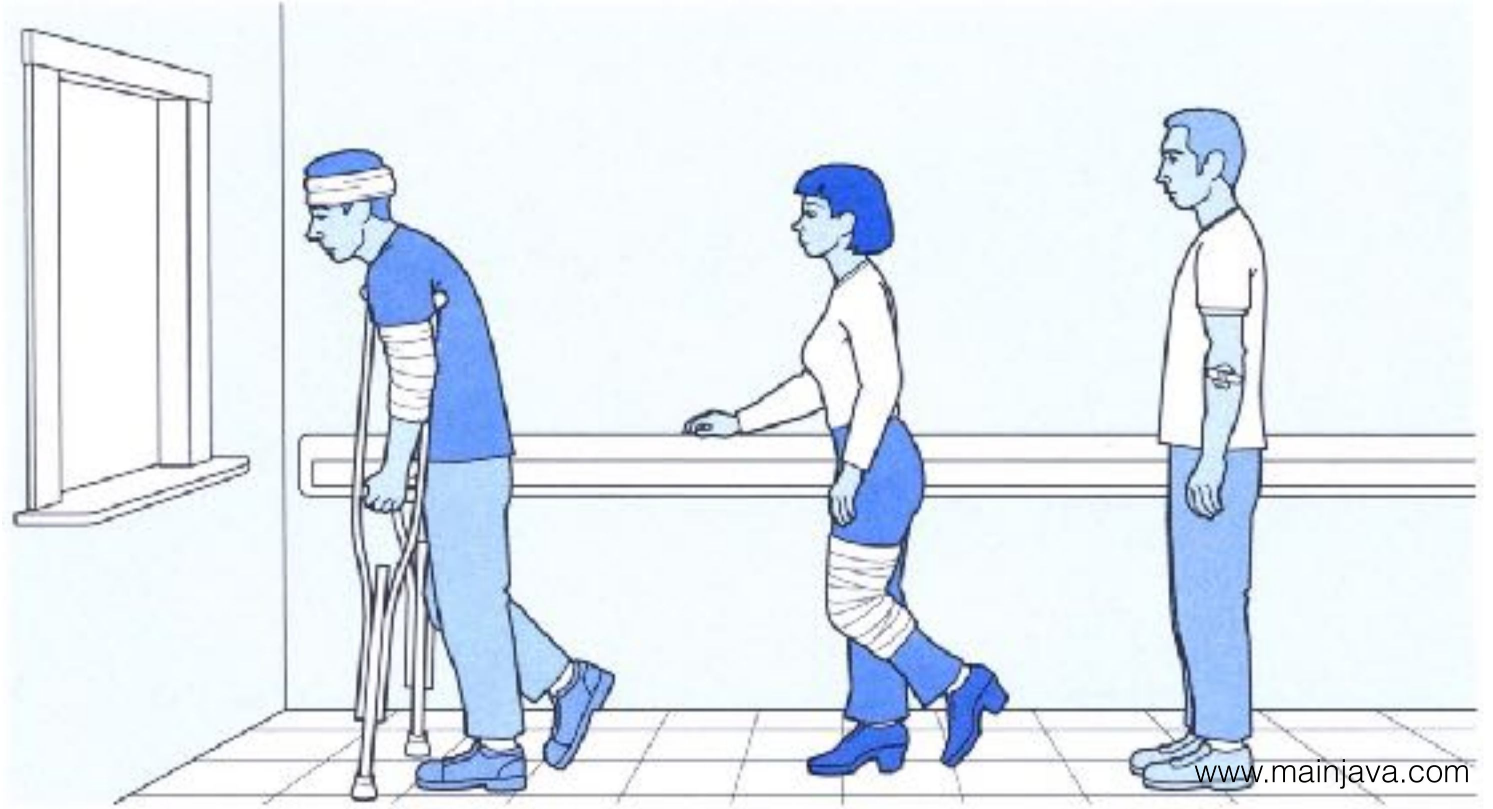
www.heapson.com



www.mainjava.com

CSCI 241

Lecture 14
Heaps and the Priority Queue ADT

Announcements

- Many people are losing style points on A1.

