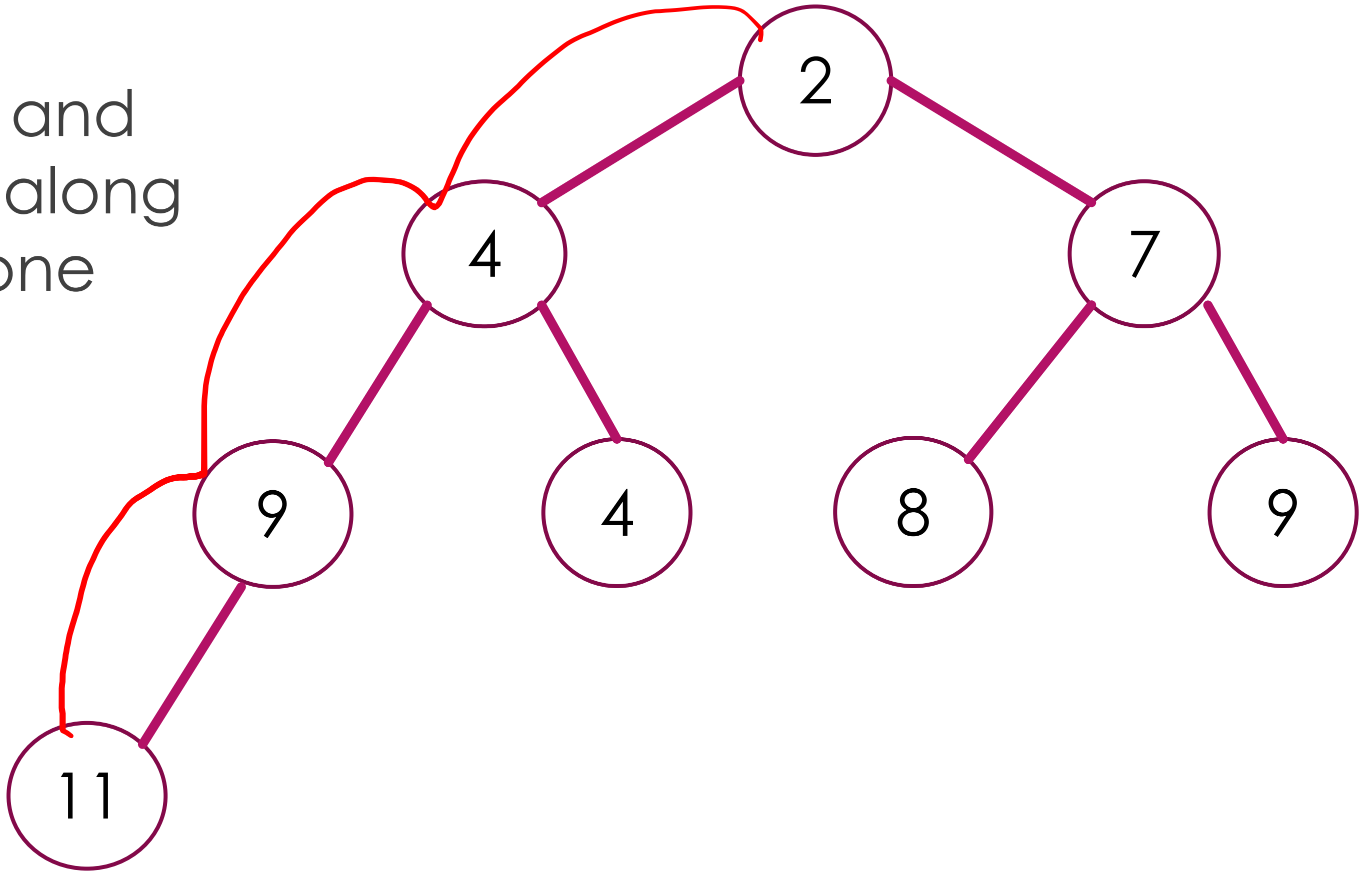
2. **Percolate down** from new root by swapping with smallest child until no longer an ordering invariant violation



