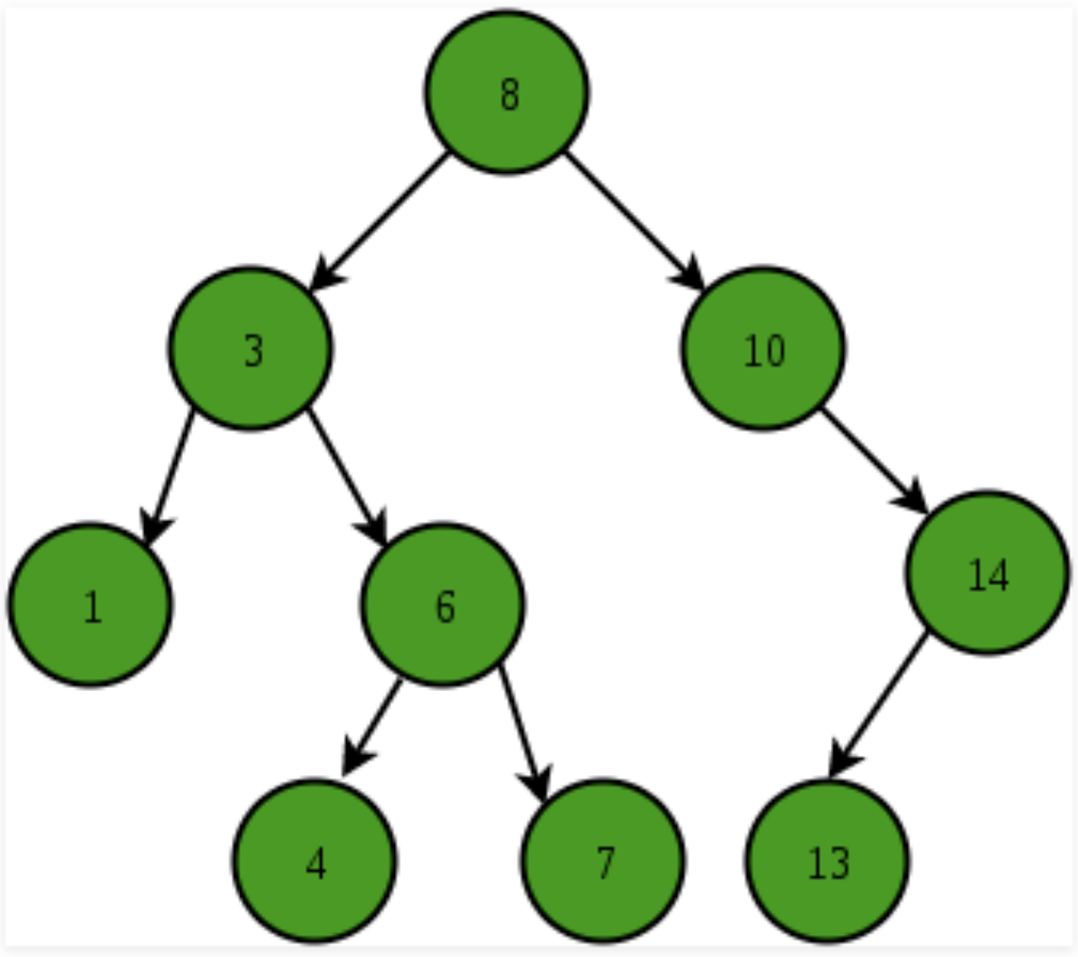
Trees — part 5

The final chapter

CS 206
April 2

Traversals of binary tree



- **Depth First**
  - **inorder**
    - **1, 3, 4, 6, 7, 8, 10, 13, 14**
  - **Preorder**
    - **8, 3, 1, 6, 4, 7, 10, 14, 13**
  - **Postorder**

- **Breadth First:**
- **8, 3, 10, 1, 6, 14, 4, 7, 13**

Horrible Breadth-First Traverser

```
public void badBF() {
    int height = height();
    for (int i=0; i<(height+1); i++) {
        printAtHeight(root, i, 0);
    }
    System.out.println();
}
```

```
private void printAtHeight(Node tp, int h2Print, int cd) {
    if (tp==null) return;
    if (h2Print==cd) {
       System.out.print(tp + " “);
       return;
    }
    if (h2Print<cd) return;
    printAtHeight(tp.left, h2Print, cd+1);
    printAtHeight(tp.right, h2Print, cd+1);
}
```