- Please review style guidelines in the syllabus and on the assignment writeup.
    - A 300-line main() method is not good style.
      break it into smaller methods
    - Copy/pasting code to print an array every time you need to do it is not good style.
      write a helper method, call it when needed
    - Inconsistent indentation is not good style.
      there are tools that will fix this for you if you can't manage it
    - `while(++k>j)A[i++]=B[k];`
      might make you feel clever, but it is not good style.
      conciseness is only good in the service of **clarity**

Goals

- Understand the purpose and interface of the Priority Queue ADT.

- Know the definition and properties of a heap.

- Understand how to implement a Priority Queue using a heap

- Be prepared to implement heap insertion and removal.

Preliminaries - Interfaces

Java has a thing called an **interface**.

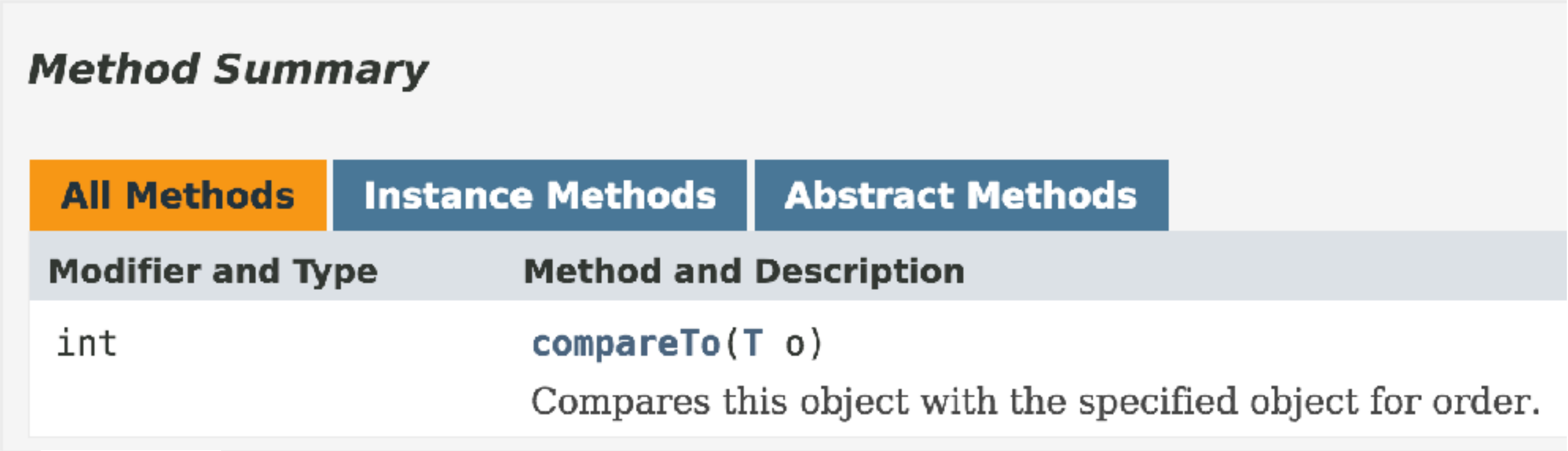
It’s like a class, but doesn’t have method bodies. It only exists so other classes can **implement** it.

public interface Set

Specifies public method names, specs, parameters, return values, etc.

Preliminaries - `Comparable`

The **`Comparable`** interface has one method:



Method Summary

Modifier and Type	Method and Description
`int`	`compareTo(T o)` Compares this object with the specified object for order.

