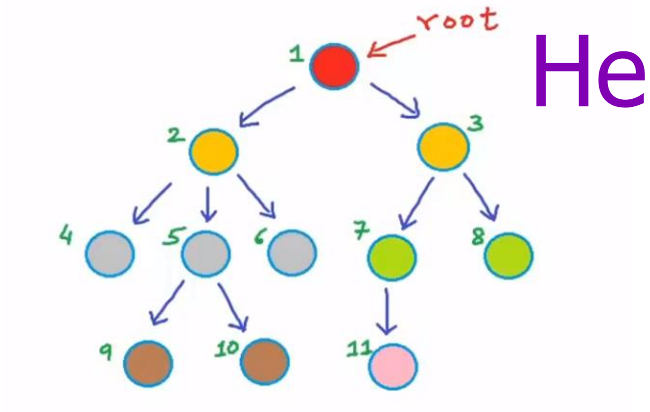
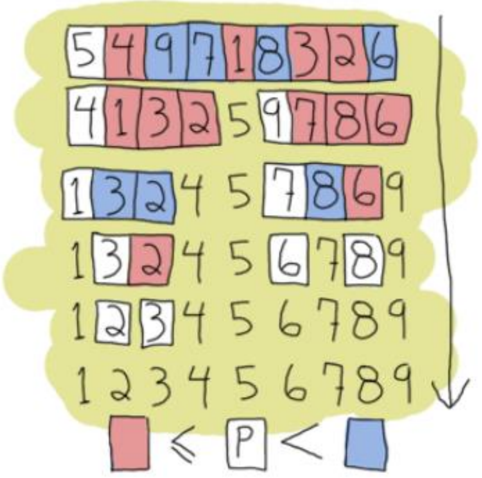
ECE 250 Data Structures & Algorithms



Heaps & Priority Queues



Ziqiang Patrick Huang

Electrical and Computer Engineering

University of Waterloo

WATERLOO | ENGINEERING

New ADT: Priority Queue

- Recall Queues: First in, First out
  - void enqueue(T): add to end of queue
  - T dequeue(): take from front of queue
  - T peek(): look at front of queue

- Priority Queues: highest priority first
  - void enqueue(T, int): add with a given priority
  - T dequeue(): take highest item
  - T peek(): look at highest priority item

Priority Queue: Why?

- Why would we want this?
  - Task scheduling
    - OS interrupt handling
  - Dijkstra’s shortest path algorithm
    - Find the next node to explore in PQ
    - Nodes added to PQ with distance as “priority”
  - Huffman compression algorithm
    - Use PQ to construct a Huffman tree based on the frequency of each symbol
    - Huffman tree then is used to encode data
  - … and many more

Naïve Implementation of PQ

- Naïve implementation
  - Unsorted Linked List
    - Enqueue = Add to Front: O(1)
    - Dequeue = Remove Max [or Min]: O(n)
    - Peek = Find Max [or Min]: O(n)
  - Sorted Linked List
    - Enqueue = sorted insert: O(n)
    - Dequeue = remove from front: O(1)
    - Peek = Get Front: O(1)
- Better implementation
  - Balanced BST
    - Enqueue = BST Add: O(log(n))
    - Dequeue = BST remove: O(log(n))
    - Peek = BST search: O(log(n))

Can we do better?

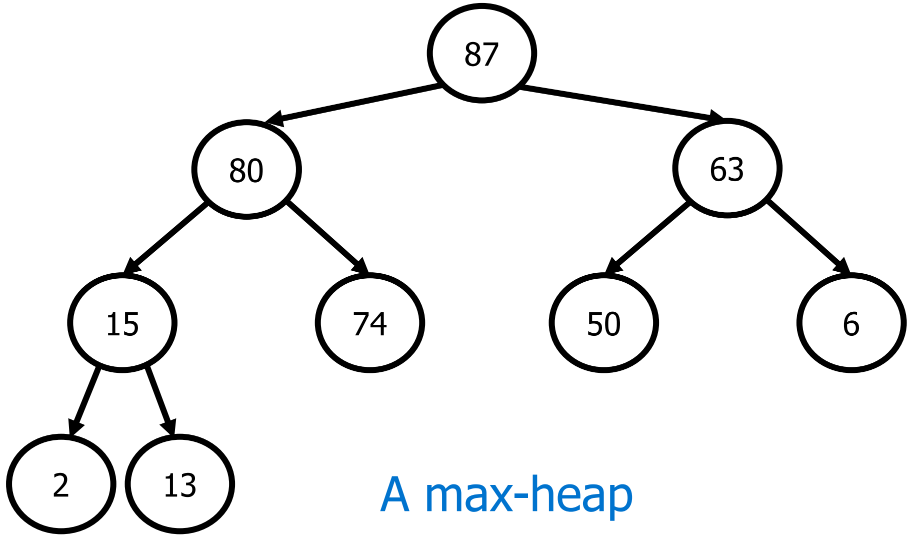
New Data Structure: Heap

	Sorted Linked List	Unsorted Linked List	Balanced BST	Heap
enqueue	O(n)	O(1)	O(log(n))	O(log(n))
dequeue	O(1)	O(n)	O(log(n))	O(log(n))
peek	O(1)	O(n)	O(log(n))	O(1)

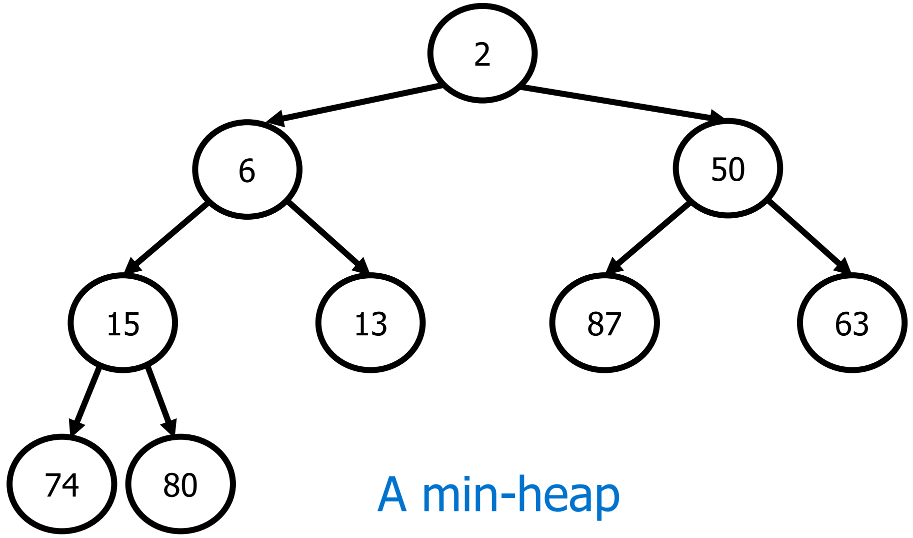
- Heap gives us good performance in every category!
  - O(1) access for largest (or smallest) element
  - Do not confuse this with the other "heap" for dynamic memory allocation