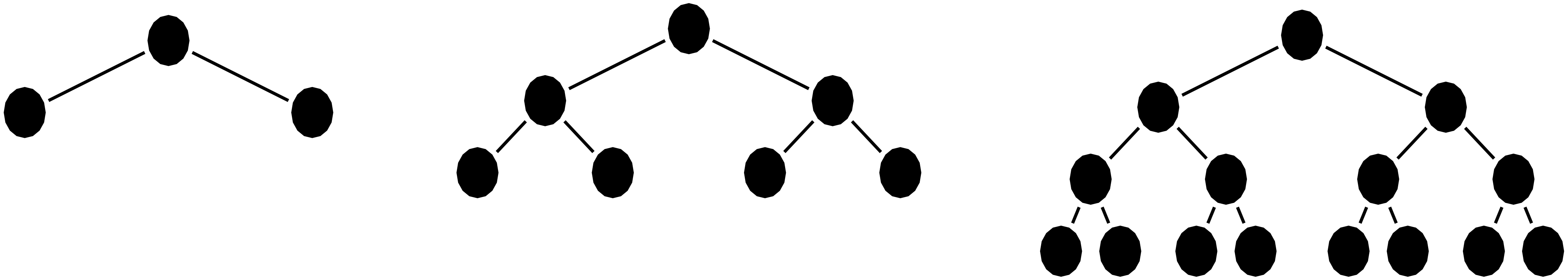
Binary heap time complexity

- In the worst case, inserting and removing makes changes along the whole path between one leaf and the root

- How long is this path?
  - The height of the tree



Height of a complete tree



Complete tree with _ nodes	Has height
3 (= $2^2 – 1$)	2
7 (= $2^3 – 1$)	3
15 (= $2^4 – 1$)	4
…	…

Height of a complete tree

Height of a complete tree with n nodes is O(log n)!
More on trees and height later in the quarter.

[illegible]	[illegible]
15 (= $2^4 - 1$)	4
…	…

Complexities for priority queue

	Sorted list	Unsorted list	Binary heap
insert	O(n)	O(1)	O(log n)
remove_min	O(1)	O(n)	O(log n)

Pause

- Any questions or anything unclear?

Other uses of binary heaps

- Heap sort!
- Put all your elements in a heap and keep finding and removing the next minimum or maximum
  - You'll do this in Homework 5
- On your own: what would be the complexity of this sort?

