

Topic 25

Heaps

"You think you know when you can **learn**,
are more sure when you can **write**,
even more when you can **teach**,
but certain when you can **program**."

- Alan Perlis



Priority Queue

- ▶ Recall priority queue
 - elements enqueued based on priority
 - dequeue removes the highest priority item
- ▶ Options?
 - List? Binary Search Tree? **Clicker 1**

Array List enqueue

BST enqueue

A. $O(N)$

$O(1)$

B. $O(N)$

$O(\log N)$

C. $O(N)$

$O(N)$

D. $O(\log N)$

$O(\log N)$

E. $O(1)$

$O(\log N)$

Another Option

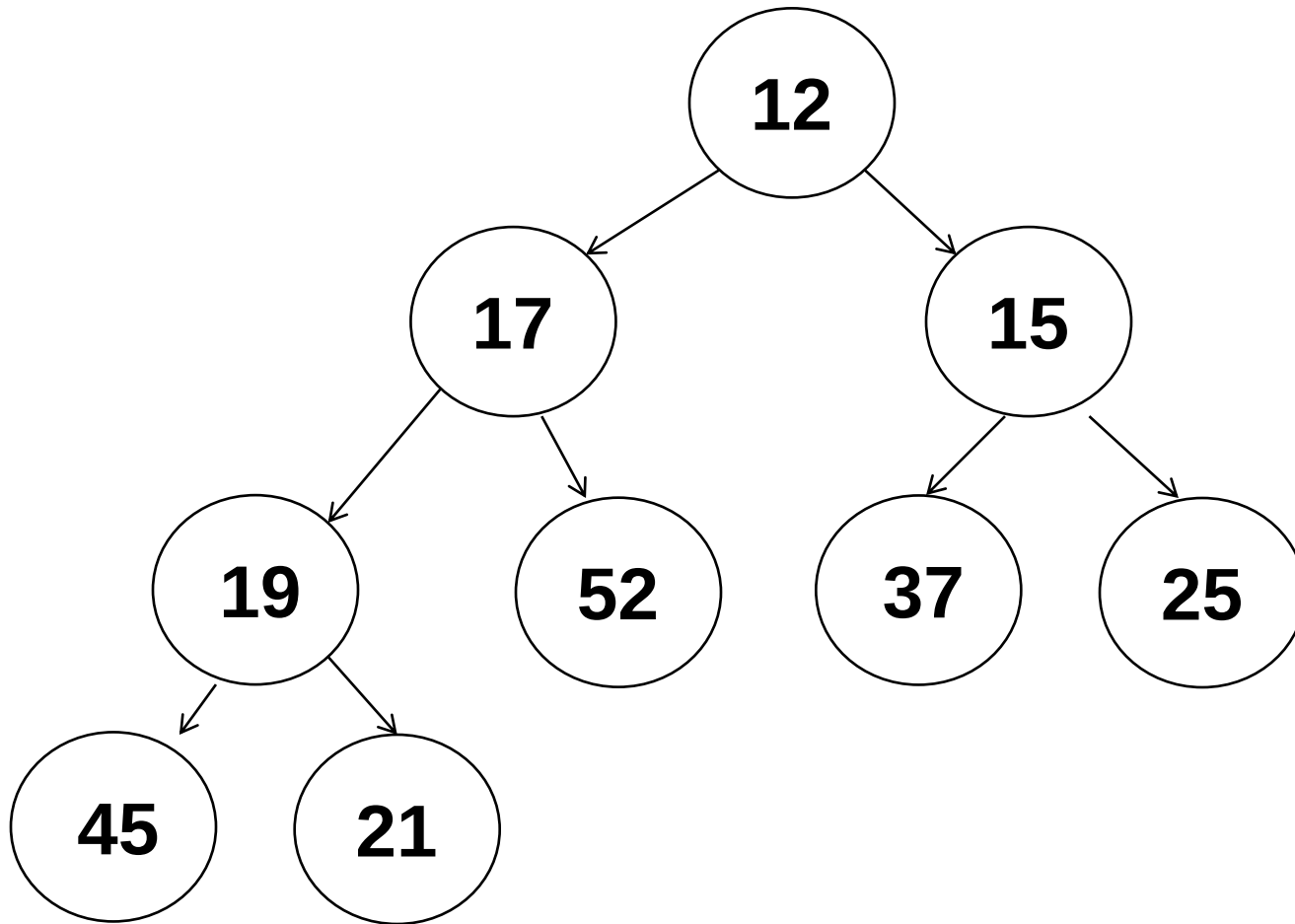
- ▶ The *heap* data structure
 - not to be confused with the runtime heap (portion of memory for dynamically allocated variables)
- ▶ Typically a complete binary tree (variations with more than 2 children possible)
 - all levels have maximum number of nodes except deepest where nodes are filled in from left to right
- ▶ Maintains the *heap order property*
 - in a min heap the value in the root of any subtree is less than or equal to all other values in the subtree

Clicker 2

► In a max heap with no duplicates where is the largest value?

- A. the root of the tree
- B. in the left-most node
- C. in the right-most node
- D. a node in the lowest level
- E. none of these

Example Min Heap

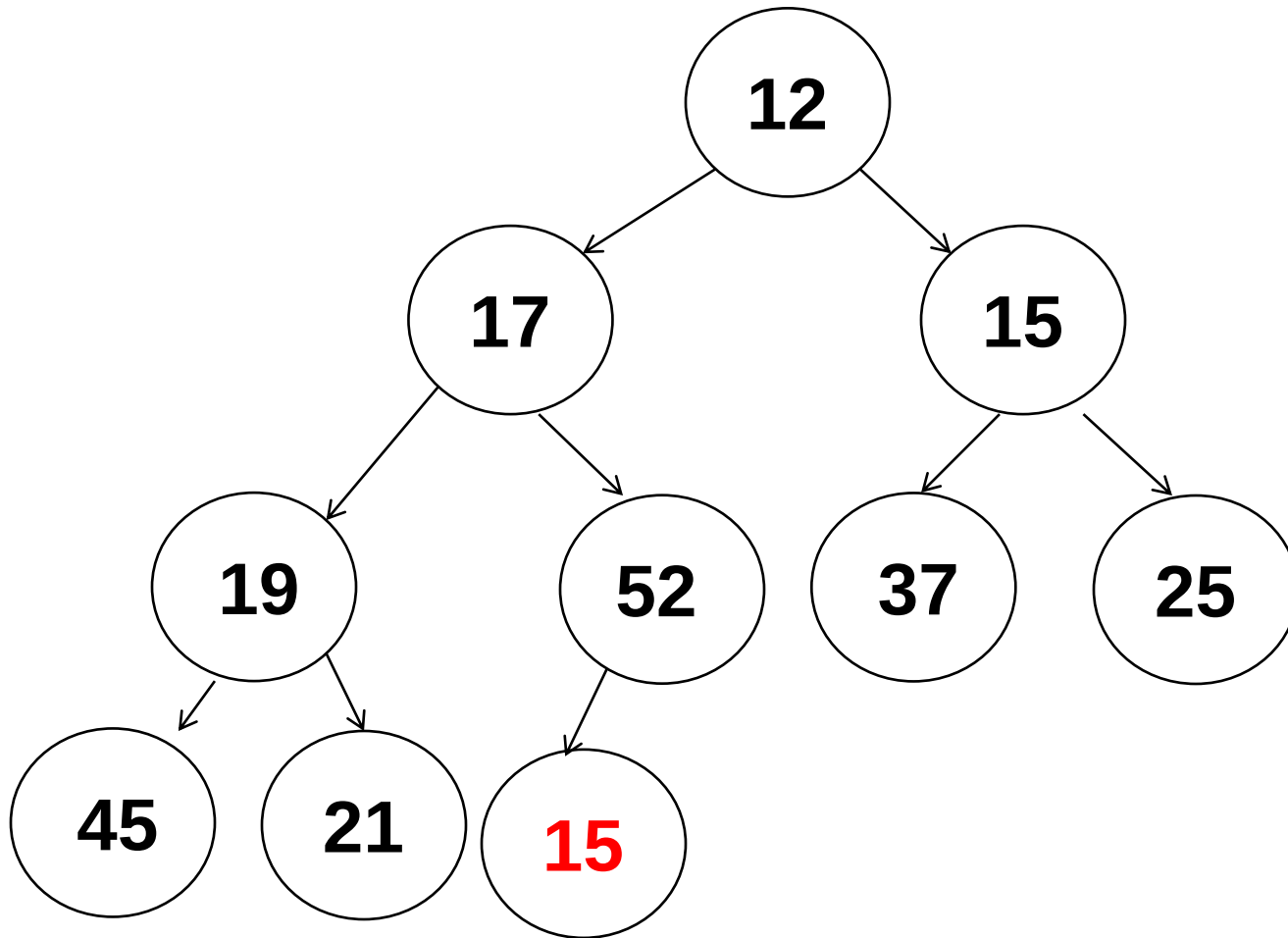


Add Operation

- ▶ Add new element to next open spot in array
- ▶ Swap with parent if new value is less than parent
- ▶ Continue back up the tree as long as the new value is less than new parent node