Heap

- Tree-like structure, different rule than BST
  - Complete tree:
    - All levels (except maybe last) completely full
    - Last level filled from left to right
  - Each node is greater/smaller than its two children for a Max/Min-Heap



A max-heap



A min-heap

WATERLOO | ENGINEERING

Priority Queue with Heap

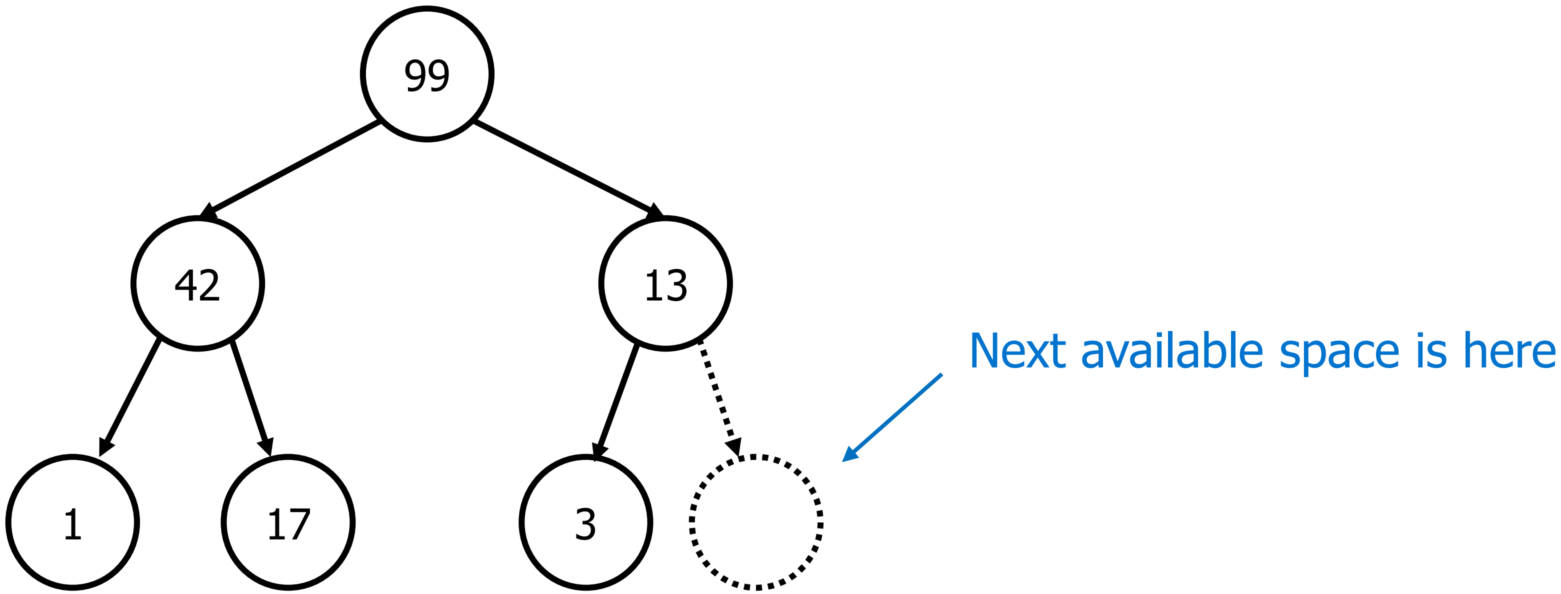
- Heap provide exactly what we want for priority queues
  - Order the heap by priority
  - Highest priority = largest/smallest ? → max/min-heap
  - Peek = look at root → O(1)
  - Enqueue/Dequeue need to be implemented efficiently (i.e., O(log(n)))

Heap Insertion

- Adding to a heap:
  - Place item in next open slot
    - Leftmost un-taken spot on unfilled level
    - First slot of a new level (if all levels are filled)
    - Ensure “completeness”
    - May violate ordering rules
  - Bubble up
    - Swap with parent if greater(max-heap) or smaller(min-heap)
    - Repeat until heap ordering is restored