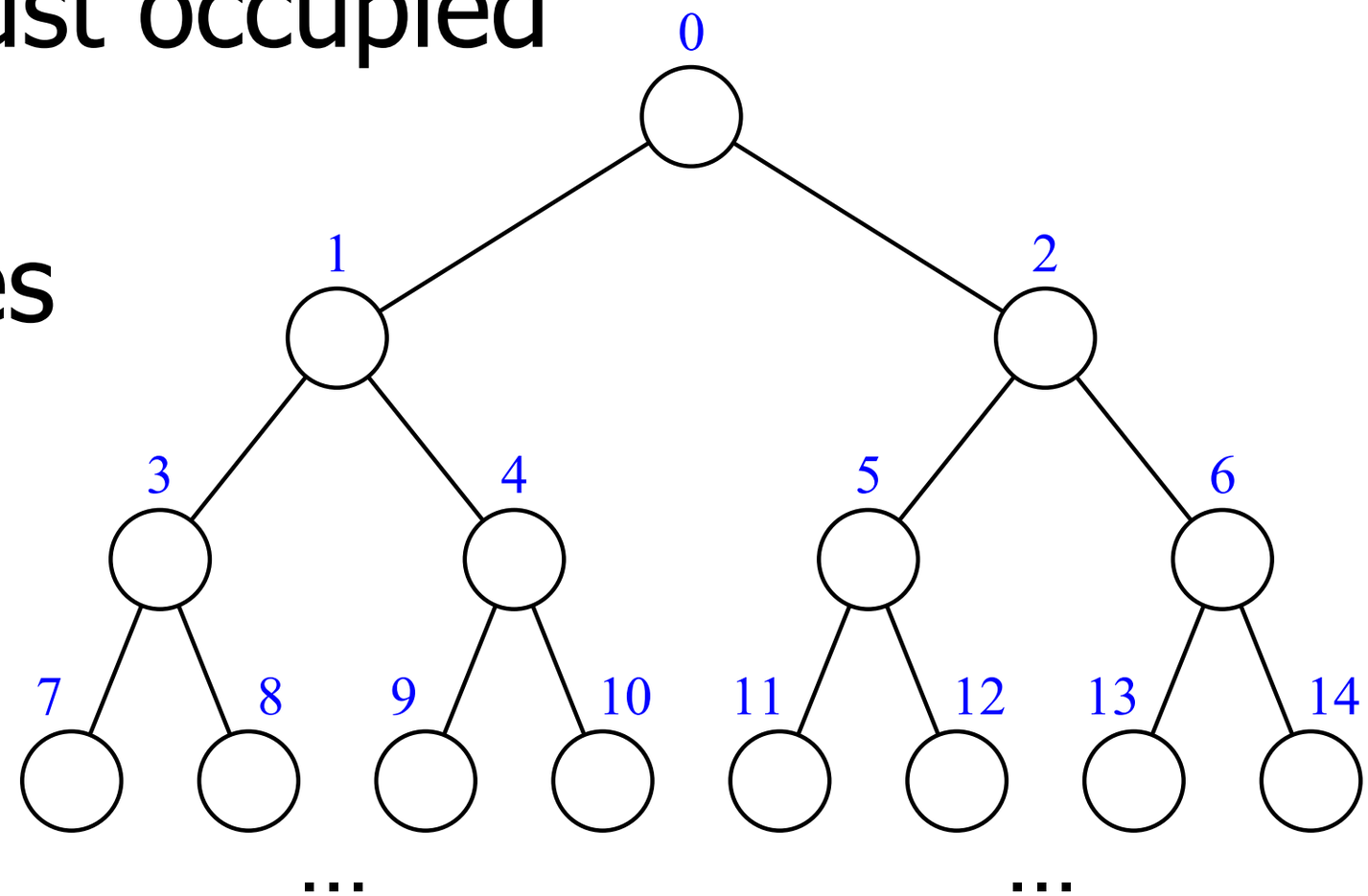
Good Breadth-First Traversal

Using Java LinkedList class as a Queue

```
public void goodBF() {
    if (root==null) return;
    LinkedList<Node> bfq = new LinkedList<>();
    bfq.add(root);
    while (!bfq.isEmpty()) {
        Node n = bfq.poll();
        System.out.print(n + " ");
        if (n.left!=null) bfq.add(n.left);
        if (n.right!=null) bfq.add(n.right);
    }
    System.out.println();
}
```