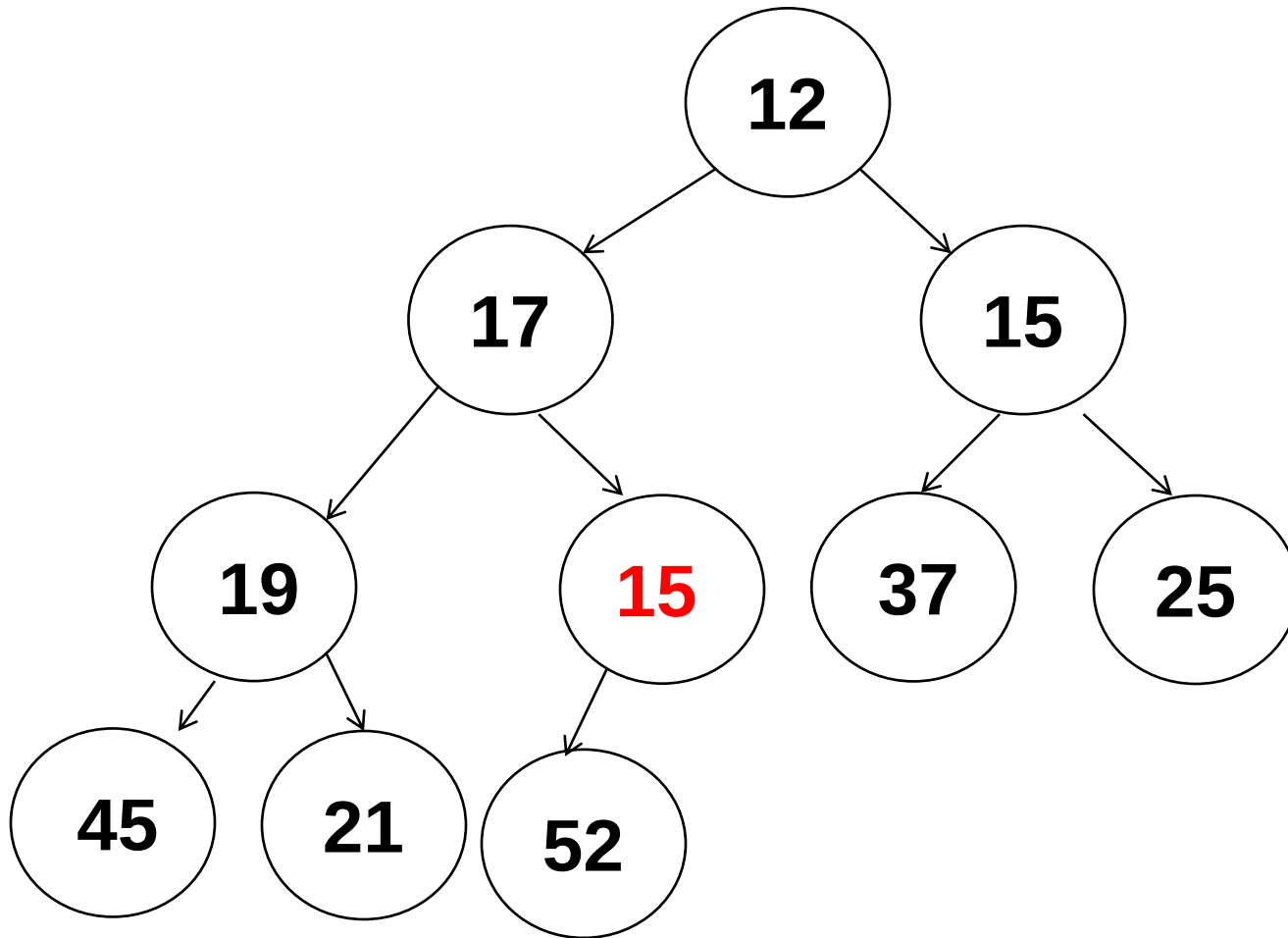
Add Example

- Add 15 to heap (initially next left most node)