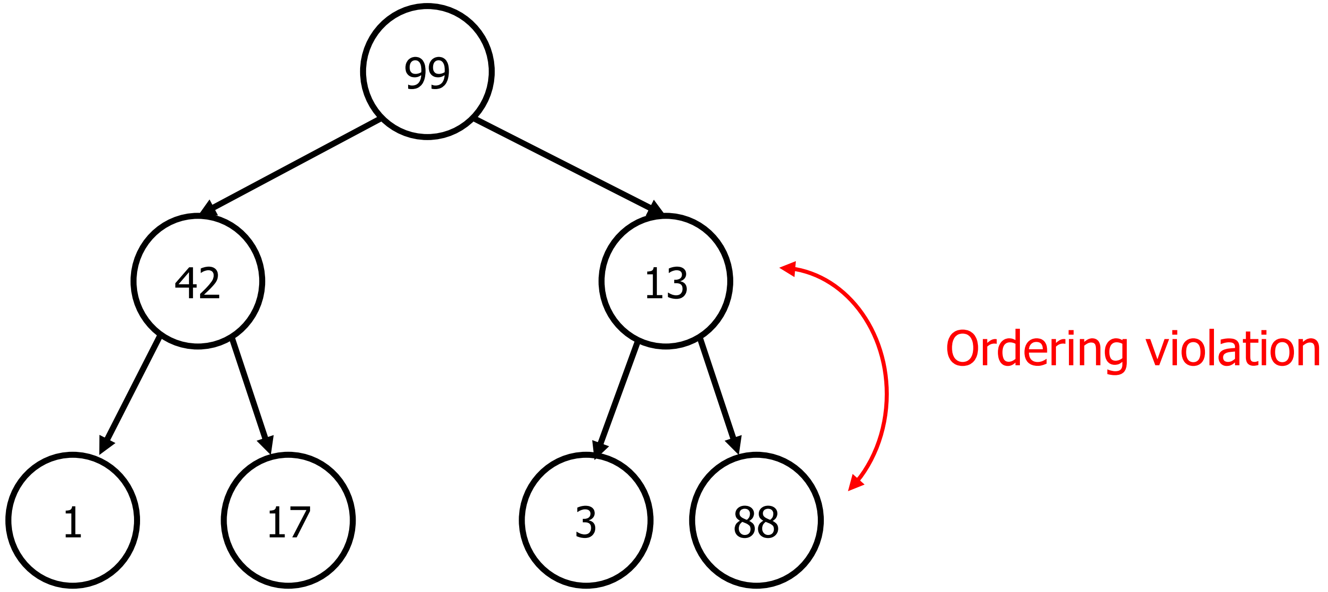
Heap Insertion Example

Add 88 to this max-heap



Heap Insertion Example

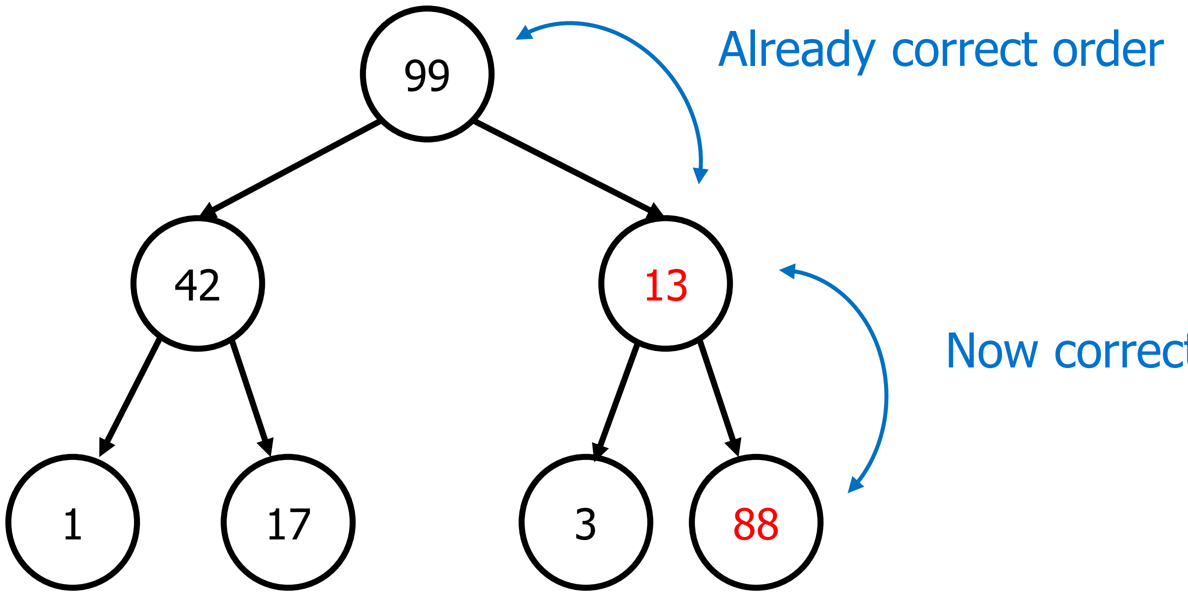
Add 88 to this max-heap



WATERLOO | ENGINEERING

Heap Insertion Example

Add 88 to this max-heap

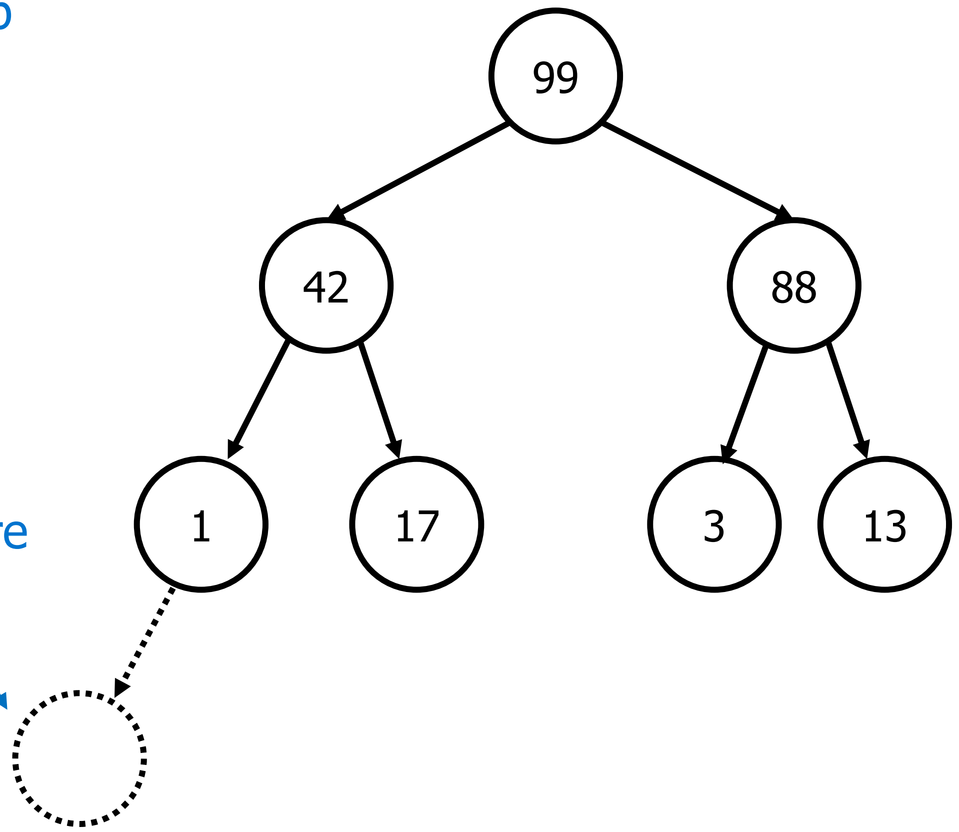


Done

Heap Insertion Example

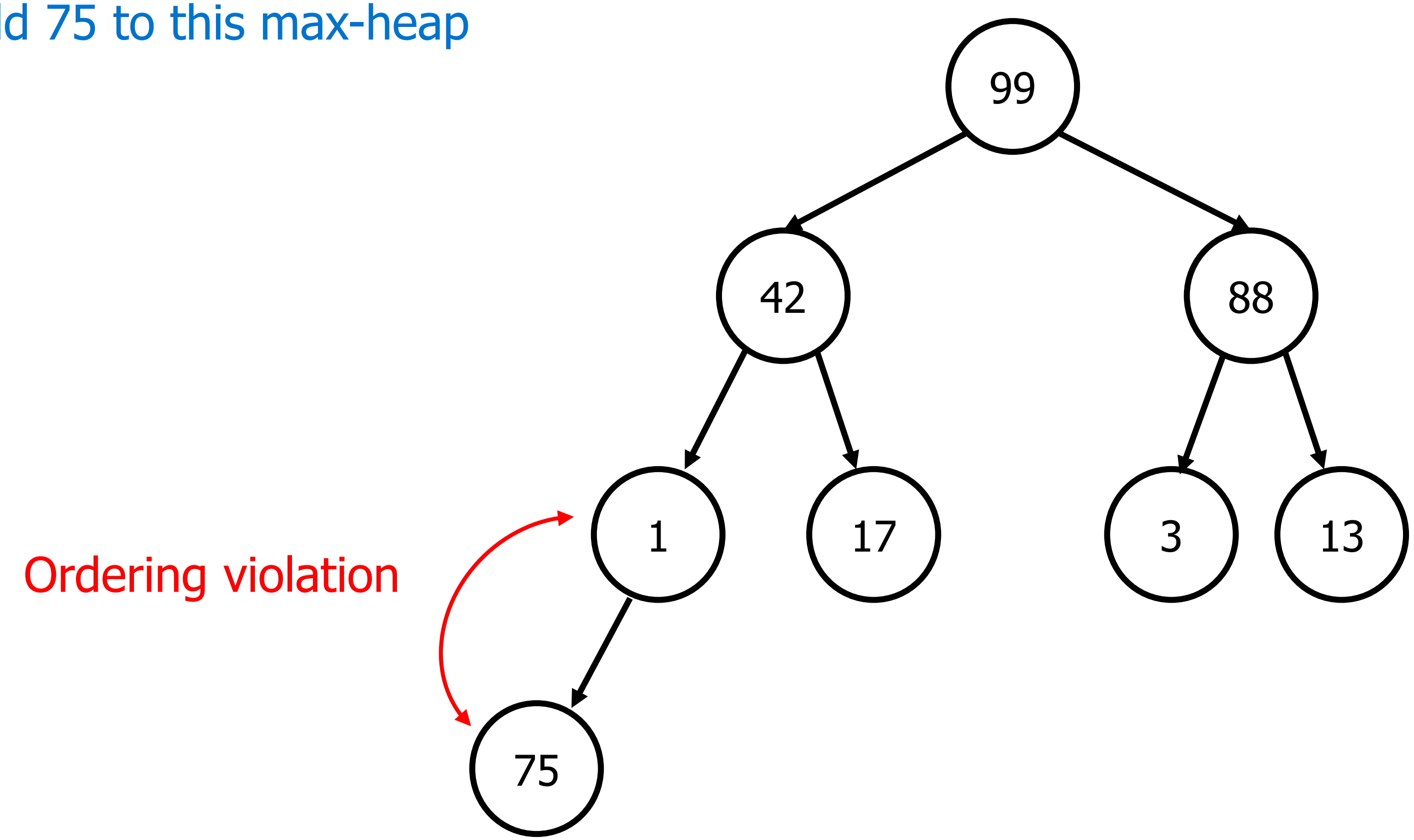
Add 75 to this max-heap