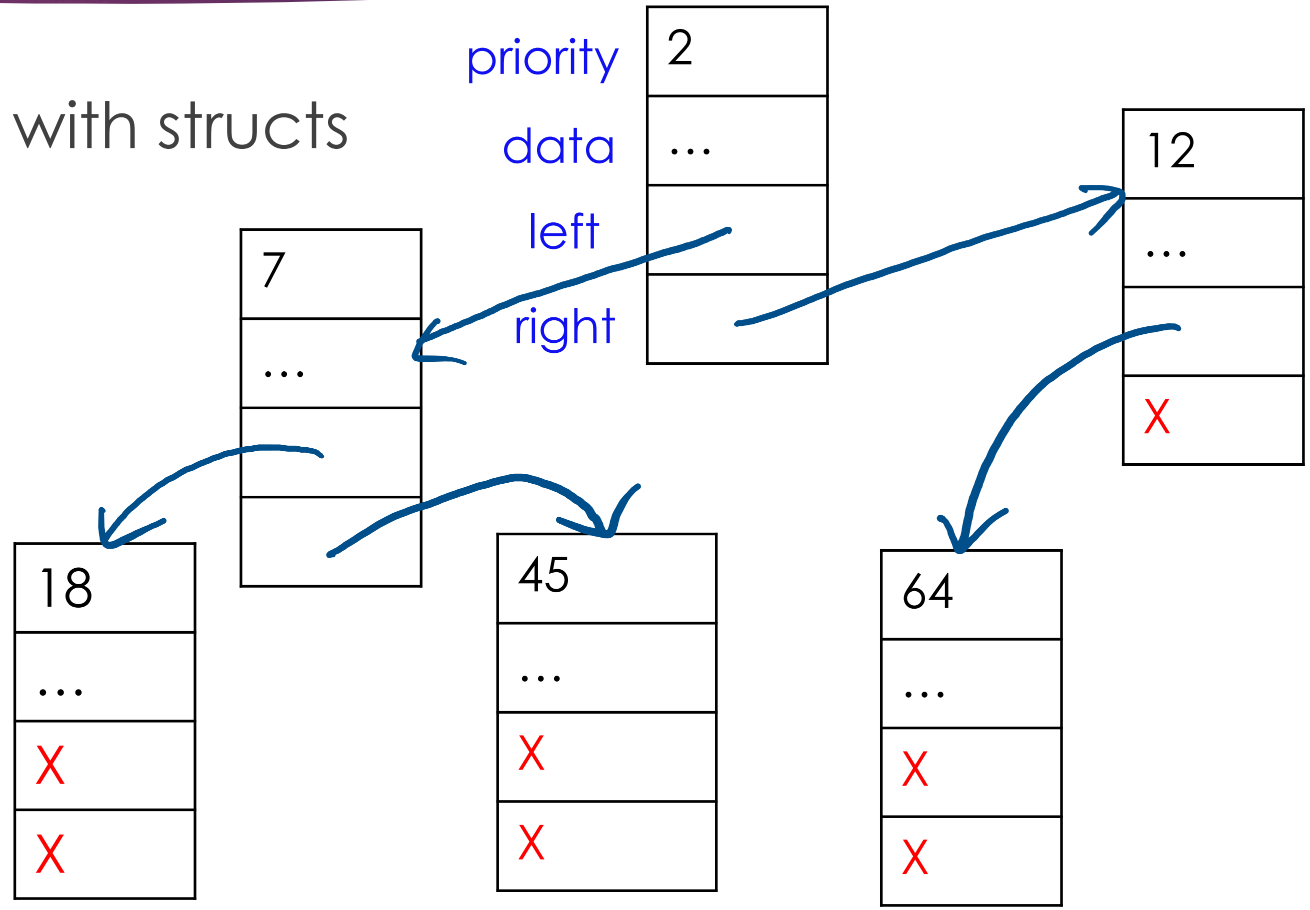
Representing a binary heap

- Binary heap is a concrete data structure, but we need to represent it with building blocks
  - Similar to how a ring buffer is a data structure but is represented with a simple array
  - Two layers of abstraction

- Ideas using structs and/or vectors?