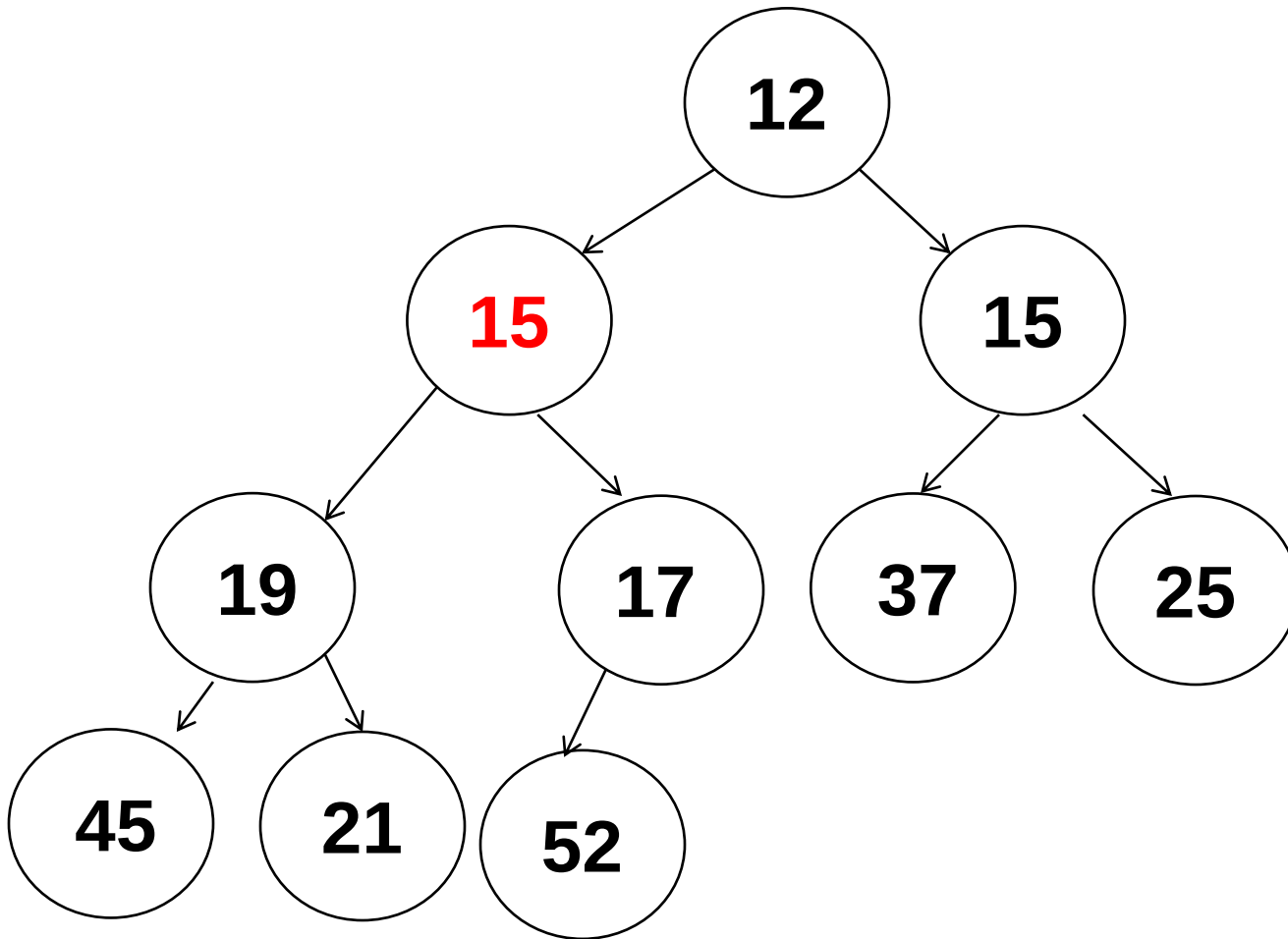
Add Example

- Swap 15 and 52



Enqueue Example

- Swap 15 and 17, then stop



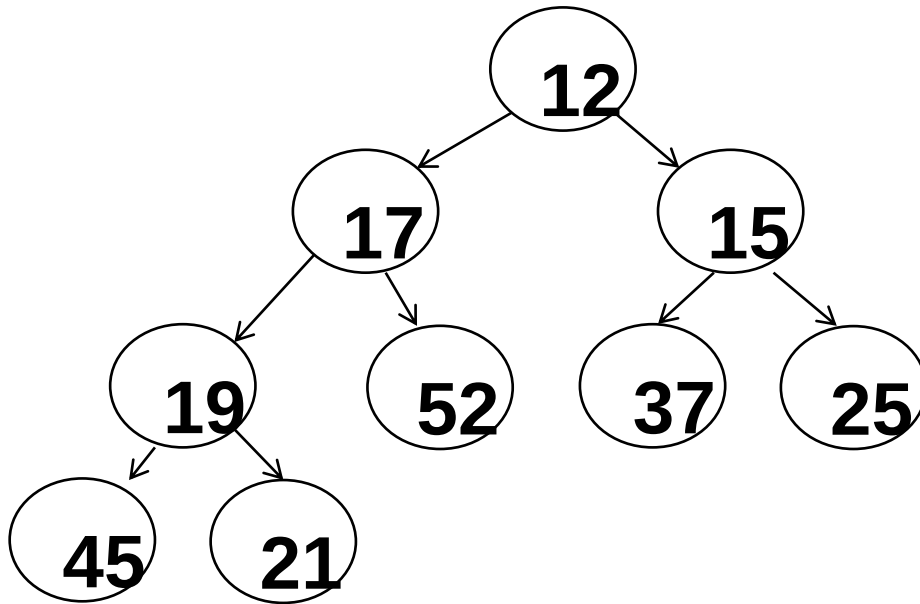
Add Example

- ▶ Insert the following values 1 at a time into a min heap:

16 9 5 8 13 8 8 5 5 19 27 9 3

Internal Storage

- ▶ Interestingly heaps are often implemented with an array instead of nodes



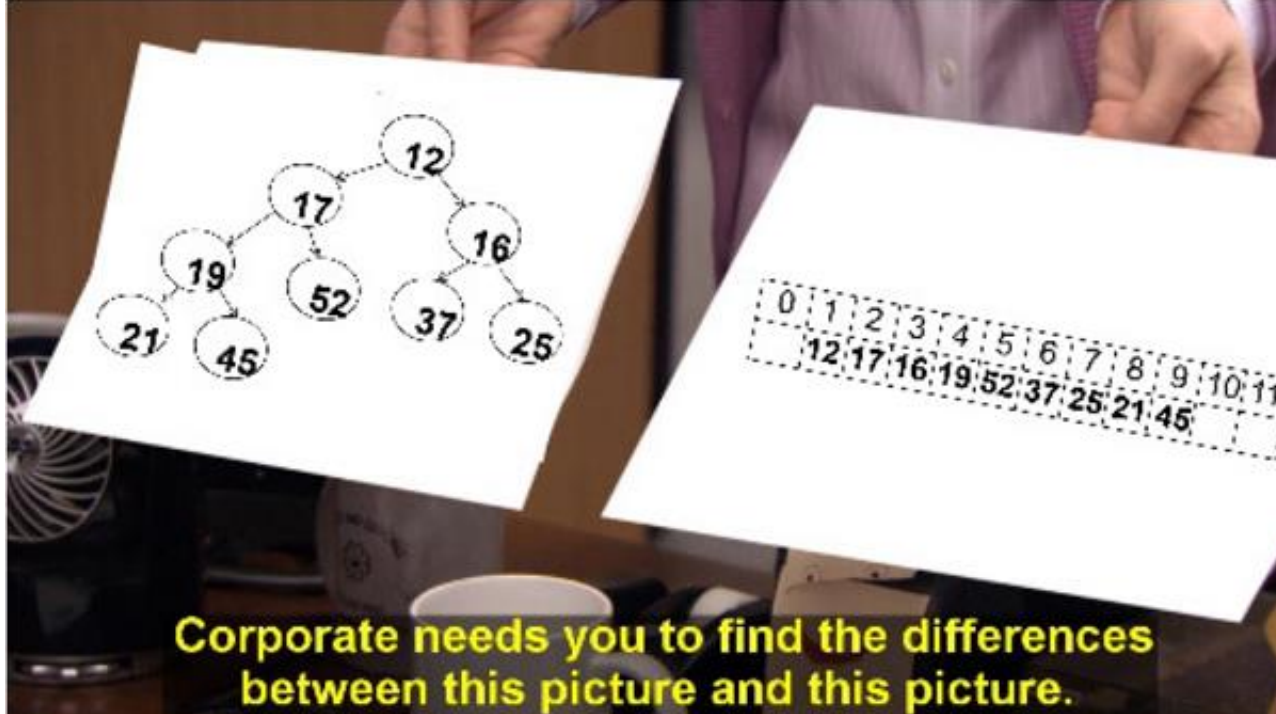
for element at index i :

parent index: $i / 2$

left child index: $i * 2$

right child index: $i * 2 + 1$

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	12	17	15	19	52	37	25	45	21						



**In Honor of
Elijah,
The Meme King,
Spring 2020**



PriorityQueue Class

```
public class PriorityQueue<E extends Comparable<? super E>>
{

    private int size;
    private E[] con;

    public PriorityQueue() {
        con = getArray(2);
    }

    private E[] getArray(int size) {
        return (E[]) (new Comparable[size]);
    }
}
```

PriorityQueue enqueue / add

```
public void enqueue(E val) {
    if ( size >= con.length - 1 )
        enlargeArray(con.length * 2);

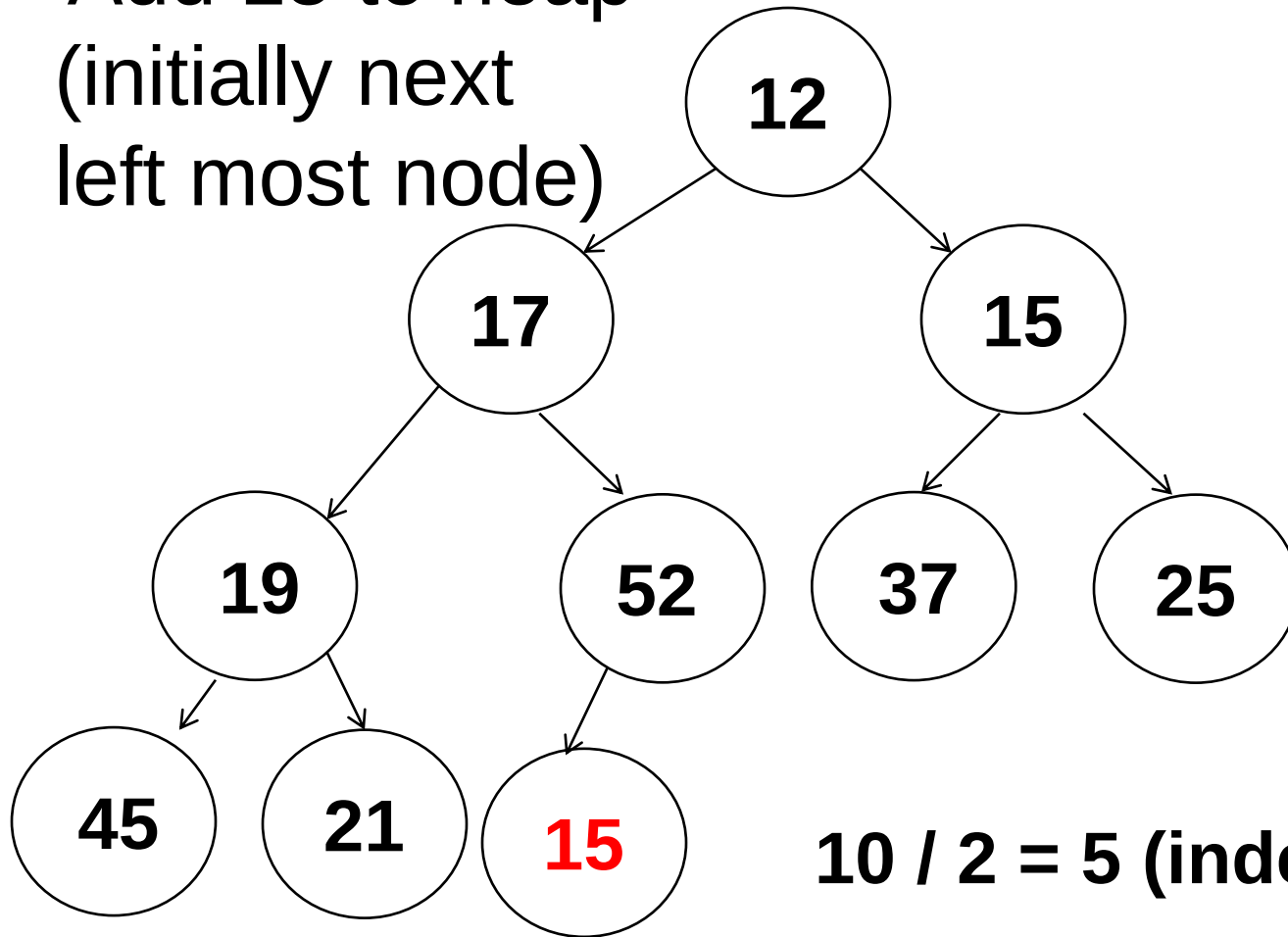
    size++;
    int indexToPlace = size;
    while ( indexToPlace > 1
            && val.compareTo( con[indexToPlace / 2] ) < 0 ) {

        con[indexToPlace] = con[indexToPlace / 2]; // swap
        indexToPlace /= 2; // change indexToPlace to parent
    }
    con[indexToPlace] = val;
}

private void enlargeArray(int newSize) {
    E[] temp = getArray(newSize);
    System.arraycopy(con, 1, temp, 1, size);
    con = temp;
}
```