Next available space is here



Heap Insertion Example

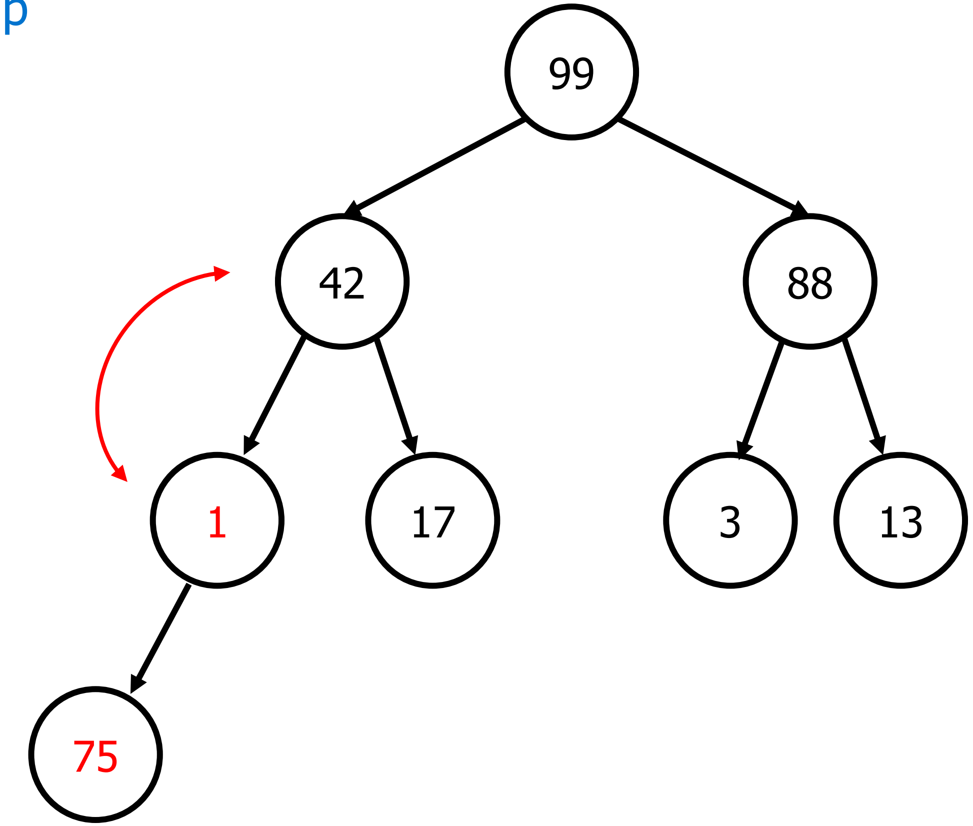
Add 75 to this max-heap



Heap Insertion Example

Add 75 to this max-heap

Ordering violation

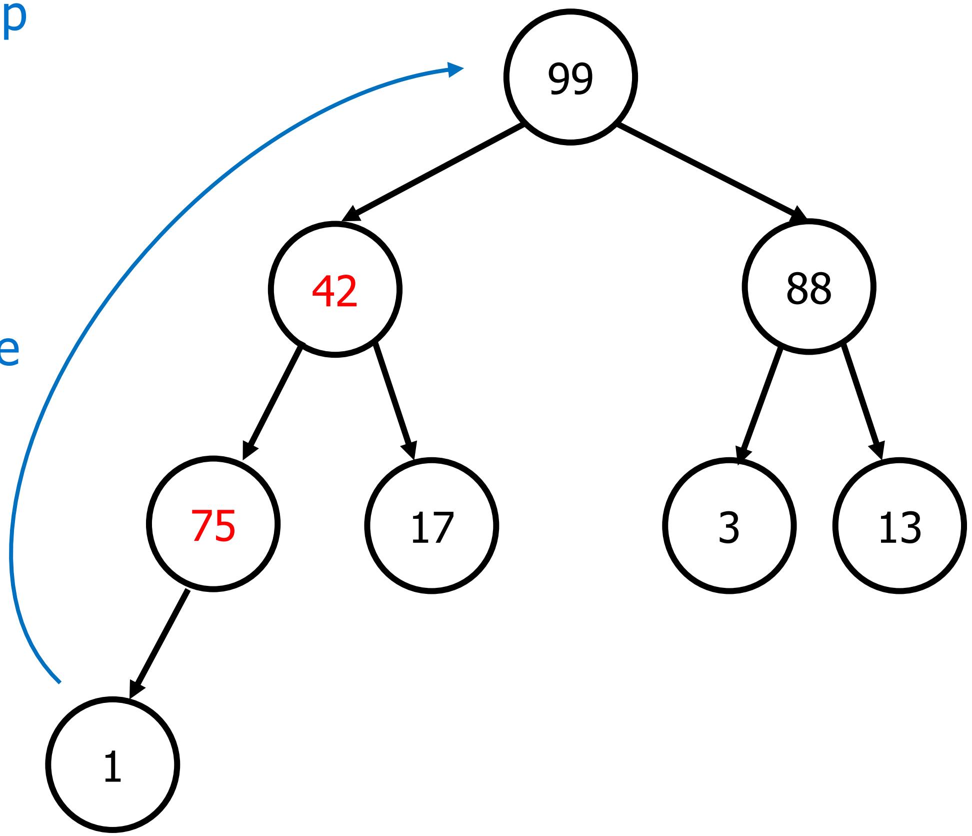


Heap Insertion Example

Add 75 to this max-heap

Guaranteed O(log(n))

Because of complete tree



Bubble up preserve “completeness”

Done

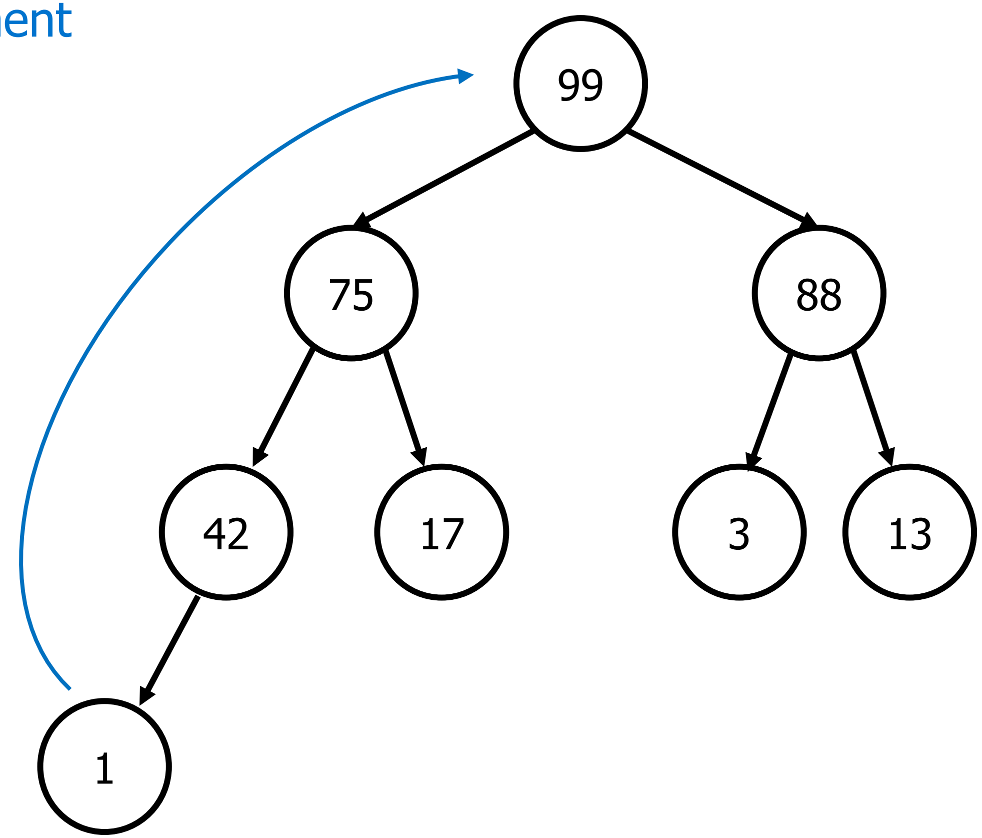