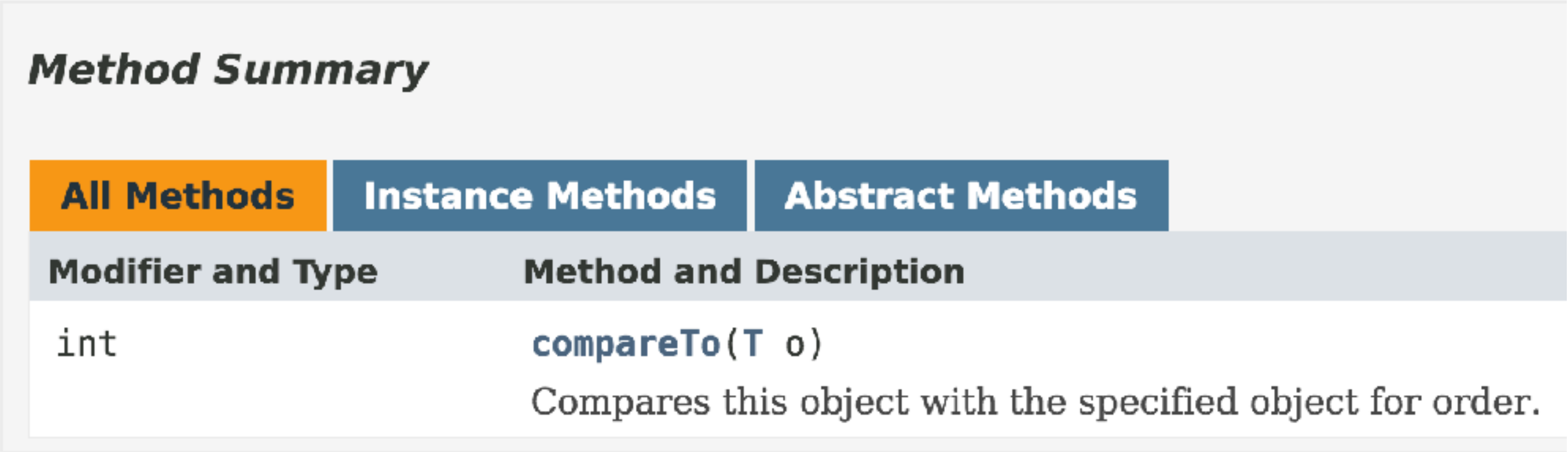
Returns:

`a negative integer if` **`this`** `< o`
`zero if` **`this`** `is equal to o`
`a positive integer if` **`this`** `is > o.`

From A2: you can call w.compareTo(node.word) because `String` <u>implements</u> `Comparable`.

Preliminaries - `Comparable`

The **`Comparable`** interface has one method:



Method Summary

Modifier and Type	Method and Description
`int`	`compareTo(T o)` Compares this object with the specified object for order.

If you can compare items, you can sort them!

They have a well-defined **ordering**.

Priority Queue

Like a Queue, but:

- items are **Comparable**
- removal (called **poll()**) returns item with the "highest priority"
    - we define "highest priority" as "smallest" element according to **`compareTo()`**
    - if multiple "smallest" elements are equal (**`compareTo`** returns 0), we can remove either.

```
interface PriorityQueue {
 boolean add(Object e); // insert e
 Object peek(); // return min element
 Object poll(); // remove/return min element
 void clear();
 boolean contains(Object e);
 boolean remove(Object e);
 int size();
 Iterator iterator();
}
```

Implement Priority Queue using LinkedList

An unsorted list:

- **add()** - new element goes at front of list - O(   )
- **poll()** - search the list, remove smallest - O(   )
- **peek()** - search the list, return smallest - O(   )

A sorted list:

- **add()** - insert item in sorted position - O(   )
- **poll()** - min element is at front - O(   )
- **peek()** - min element is at front - O(   )

Implement Priority Queue using LinkedList

An unsorted list:

- **add()** - new element goes at front of list - O(1)
- **poll()** - search the list, remove smallest - O(n)
- **peek()** - search the list, return smallest - O(n)

A sorted list:

- **add()** - insert item in sorted position - O(n)
- **poll()** - min element is at front - O(1)
- **peek()** - min element is at front - O(1)

Question to ponder:

What would be the runtime of add, peek, and poll if you implement a Priority Queue using a BST?

What about an AVL tree?