Enqueue / add Example With Array Shown

- Add 15 to heap
(initially next
left most node)

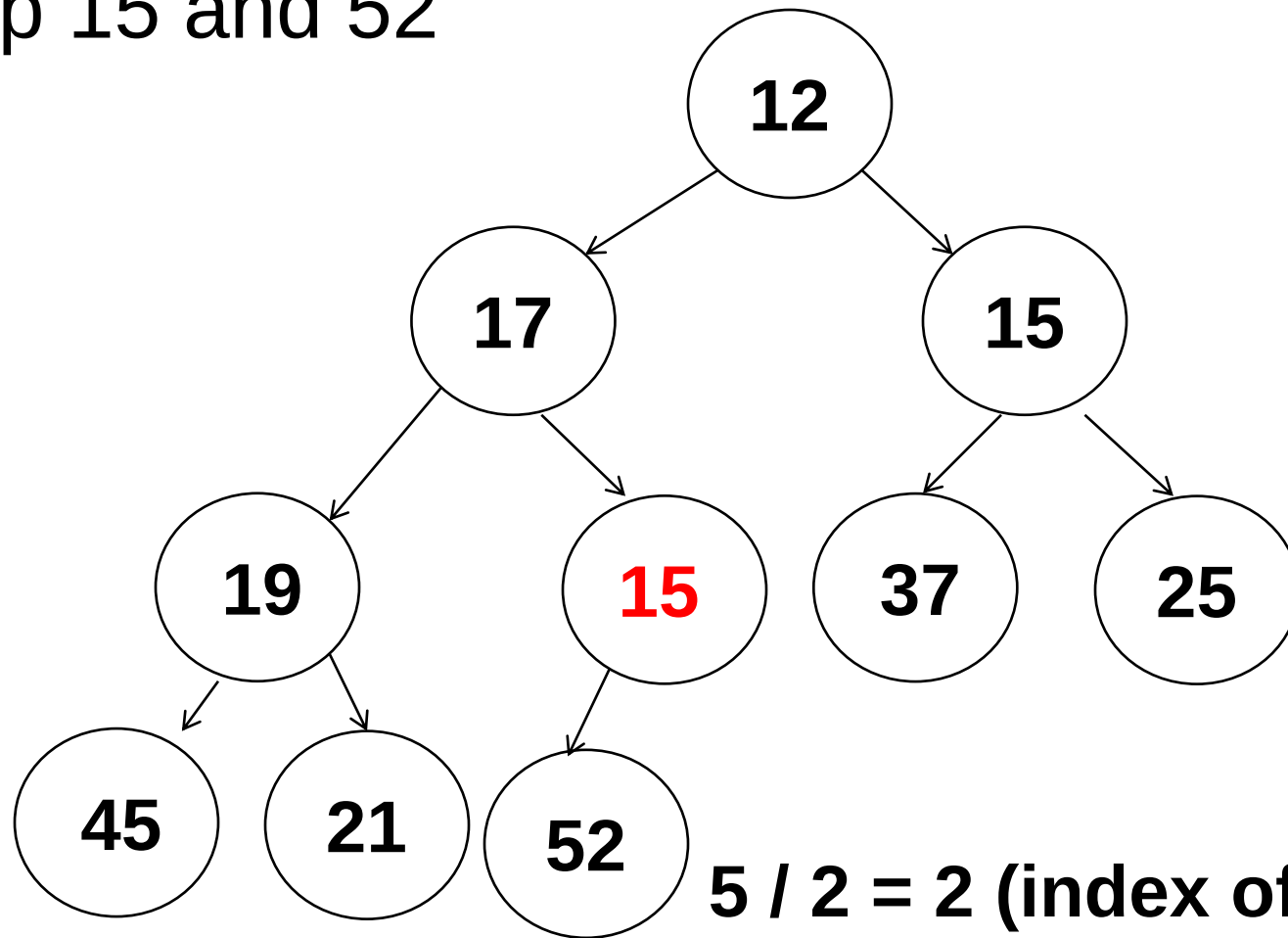


$$10 / 2 = 5 \text{ (index of parent)}$$

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	12	17	15	19	52	37	25	45	21	15					