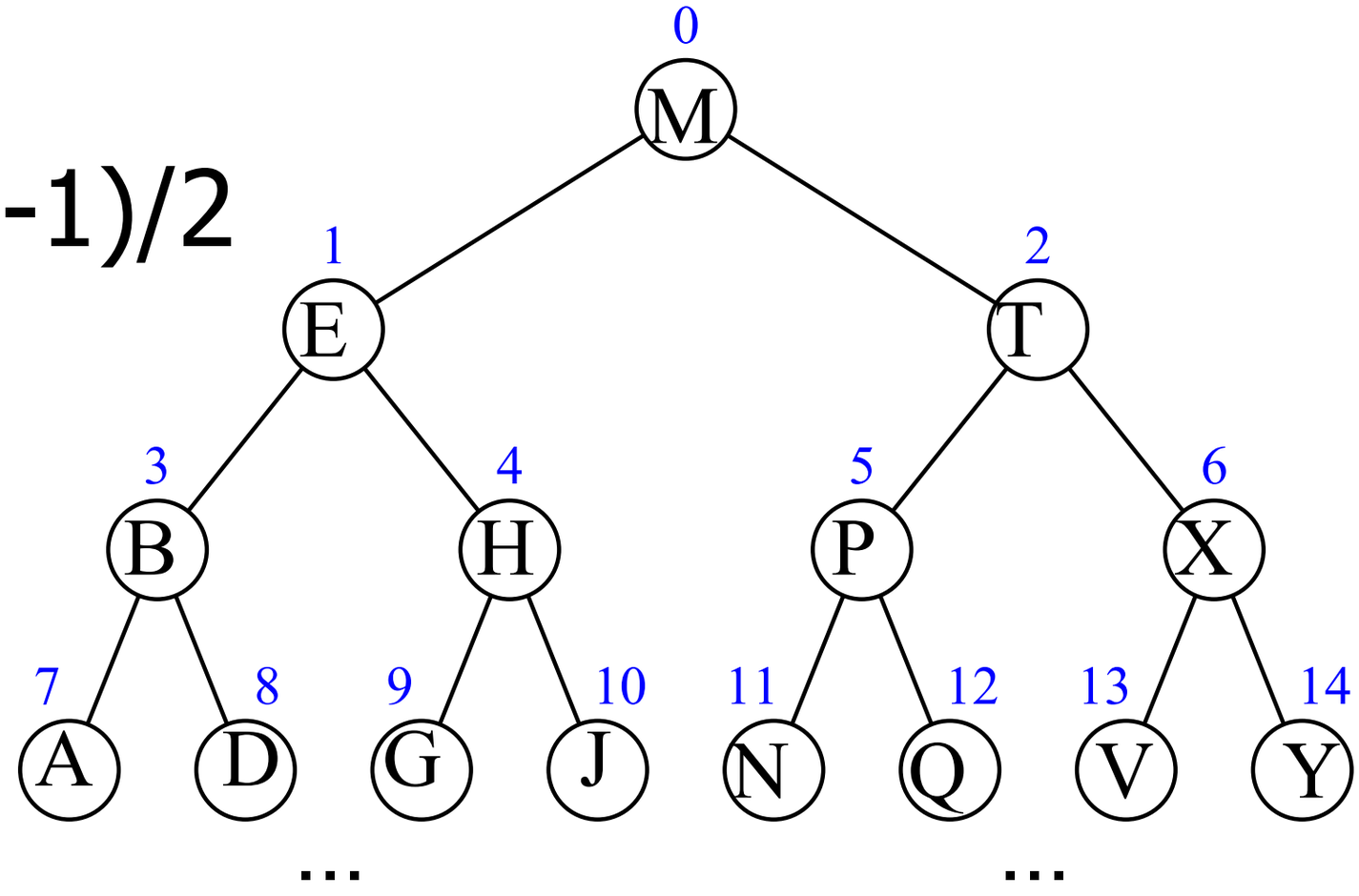
Array-based Binary Tree

- Number nodes level-by-level, left-to-right
- Numbering is based on all positions, not just occupied positions
- Numbering gives location in array