Priority Queue: heap implementation

- A heap is a **concrete** data structure that can be used to **implement** a Priority Queue

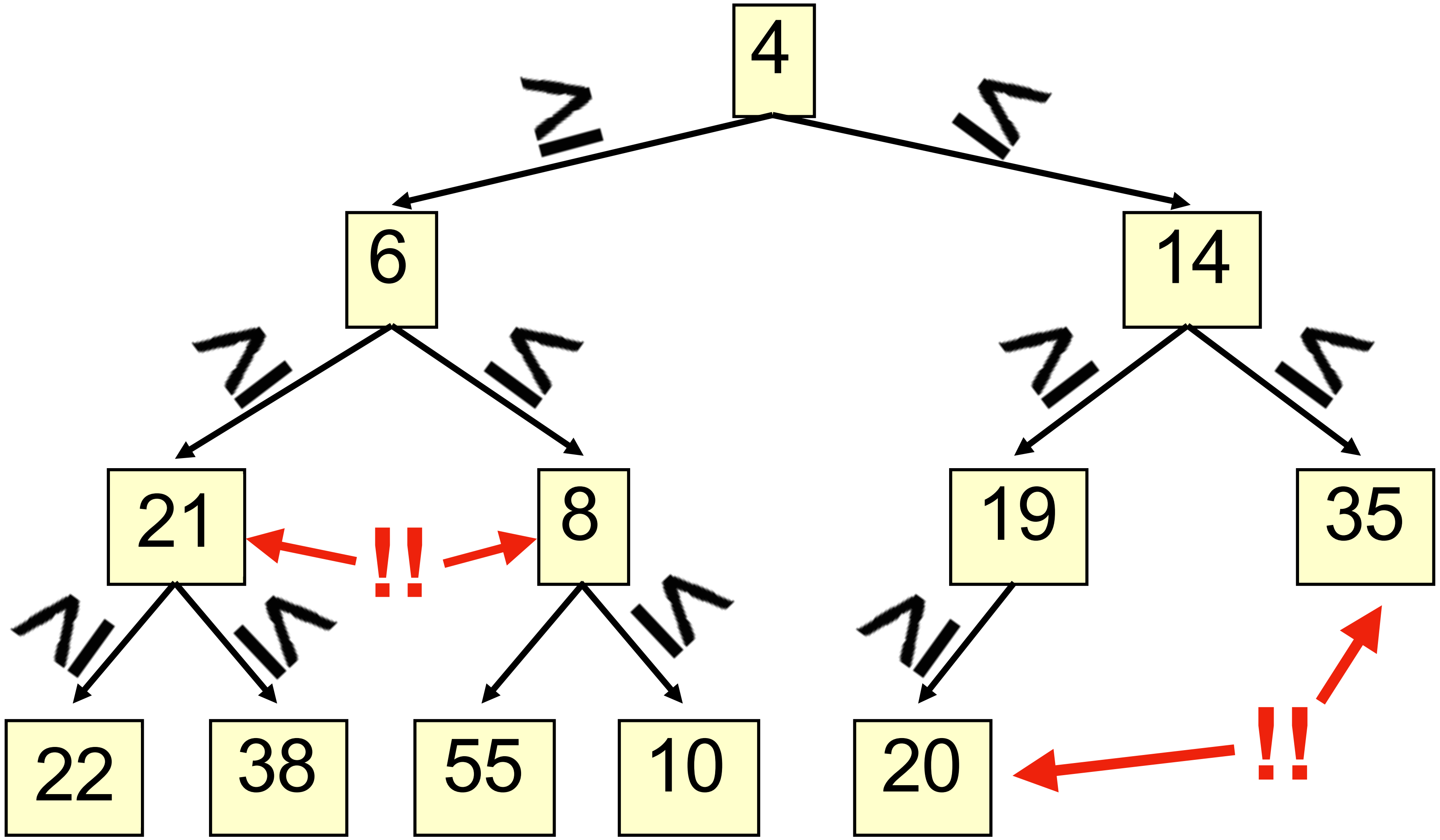
- Better runtime complexity than either list implementation:
  - **peek()** is O(1)
  - **poll()** is O(log n)
  - **add()** is O(log n)

- Not to be confused with *heap memory*, where the Java virtual machine allocates space for objects – different usage of the word heap.

A heap is a special binary tree with two additional properties.

A heap is a special binary tree.