Enqueue Example With Array Shown

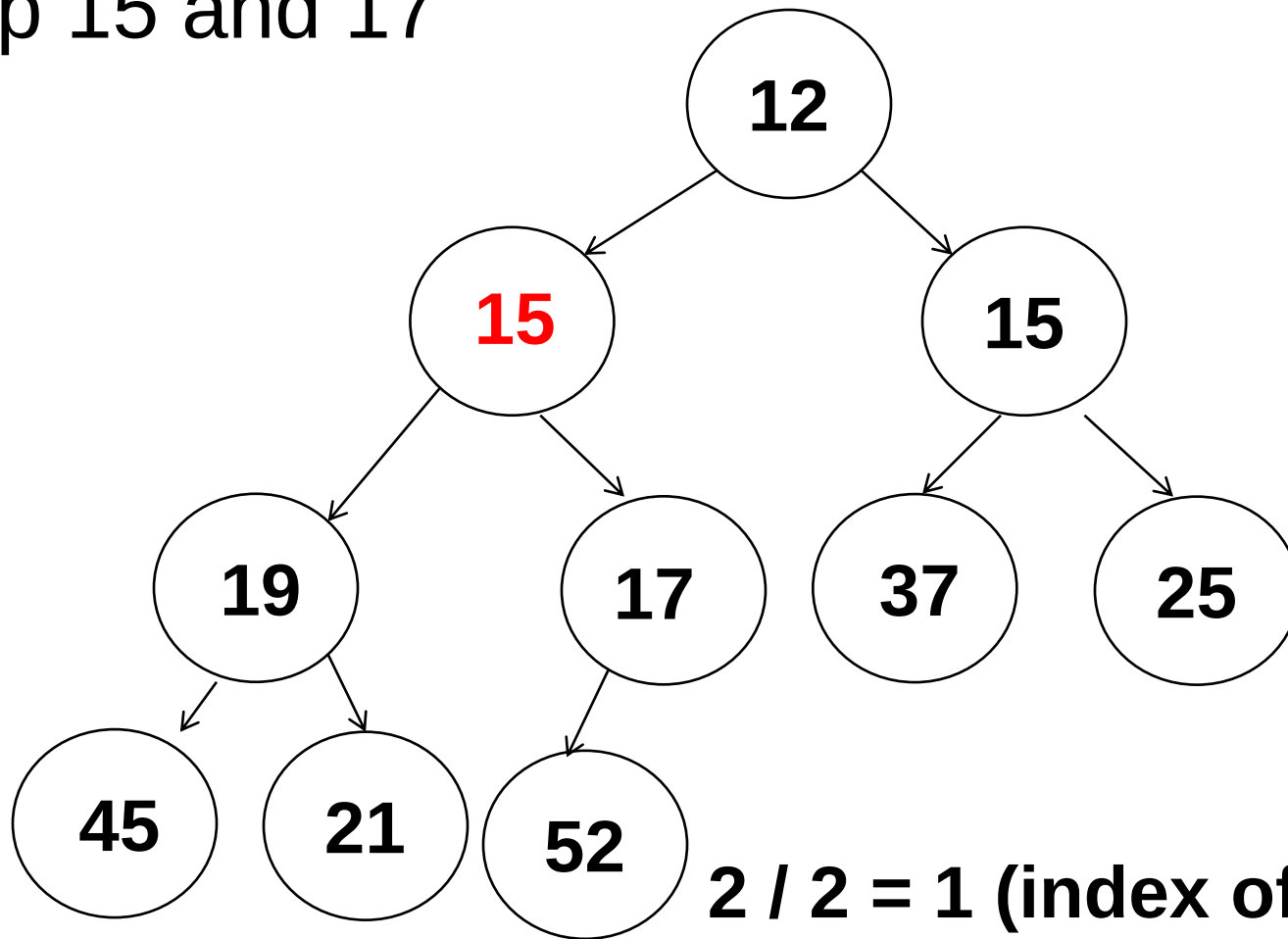
- Swap 15 and 52



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	12	17	15	19	15	37	25	45	21	52					

Enqueue Example With Array Shown

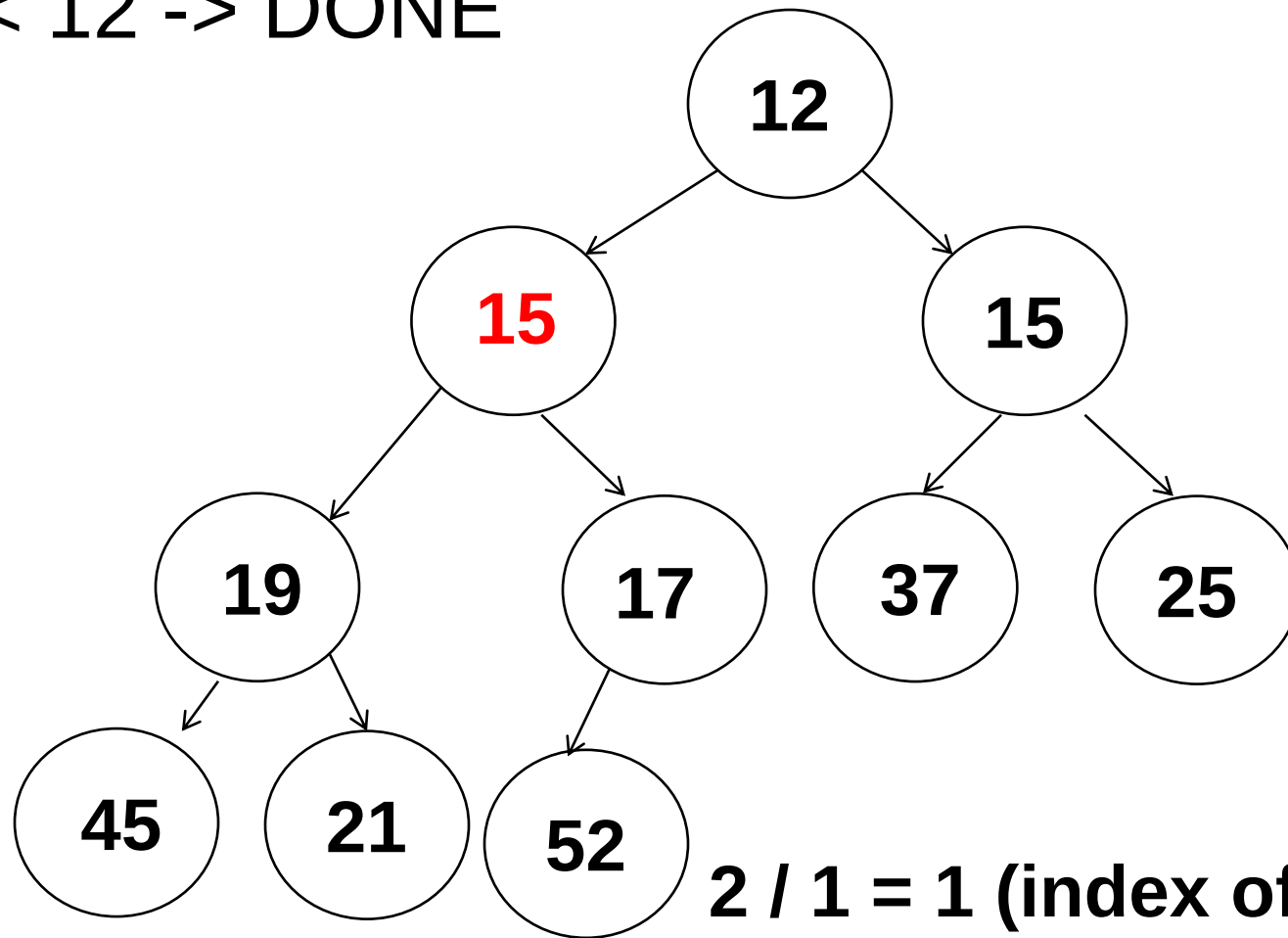
- Swap 15 and 17



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	12	15	15	19	17	37	25	45	21	52					

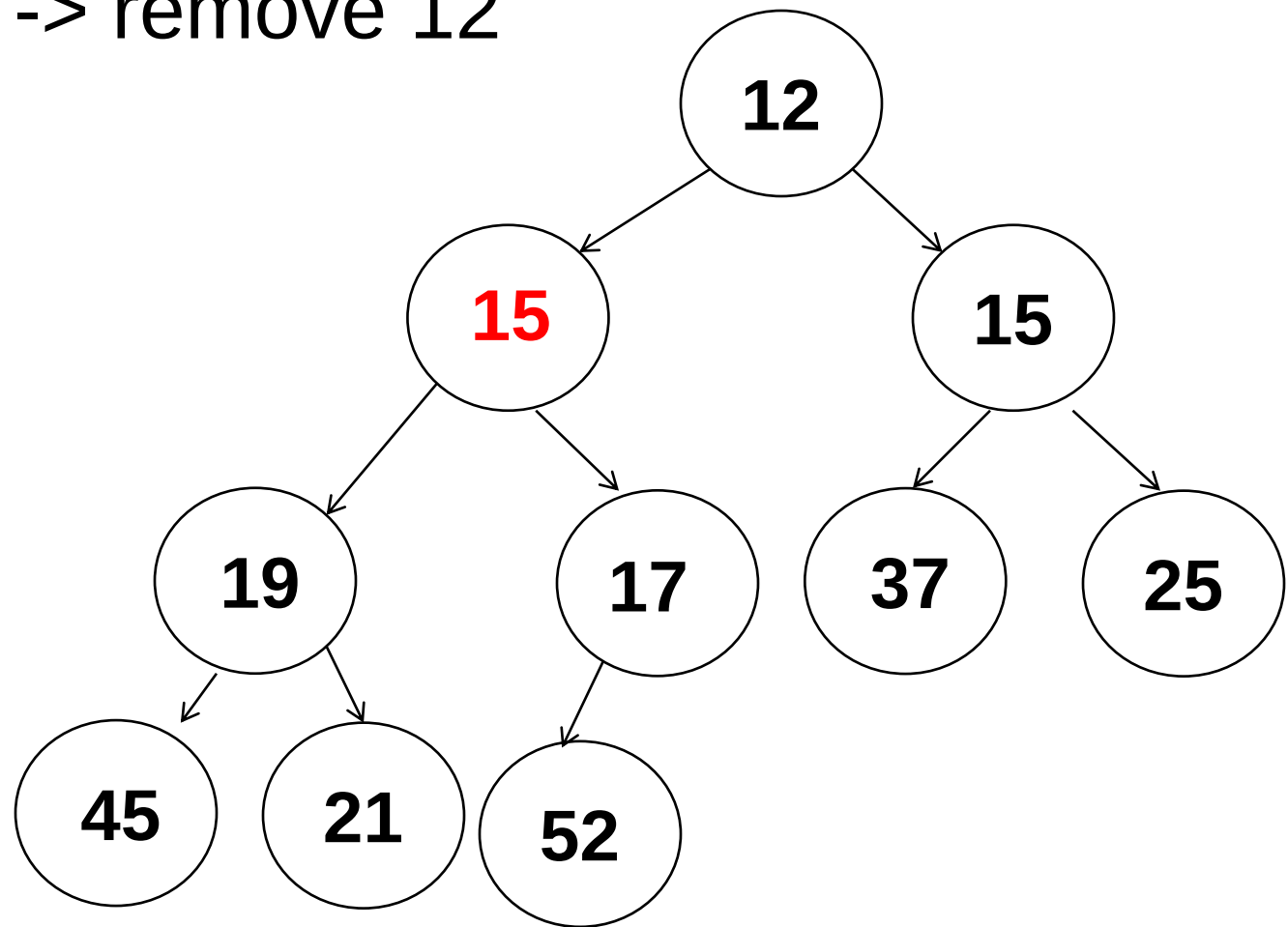
Enqueue Example With Array Shown

► 15 !< 12 -> DONE



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	12	15	16	19	17	37	25	45	21	52					