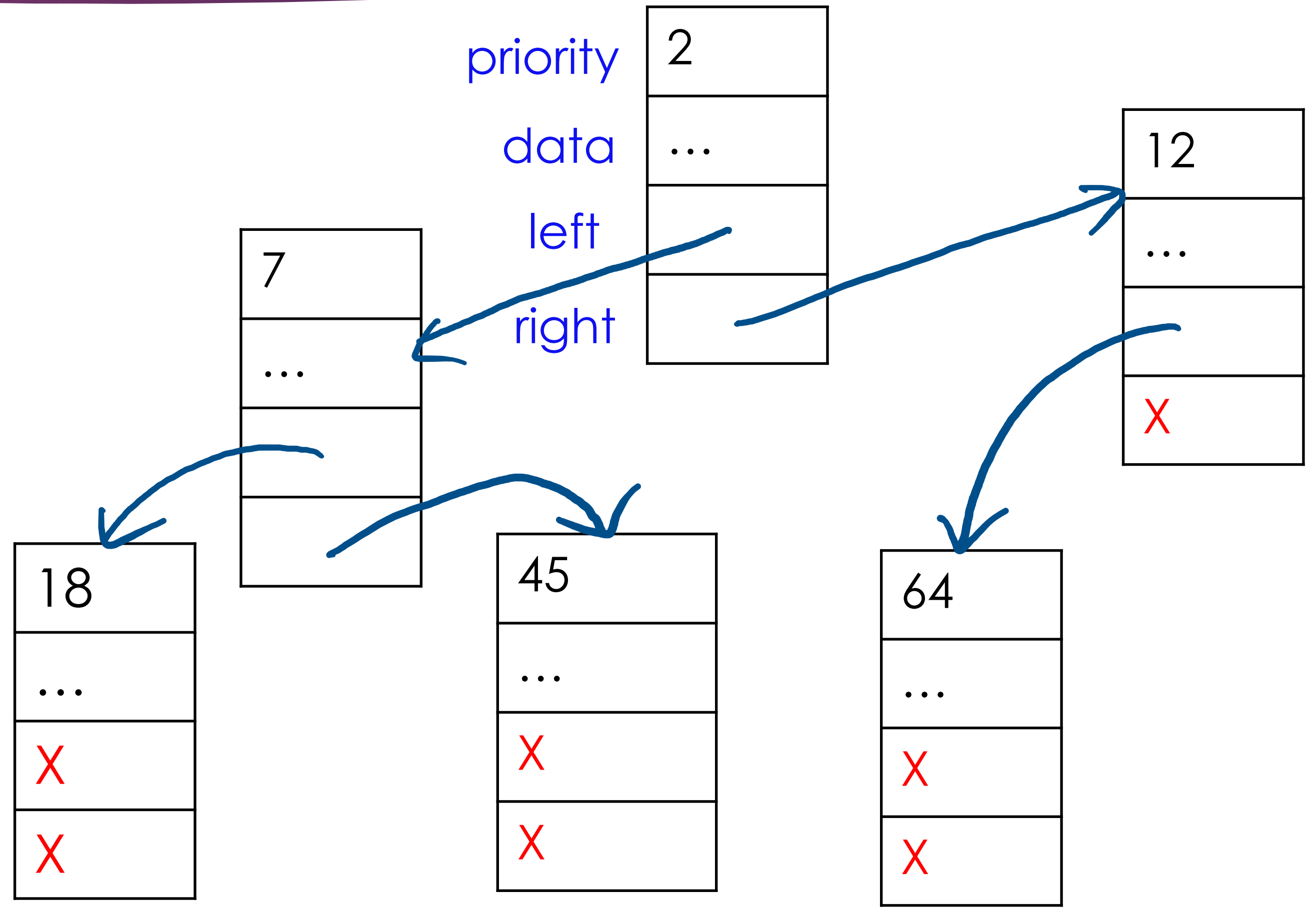
Binary heap with structs

- K-ary trees ***could*** be represented with structs
  - Intuitive representation



In-class exercise (5 minutes)