WATERLOO | ENGINEERING

Heap Deletion

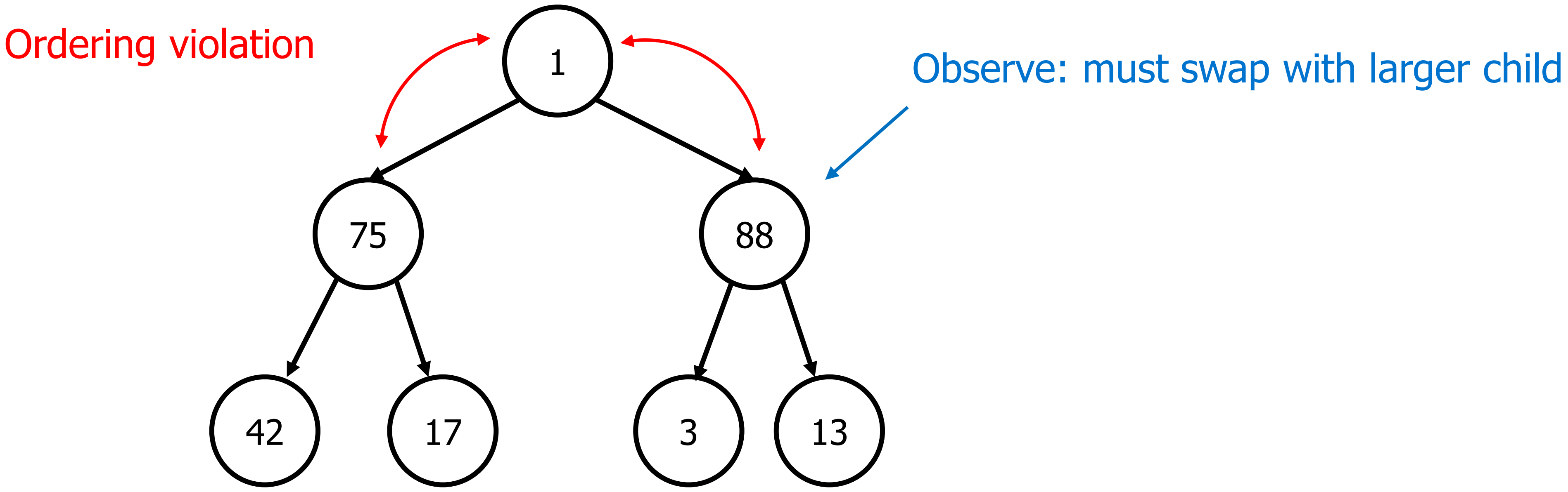
- Deleting max/min (root) from a heap:
  - Replace max/min (root) with rightmost item in last level
    - Ensure “completeness”
    - Violated ordering rules
  - Bubble down
    - Compare with its two children and determine which is largest/smallest
    - Make that one parent, swap if needed
    - Repeat until heap ordering is restored