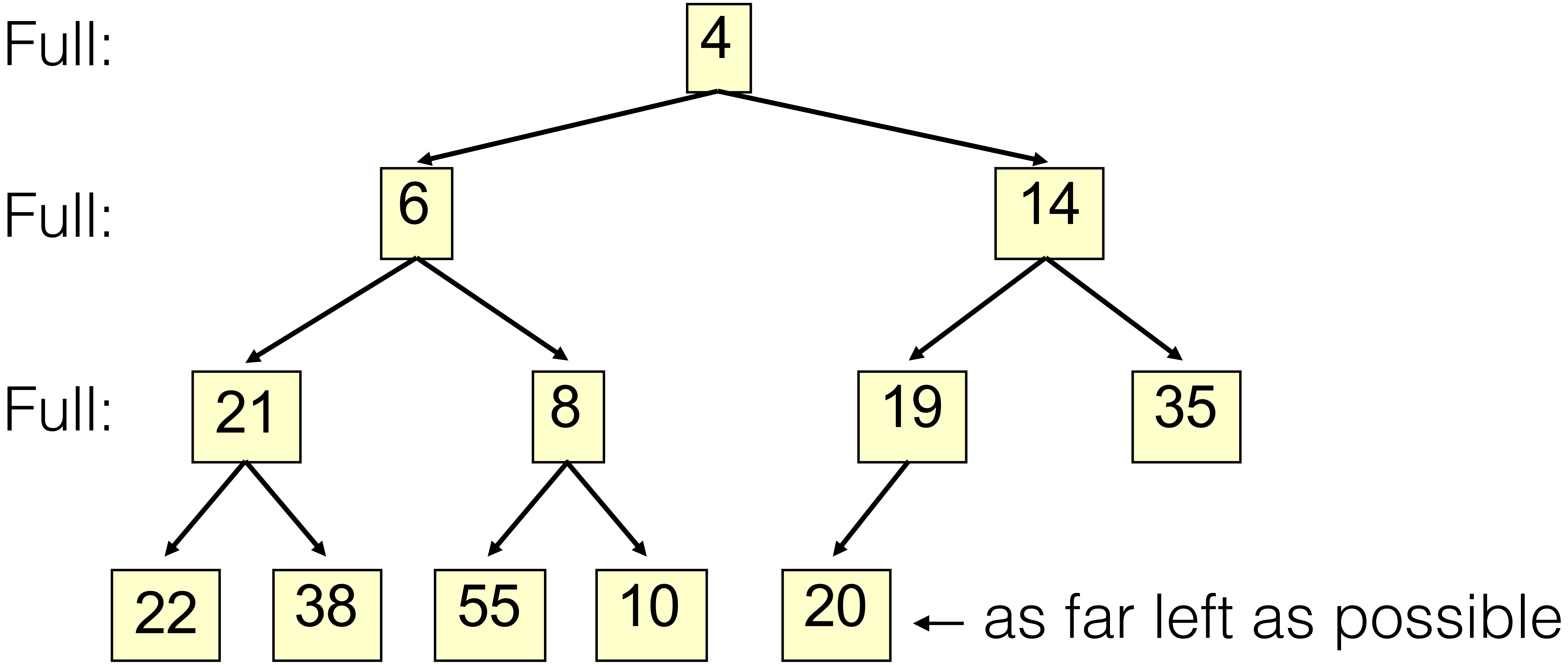
1. **Heap Order Invariant:**
Each element >= its parent.