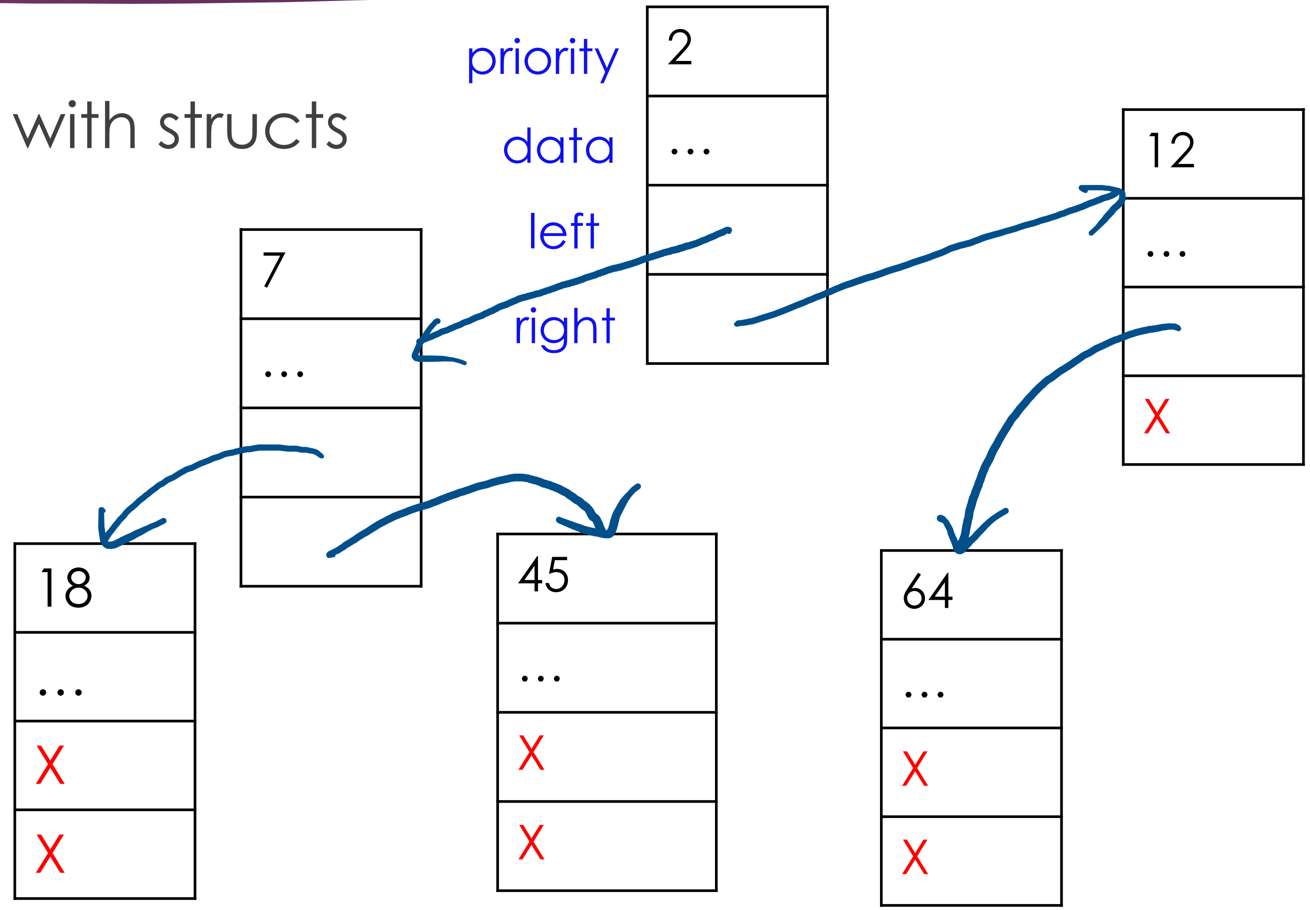
This representation is not sufficient to enable **bubbling up**.



1. What is the issue?
2. What specifically would you need to add to enable that operation?

Binary heap with structs

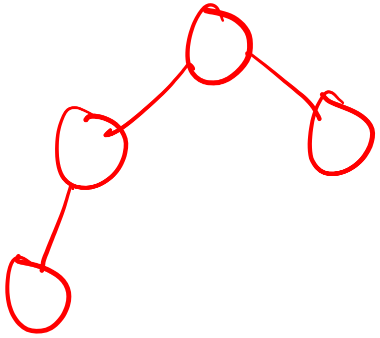
- K-ary trees ***could*** be represented with structs
  - Intuitive representation
- Issues with this approach?
  - What about bubbling up?
  - Need to have access to parent too
  - Becomes messy
- Let's not do that



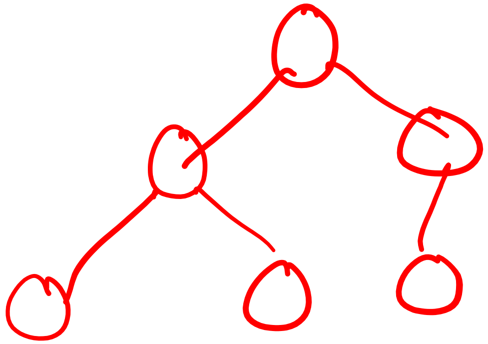
However... recall the shape invariant

- Heaps are always complete trees!
- For a given number of elements, there is only one possible shape!