Heap Deletion Example

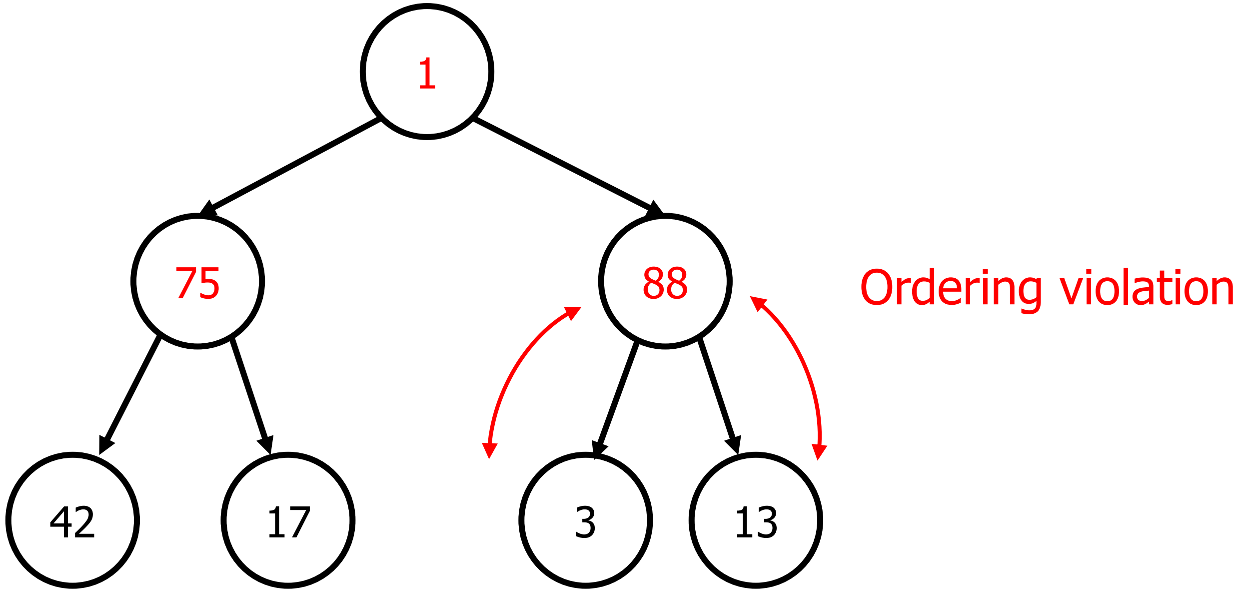
Now remove max element



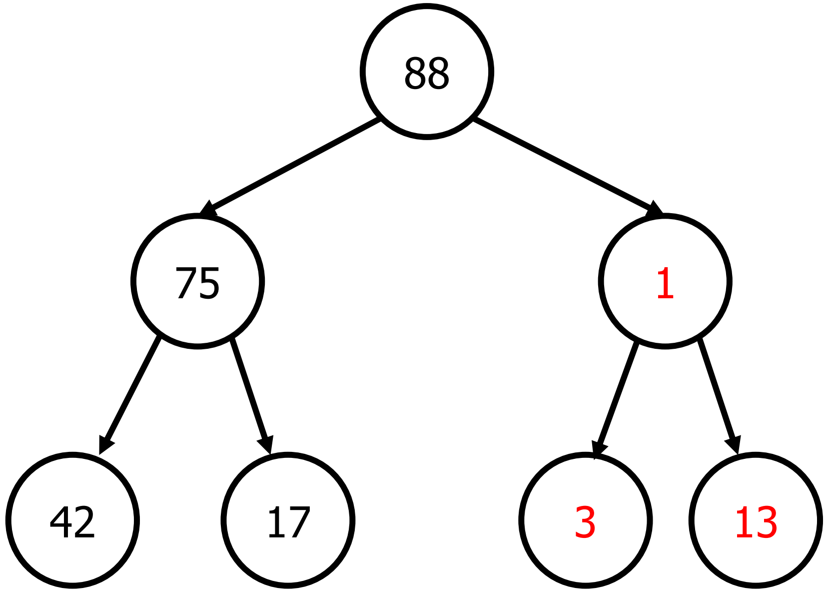
Heap Deletion Example



Heap Deletion Example



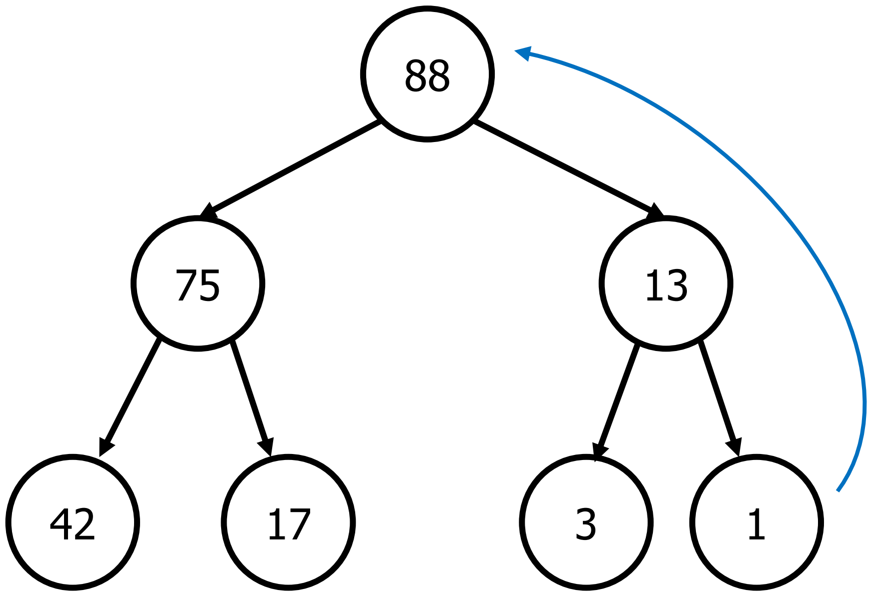
Heap Deletion Example



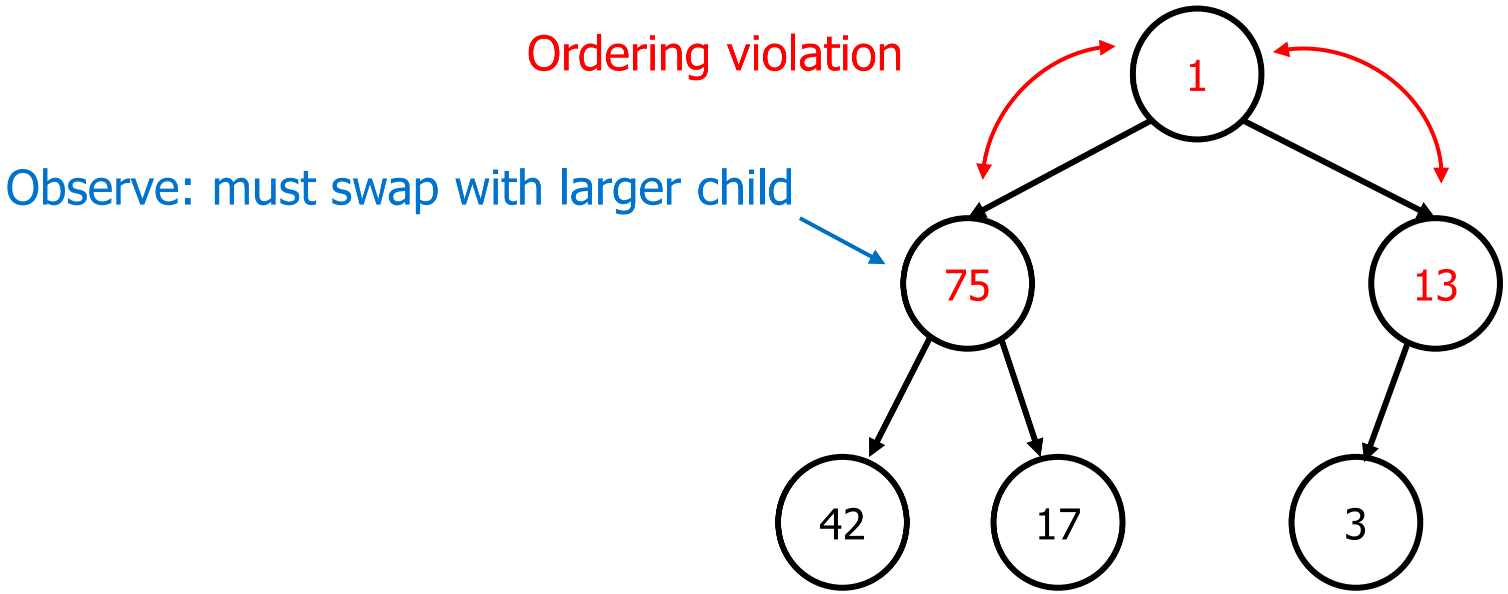
Done

Heap Deletion Example

Remove max element again

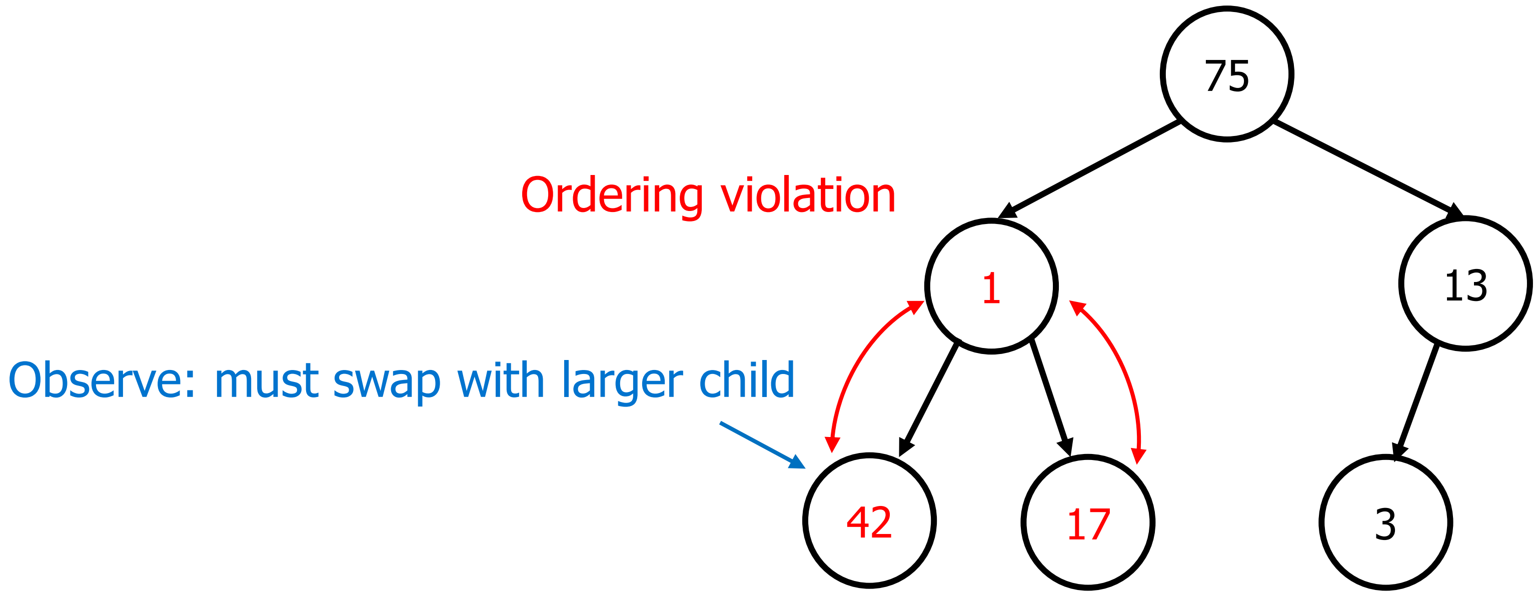


Heap Deletion Example



WATERLOO | ENGINEERING

Heap Deletion Example



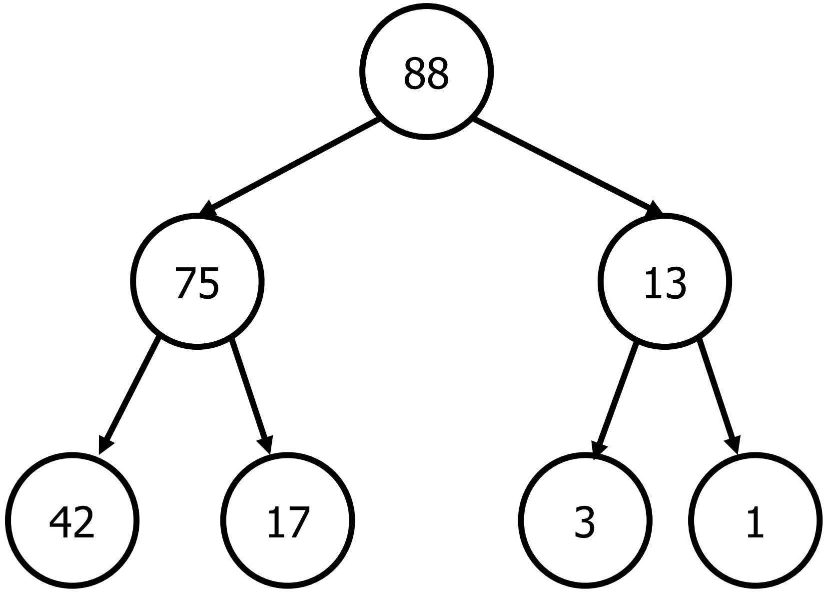
Done

Array Representation of Heaps

- Heaps: conceptually trees, implemented in arrays
  - No need for explicit pointers, just use math
  - Need indexing scheme locate an item’s parent and children
    - Scheme 1: Root at index 0
    - Scheme 2: Root at index 1

Array Representation of Heaps

Level order traversal



13 → i = 2

Parent at index (2-1)/2 = 0 (integer division)

Left child at index (2*2+1) = 5

Right child at index (2*2+2) = 6

Scheme 1

88	75	13	42	17	3	1	
0	1	2	3	4	5	6	7

Scheme 2

	88	75	13	42	17	3	1

Each level has twice as many nodes as the previous level

	Root at 0	Root at 1
Parent	$(i-1)/2$	$i/2$
Left child	$2i+1$	$2i$
Right child	$2i+2$	$2i+1$

Array Representation of Heaps

- Why would you want the root at index 1 scheme?

- Allows element 0 to be a sentinel
  - Stops bubbling up without an explicit “if”
  - Min-heap → smallest possible item of a given type
  - Max-heap → largest possible item of a given type
  - No need to check if the inserted node has been bubbled up to root

- Also, makes the math to find a node’s parent’s index easy
  - $i/2$ vs $(i-1)/2$
  - Same for left child ($2i$ vs $2i + 1$)

Array-based Heap Definition

```
class Heap {                                    // Can be templated
 private:
        int * data;                             // The underlying array
        int array_size;                         // Useful for checking full & growing size
        int last_element;                       // Useful for insertion
        void bubbleUp (int index);