► Remove -> remove 12

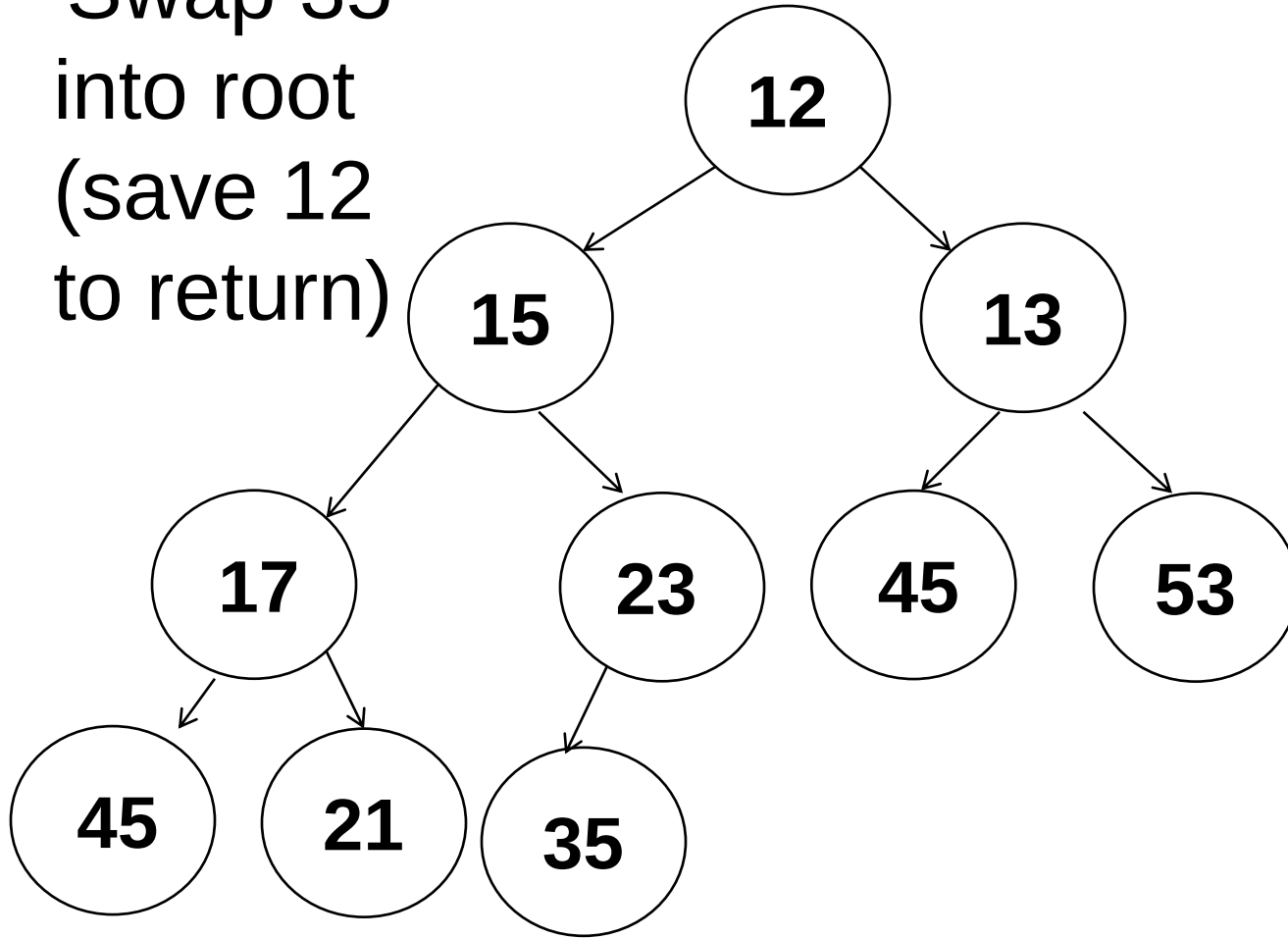


Remove / Dequeue

- ▶ min value / front of queue is in root of tree
- ▶ swap value from last node to root and move down swapping with smaller child unless values is smaller than both children

Deque Example

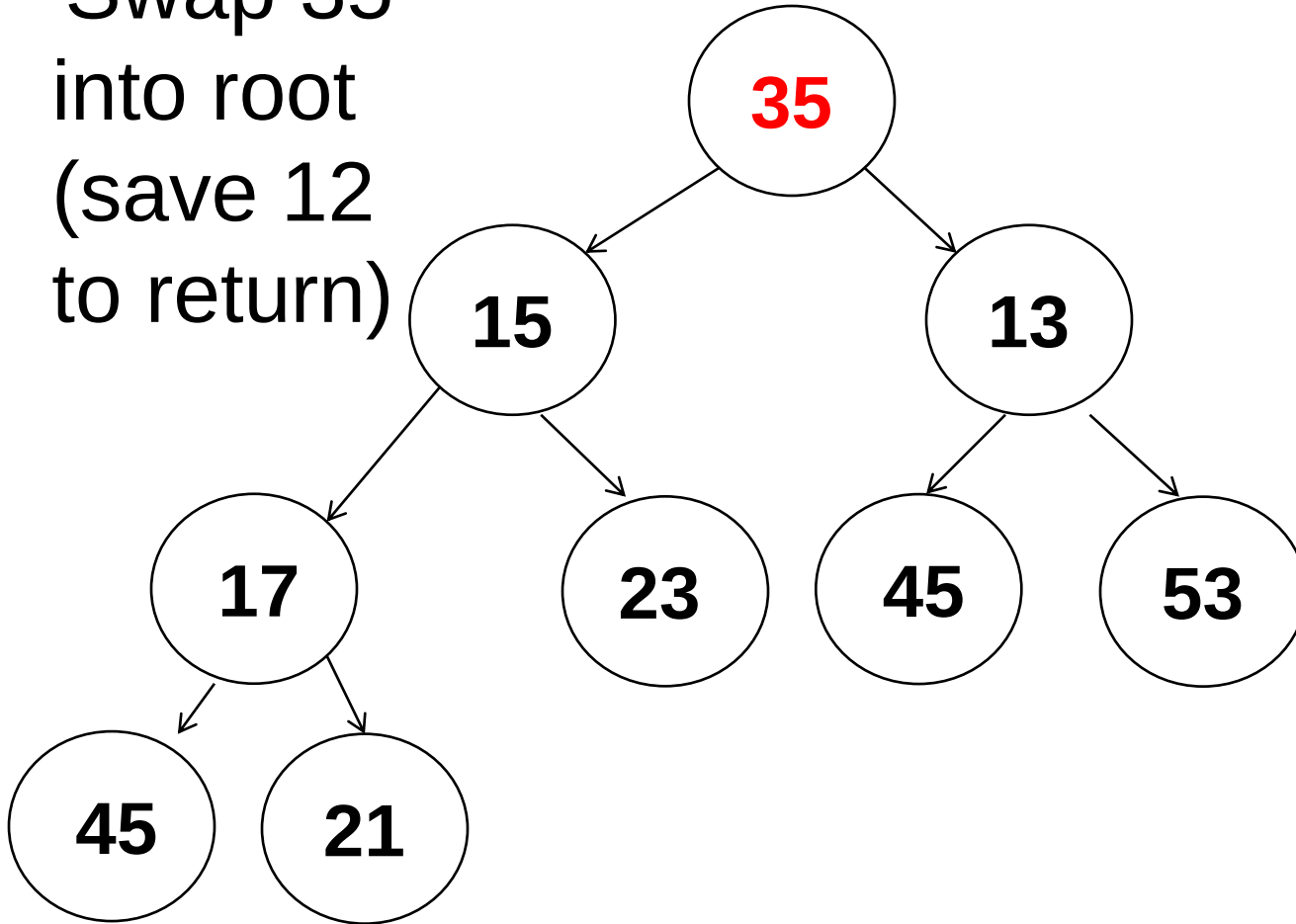
- Swap 35 into root (save 12 to return)