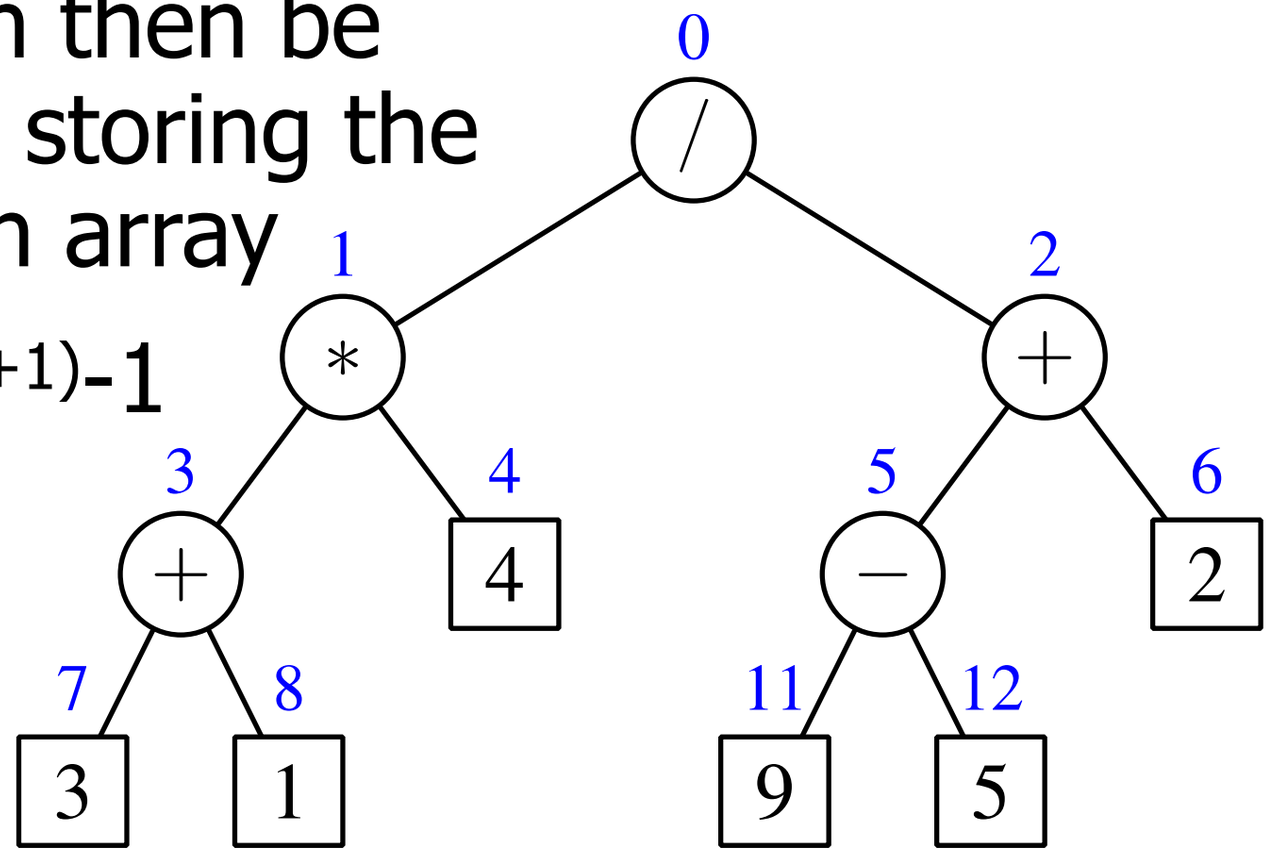
Parent/Child numbering

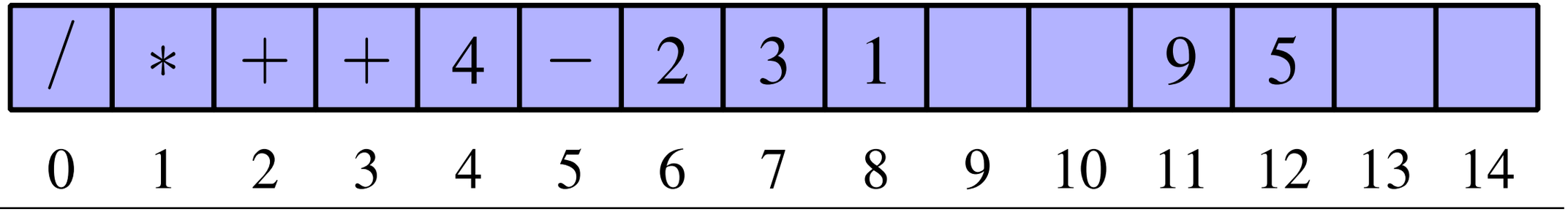
- $f(root) = 0$
- f(l) = 2f(p) + 1
- f(r) = 2f(p) + 2
- f(p) = floor(f(c)-1)/2



Array-based Binary Tree

- The numbering can then be used as indices for storing the nodes directly in an array
- Size of array = $2^{(h+1)}$-1





height()

```
public int height() {
    return iHeight(0);
}
private int iHeight(int loc) {
    if (loc>backArray.length) return 0;
    if (backArray[loc] == null) return 0;
    int hl = iHeight(loc*2+1);
    int hr = iHeight(loc*2+2);
    return (hr>hl?hr:hl)+1;
}
```

height (alternative)

```
public int altHeight() {
    for (int i=backArray.length-1; i>=0; i--) {
        if (backArray[i]!=null) {
            int j=0;
            while(i>0) {
                j++;
                i/=2;
            }
            return j;
        }
    }
    return 0;
}
```

Performance of Trees

	Complete Tree	Worst Tree
search		
insert		
remove		

Shallow & deep copy

```
public class DeepShallow {

    public void shde() {
    ArrayList<Node> arrOrig = new ArrayList<>();
    for (int i=0; i<10; i++) arrOrig.add(new Node(i));

    // Pointer Copy
    ArrayList arrpc = arrOrig;

    //Shallow Copy // objects in the array list are the same
    ArrayList<Node> arrShallow = new ArrayList<>();
    for (Node n: arrOrig)
        arrShallow.add(n);
    // Collections.copy(arrShallow, arrOrig);

    //Deep Copy // objects have same contents, but are not the same
    ArrayList<Node> arrDeep = new ArrayList<>();
    for (Node n : arrOrig)
        arrDeep.add(new Node(n.payload));
    }
```

Priority Queue

- A queue that maintains order of elements according to some priority
  - Removal order, not general order