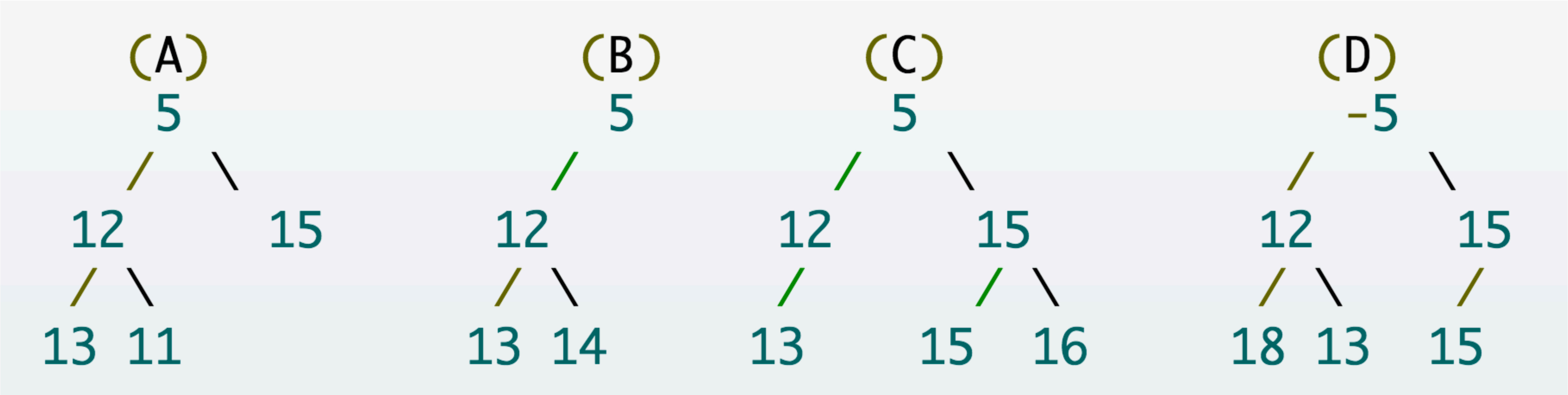
A heap is a special binary tree.

2. **Complete:** no holes!

- All levels except the last are **full**.
- Nodes in last level are as far left as possible.



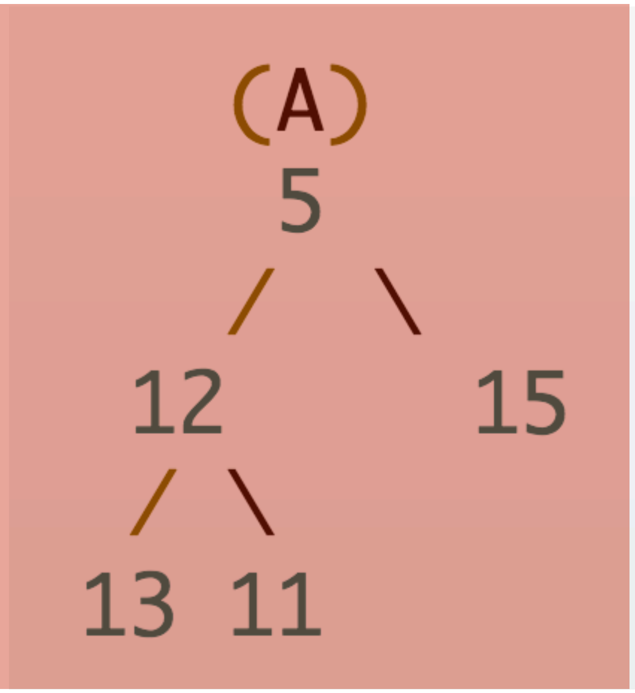
Heap it real.



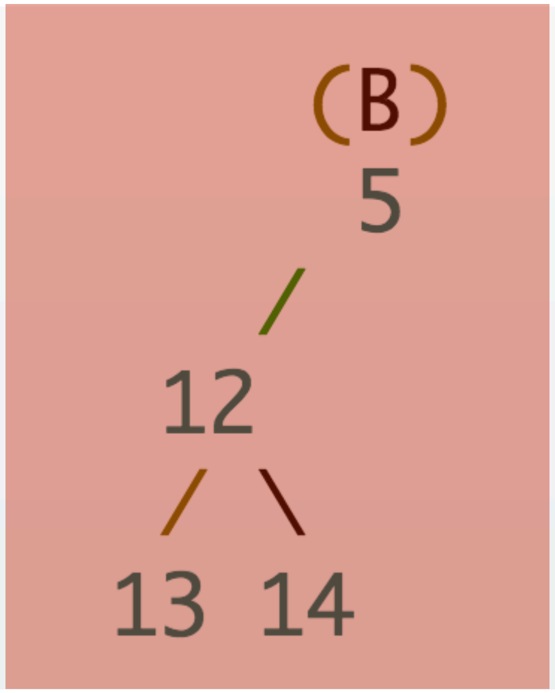
```
   (A)             (B)       (C)             (D)
    5               5         5               -5
   /  \           /         /   \           /    \
 12     15      12        12     15       12     15
 / \            / \       /      / \      / \    /
13 11          13 14     13    15  16    18 13  15
```

Which of these is a valid heap?