        void bubbleDown (int index);
 public:
        void insert (int item)
        int remove()                            // Always remove root
};
```

Heap Insertion Implementation

```
void insert (int item) {
```

Try it yourself first!

```
}
```

Heap Insertion Implementation

```
void insert (int item) {
        last_element++;
        if (last _element == array_size) { // Resize as needed
                array_size = array_size * 2;
                data = realloc (array_size * sizeof(*data));
        }
        data[last_element] = item           // Put in next available spot
        bubblUp(last_element);              // BubbleUp
}
```

Bubble up: Without Sentinel

```
void bubbleUp (int index) {                    // it's recursive!
    if (index == 0) {                          // check that we aren't at the top already
        return;
    }
    int parent = (index – 1) / 2;              // compute parent index
    if (data[parent] < data[index]) {
        int temp = data[parent];
        data[parent] = data[index]             // swap parent and index
        data[index] = temp;
        bubbleUp(parent);                      // bubbleUp parent
    }
    return;
}
```

Bubble up: With Sentinel

```
void bubbleUp (int index) {
        if (index == 1) {                    // Using sentinel? Don't need this check!
                return;
        }
        int parent = index / 2;              // compute parent index
        if (data[parent] < data[index]) {
                int temp = data[parent];
                data[parent] = data[index]   // swap parent and index
                data[index] = temp;
                bubbleUp(parent);            // bubbleUp parent
        }
        return;
}
```

Heap Deletion Implementation

```
int remove () {
```

Try it yourself first!

```
}
```

Heap Deletion Implementation

```
int remove () {
        int ans = data[1];                      // root at data[1] (data[0] is sentinel)
