CSCI 241
Scott Wehrwein

Heaps:
Definition, add, Storage
Goals

Know the definition and properties of a heap.

Understand how heaps are stored in practice.

Be prepared to implement add and execute it on paper.
Heap implements PriorityQueue

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

Better runtime complexity than list, BST, AVL 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 $\geq$ 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.

Full:

```
     4
    /\  \
   6  14
  /   /   \
 21  8   19
 /   /     \
22 38 55 10
```

```
     4
    /\  \
   6  14
  /   /   \
 21  8   19
 /   /     \
22 38 55 10
```

```
     4
    /\  \
   6  14
  /   /   \
 21  8   19
 /   /     \
22 38 55 10
```

```
     4
    /\  \
   6  14
  /   /   \
 21  8   19
 /   /     \
22 38 55 10
```
Heap Operations

interface PriorityQueue<V v, P p> {
    // insert value v with priority p
    void add(V v, P p);

    // return value with min priority
    V peek();

    // remove/return value with min priority
    V poll();

    // more methods...
}
void add(V v, P p);

Algorithm:
• Add v in the wrong place
• While v is in the wrong place
  • move v towards the right place
void add(V v, P p);
void add(V v, P p);
void add(V v, P p);
void add(V v, P p);
void add(V v, P p);
void add(V v, P p);

Algorithm:
• Add v in the wrong place \textit{(the leftmost empty leaf)}
• While v is in the wrong place \textit{(its p is less than its parent's)}
  • move v towards the right place \textit{(swap with parent)}

The heap invariant is maintained!
public class HeapNode {
    private int value;
    private HeapNode left;
    private HeapNode right;
    ...
}

public class Heap {
    HeapNode root;
    ...
}
Implementing Heaps

public class Heap
{
    private int value;
    private Heap left;
    private Heap right;
    ...
}
A heap is a special binary tree.

2. **Complete**: no holes!

```
  4
 /   \
6     14
 /  \\   /\
21   8 19 35
 / \
22 38 55 10 20
```

← as far left as possible
Numbering Nodes

Level-order traversal:

2. Complete: no holes!
node $k$’s parent is
node $k$’s children are nodes and
Numbering Nodes

node $k$'s parent is $(k - 1)/2$
node $k$'s children are nodes and
Numbering Nodes

node $k$’s parent is $(k - 1)/2$
node $k$’s children are nodes $2k + 1$ and $2k + 2$
Implementing Heaps

```java
public class Heap {
    private Entry[] heap;
    private int size;
```

...
Implicit Tree Structure

2. Complete: no holes!

```
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
4 6 14 21 8 19 35 22 38 55 10 20
```

```
Implicit Tree Structure

2. Complete: no holes!

```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```