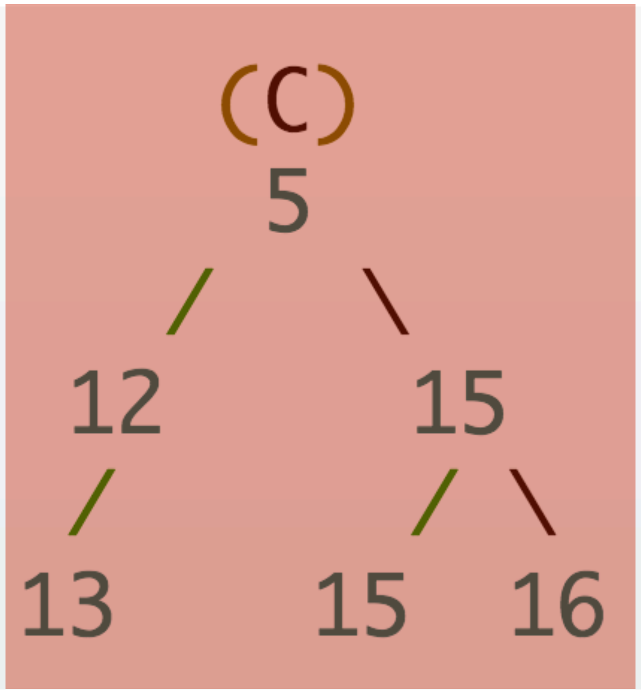
Heap it real.



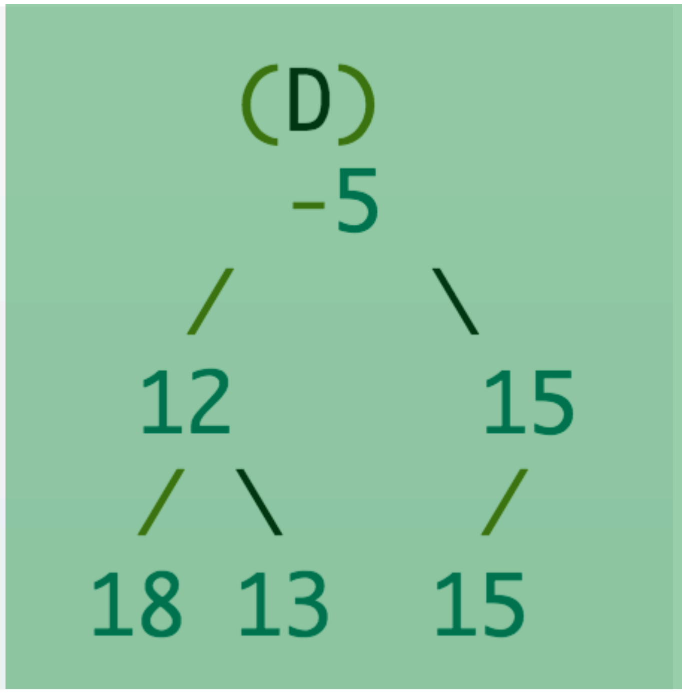
```
   (A)
    5
   /  \
 12     15
 / \
13 11
```



```
    (B)
     5
   /
 12
 / \
13 14
```



```
    (C)
     5
   /   \
 12     15
 /      / \
13    15  16
```



```
   (D)
    -5
  /    \
 12     15
 / \    /
18 13  15
```

Which of these is a valid heap?

11 is < its parent

level 1 is not full
(5 needs a right child)

leaves are not as
far left as possible

heap!

Heap operations

```
interface PriorityQueue {
 boolean add(Object e); // insert e
 Object peek(); // return min element
 Object poll(); // remove/return min element
 void clear();
 boolean contains(Object e);
 boolean remove(Object e);
 int size();
 Iterator iterator();
}
```