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	12	15	13	17	23	45	53	45	21	35					

Deque Example

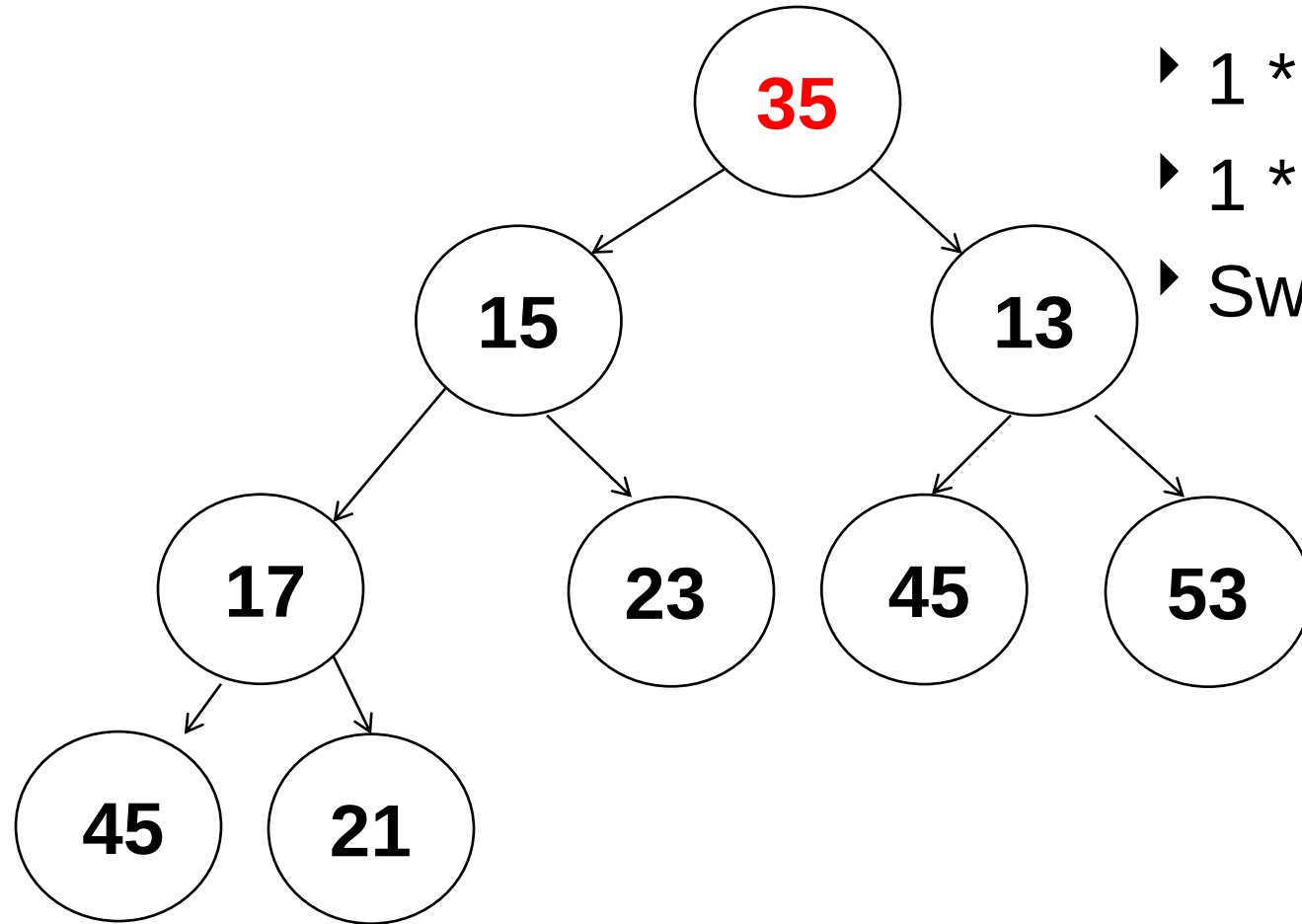
- Swap 35 into root (save 12 to return)



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	35	15	13	17	23	45	53	45	21						

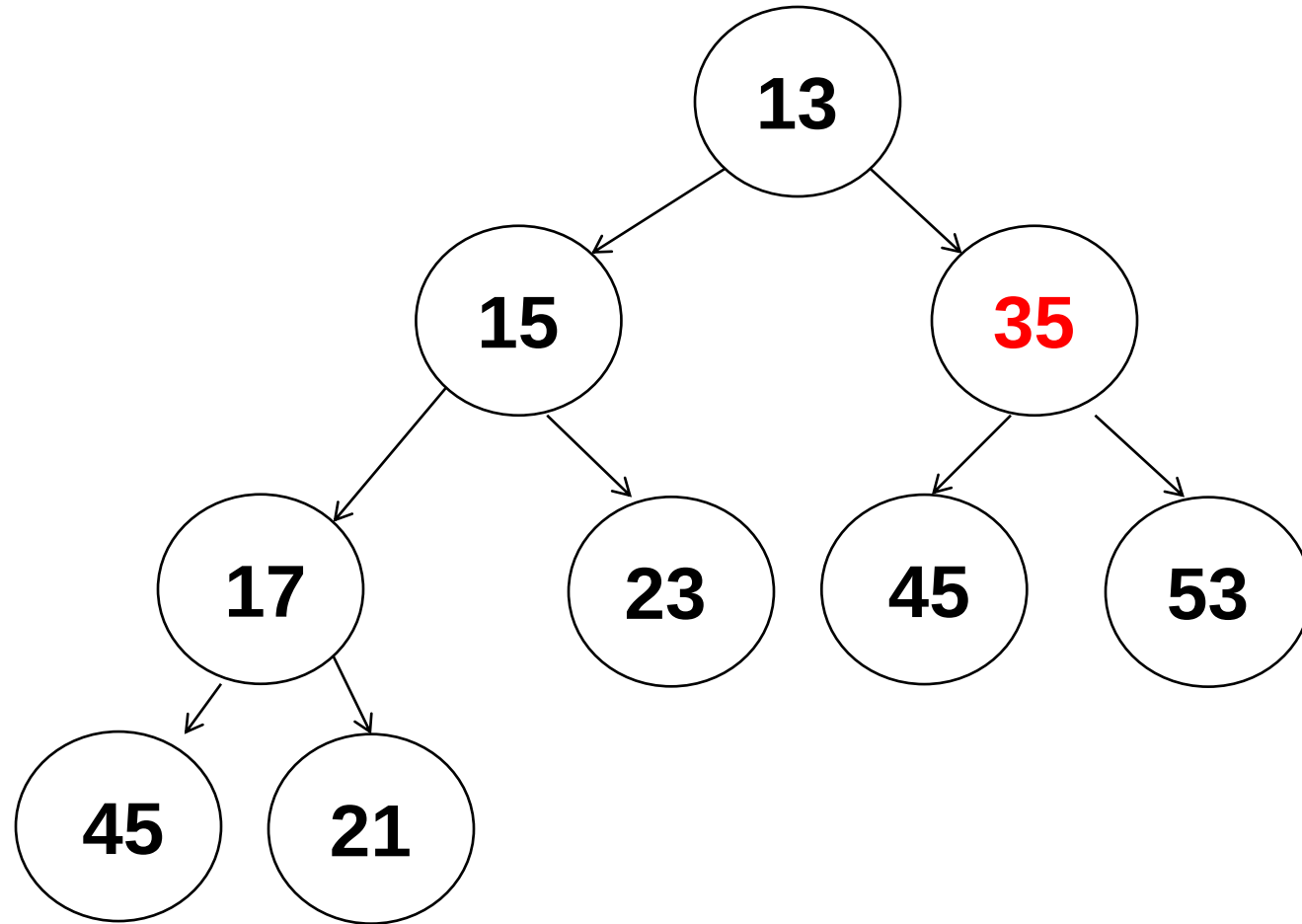
Dequeueue Example

- ▶ Min child?
- ▶ $1 * 2 = 2 \rightarrow 15$
- ▶ $1 * 2 + 1 = 3 \rightarrow 13$
- ▶ Swap with 13