Heap with 4 elements

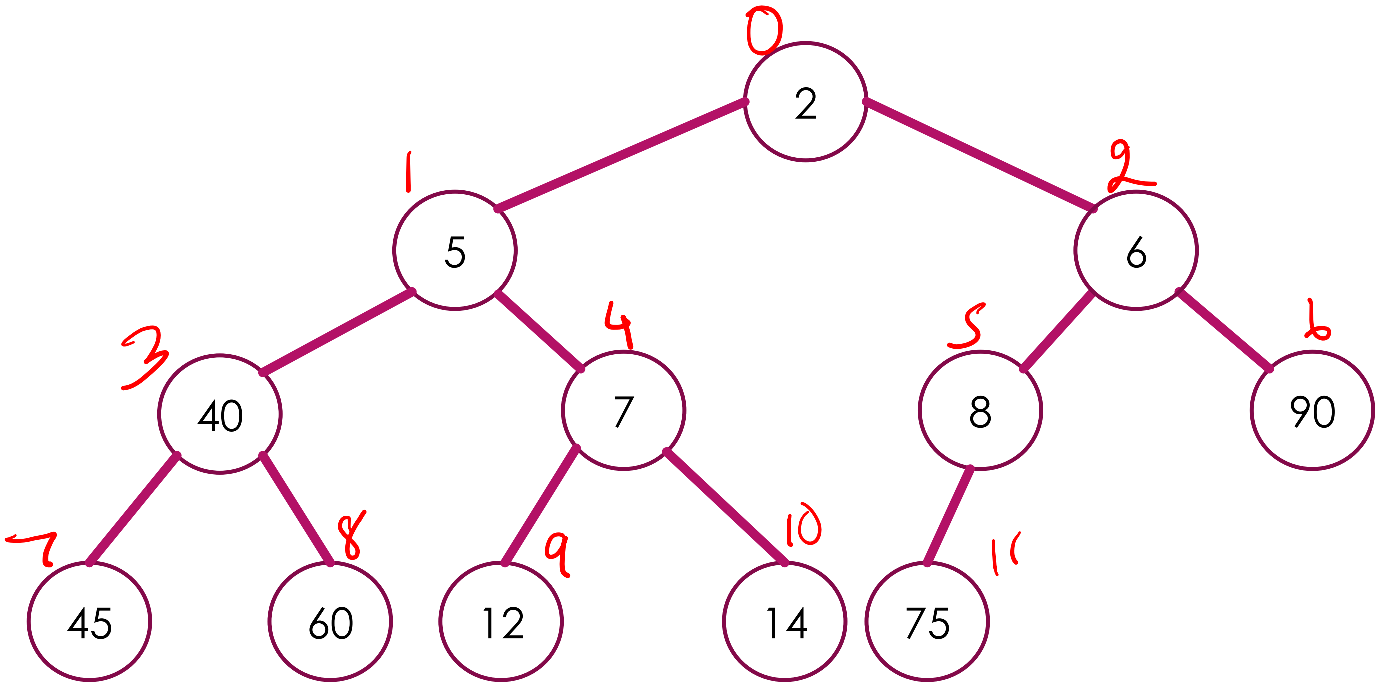


Heap with 6 elements



Representing binary heaps

- Let's label each element from top-bottom and left-right

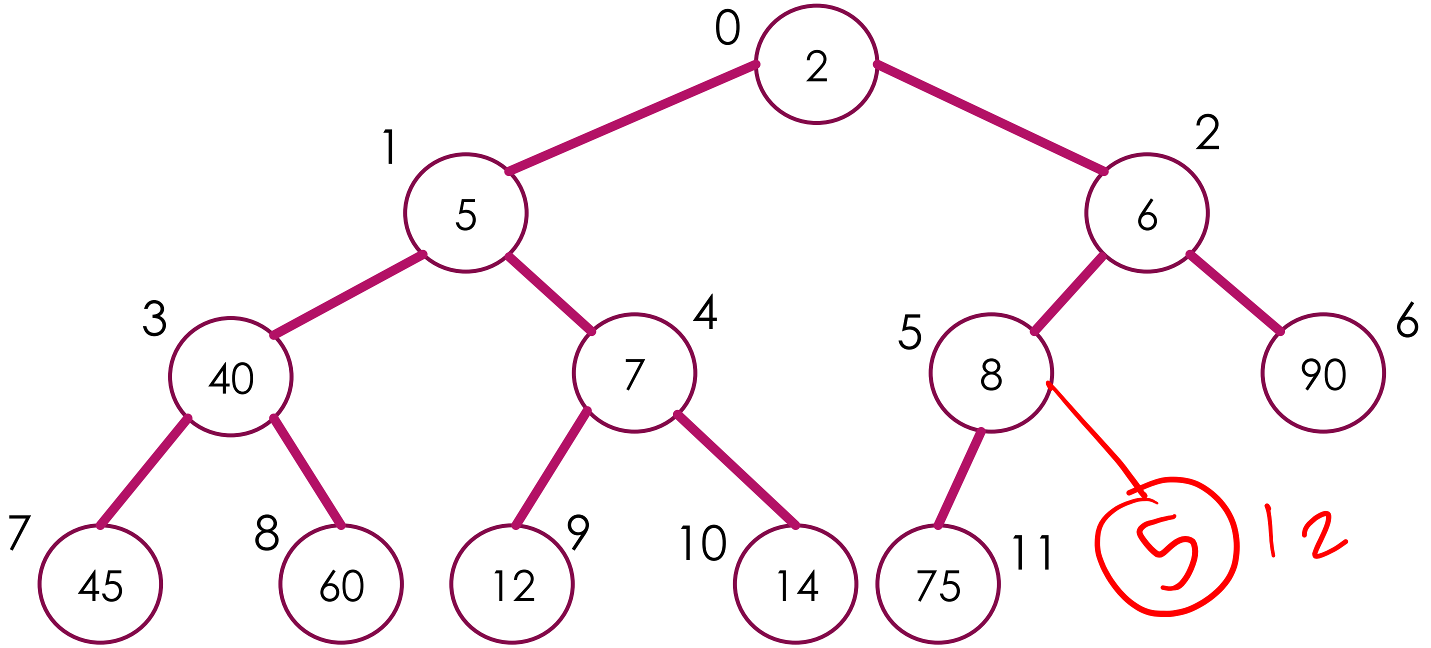


- This can fit perfectly in an array!

Representing binary heaps as arrays

- Store values on the heap in an array:
  - In order from top level to bottom level
  - In order from left to right