    - the rest may or may not be sorted

- Priority Queues in real world
  - Clubs with VIP entrances
  - Homework
  - Medical triage

Key

- Priority queues are ordered by some key, which may be:
  - derived from the data element
    - one field
    - combination of fields
  - independent of data element
    - for example: insertion time
  - Part (or all) of the data element
- best practice is make keys implement `Comparable` relation between keys using `compareTo`

Key-Value Pair

- Typically think of PQ as containing a pair
  - (Key, Value)
    - Key defines priority
    - Value is data the objects store
- KV pairs are frequently used
- Ideally keys are unique
  - how to handle duplicate keys?
- Ideally keys have a natural ordering.
  - Using `compareTo` allows arbitrary comparisons
- Values need not be numerical or unique

PriorityQueue Interface

```
public interface QueueInterface<Q> {
    boolean isEmpty();
    int size();
    boolean offer(Q q);
    Q poll();
    Q peek();’
    // also add, remove, element
}
```

```
public interface PriorityQInterface<K extends Comparable<K>, V> {
    boolean isEmpty();
    int size();
    boolean offer(K key, V value);
    V poll();
    V peek();
}
```

Mini-Homework

For each of the following three lists of integers, build a binary tree.

For each binary tree show its representation as a tree and how it would be stored in an array-based binary tree (see slide 7). Be sure that array indices are clear, to you and me. Assume that the backing array has 31 spaces. If item cannot be put into the array, clearly indicate that.

```
100, 200, 50, 25, 75, 60, 12, 150, 175, 187

100, 200, 400, 800, 1600, 3200, 50, 25, 12, 6, 150, 225, 275, 350

78, 165, 29, 54, 46, 117, 180, 76, 58, 56, 52, 45, 172, 194, 86
```