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	35	15	13	17	23	45	53	45	21						

Deque Example



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	13	15	35	17	23	45	53	45	21						

Dequeue Example

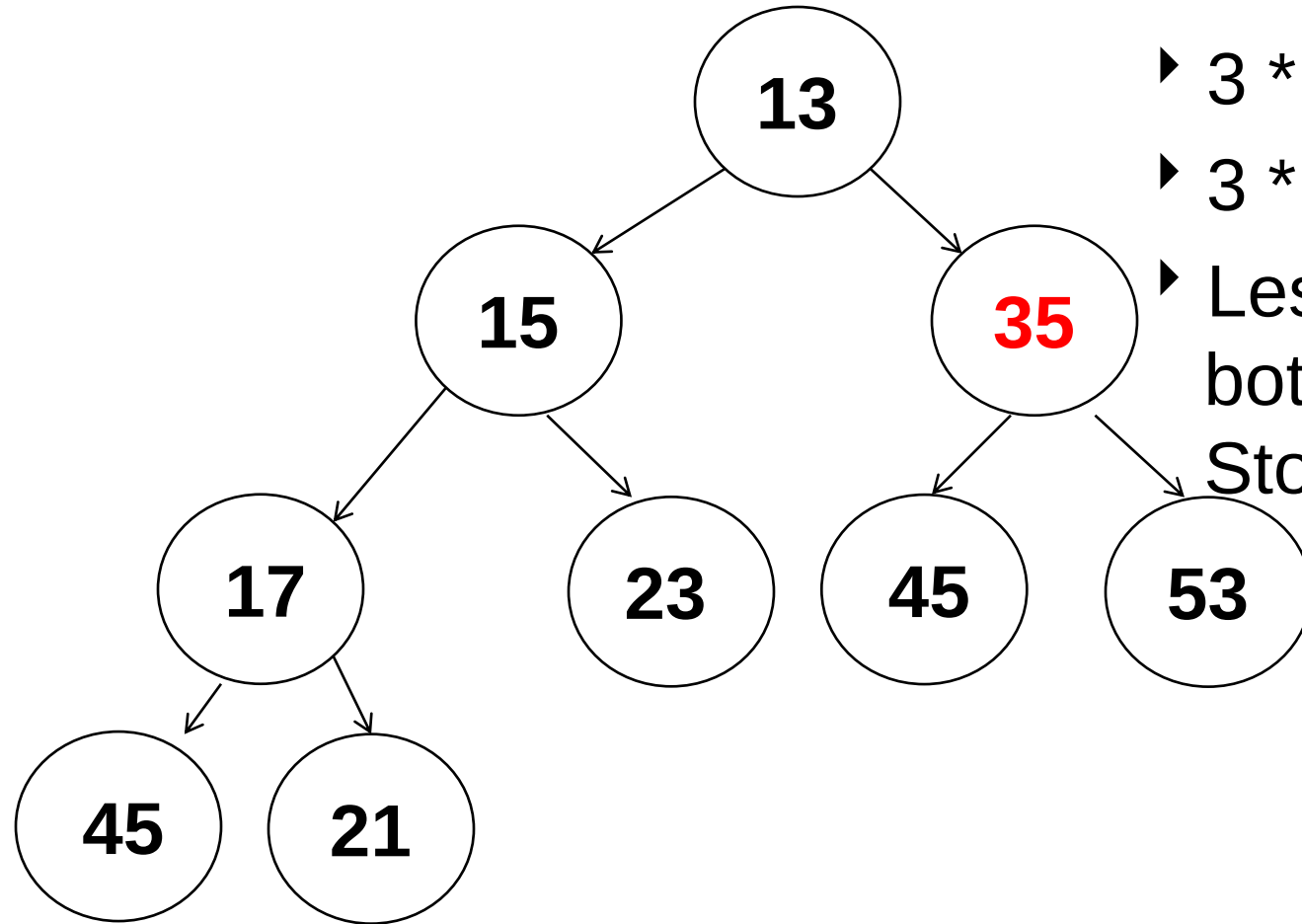
‣ Min child?

‣ $3 * 2 = 6 \rightarrow 45$

‣ $3 * 2 + 1 = 7 \rightarrow 53$

‣ Less than or equal to
both of my children!

Stop!



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	13	15	35	17	23	45	53	45	21						

Deque Code

```
public E dequeue( ) {
    E top = con[1];
    int hole = 1;
    boolean done = false;
    while ( hole * 2 < size && ! done ) {
        int child = hole * 2;
        // see which child is smaller
        if ( con[child].compareTo( con[child + 1] ) > 0 )
            child++;    // child now points to smaller

        // is replacement value bigger than child?
        if (con[size].compareTo( con[child] ) > 0 ) {
            con[hole] = con[child];
            hole = child;
        }
        else
            done = true;
    }
    con[hole] = con[size];
    size--;
    return top;
}
```

Clicker 3 - PriorityQueue Comparison

- ▶ Run a Stress test of PQ implemented with Heap and PQ implemented with BinarySearchTree

▶ What will result be?

- A. Heap takes half the time or less of BST
- B. Heap faster, but not twice as fast
- C. About the same
- D. BST faster, but not twice as fast
- E. BST takes half the time or less of Heap

Data Structures

- ▶ Data structures we have studied
 - arrays, array based lists, linked lists, maps, sets, stacks, queues, trees, binary search trees, graphs, hash tables, red-black trees, priority queues, heaps, tries
- ▶ Most program languages have some built in data structures, native or library
- ▶ Must be familiar with performance of data structures
 - best learned by implementing them yourself

Data Structures

► We have *not* covered every data structure

Abstract data types [\[edit source | edit beta \]](#)

- Container
- Map/Associative array/Dictionary
- Multimap
- List
- Set
- Multiset
- Priority queue
- Queue
- Deque
- Stack
- String
- Tree
- Graph

Some properties of abstract data types:

Structure	Stable	Unique	Cells per Node
Bag (multiset)	no	no	1
Set	no	yes	1
List	yes	no	1
Map	no	yes	2

"Stable" means that input order is retained. Other stru

Arrays [\[edit source | edit beta \]](#)

- Array
- Bidirectional map
- Bit array
- Bit field
- Bitboard
- Bitmap
- Circular buffer
- Control table
- Image
- Dynamic array
- Gap buffer
- Hashed array tree
- Heightmap
- Lookup table
- Matrix
- Parallel array
- Sorted array
- Sparse array
- Sparse matrix
- Iliffe vector
- Variable-length array

Lists [\[edit source | edit beta \]](#)

- Doubly linked list
- Linked list
- Self-organizing list
- Skip list
- Unrolled linked list
- VList
- Xor linked list
- Zipper
- Doubly connected edge list
- Difference list

Heaps [\[edit source | edit \]](#)

- Heap
- Binary heap
- **Weak heap**
- Binomial heap
- Fibonacci heap
 - AF-heap
- 2-3 heap
- Soft heap
- Pairing heap
- Leftist heap
- Treap
- Beap
- Skew heap
- Ternary heap
- D-ary heap

Trees [\[edit source | edit t](#)

In these data structures each

- Tree
- Radix tree
- Suffix tree
- Suffix array
- Compressed suffix array
- FM-index
- Generalised suffix tree
- B-tree
- Judy array
- **X-fast tree**
- **Y-fast tree**
- Ctree

Multitree [\[edit source](#)

Graphs [\[edit source | edit beta \]](#)

- Graph
- Adjacency list
- Adjacency matrix
- Graph-structured stack
- Scene graph
- Binary decision diagram
- Zero suppressed decision diagram
- And-inverter graph
- Directed graph
- Directed acyclic graph
- Propositional directed acyclic graph
- Multigraph
- Hypergraph

Other [\[edit source | edit beta \]](#)

- Lightmap
- Winged edge
- Doubly connected edge list
- Quad-edge
- Routing table
- Symbol table

http://en.wikipedia.org/wiki/List_of_data_structures

Data Structures

- ▶ deque, b-trees, quad-trees, binary space partition trees, skip list, sparse list, sparse matrix, union-find data structure, Bloom filters, AVL trees, 2-3-4 trees, and more!
- ▶ Must be able to learn new and apply new data structures