Representing binary heaps as arrays

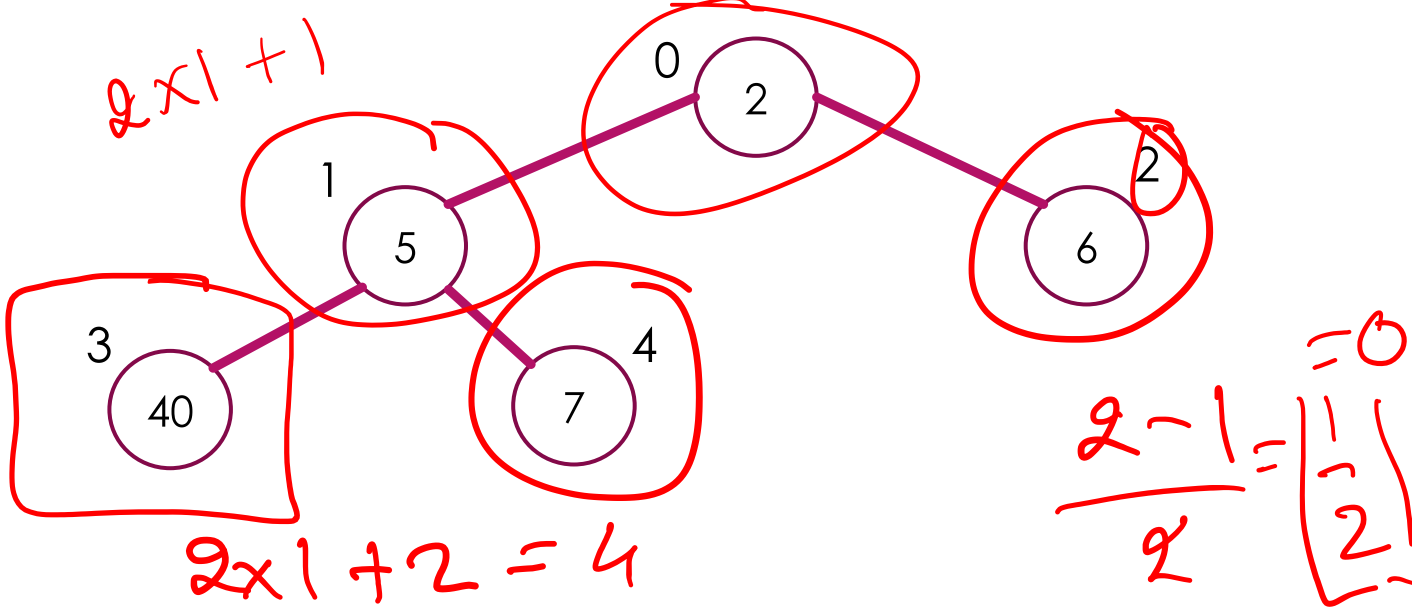


0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
2	5	6	40	7	8	90	45	60	12	14	75	5							

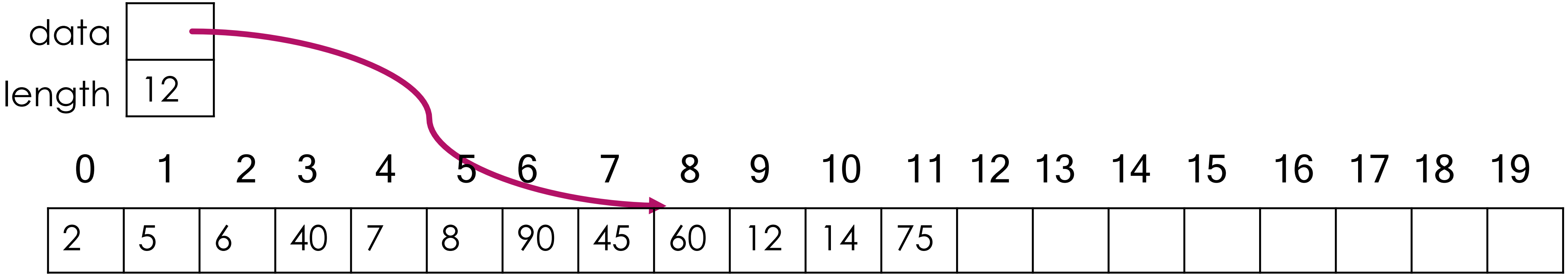
Findings parents and children

- Structure is *implicit*, so can't just follow arrows as there are no real arrows
  - Structure is well-defined though, so **array indices follow a pattern**

- Given a node at index i
  - Parent index = ⌊ (i-1)/2 ⌋
    - "floor" of quotient
  - Left child index = 2*i + 1
  - Right child index = 2*i + 2



Mapping operations to the array



- Where would you insert?
  - At index `length`
- Where would you remove?
  - Remove what's at index `length-1` and put it at index 0
- Update `length` in both cases

Next week

- Data Design
  - We'll ADTs and data structures to work on a problem from start to end
  - Come prepared to participate!
  - Preview for your final project

- Priority queues will make a comeback after next week