        data[1] = data[last_element];           // replace with last element
        last_element--;                         // implicitly remove by decreasing last index
        bubbleDown(1);                          // bubble down to fix ordering
        return ans;
}
```

Heap Deletion Implementation

```
void bubbleDown (int index) {
        int left = 2 * index;                       // calculate left & right children's indices
        int right = 2 * index + 1;
        if (left > last_element) {                  // does not have any children?
                return;
        }
        if (left == last_element) {                 // only has left child?
                if (data[left] > data[index]) { swap(data[left], data[index]);}
        } else {
                int maxidx = data[left] > data[right] ? left : right;     // determine which child is larger
                swap(maxidx, index);
                bubbleDown(maxidx);
        }
        return
}
```

Generality

- Can make templated PriorityQueue/Heap…
- Design choices and considerations
  - PQ of Ts
    - enqueue(T): use overloaded < on Ts to order
    - enqueue(T, int): order by ints, T does not need <
  - Priority is not an int?
    - Ints are finite for computers:
      - INT_MAX = largest signed int
      - UINT_MAX = largest unsigned int
    - Strings are comparable, but not finite
      - Think you have the largest string?
      - Add one letter to the end

New Operations? New Heaps

- We did “binary heap” w.r.t. Priority Queue ADT
  - Bare minimum functionalities: add, remove max/min …

- Other operations we might want
  - Union: fuse two heaps of the same type together
    - Combining data sets for analysis
    - Algorithms that intentionally stratify data first, then combine later
  - Increase priority: boost an item’s priority level
    - Increase Patients’ priority based on their current conditions

- Might need fancier heaps (not binary)
  - Binomial heaps, fibonacci heaps
  - Maintain “forests”: multiple tree-type data structures

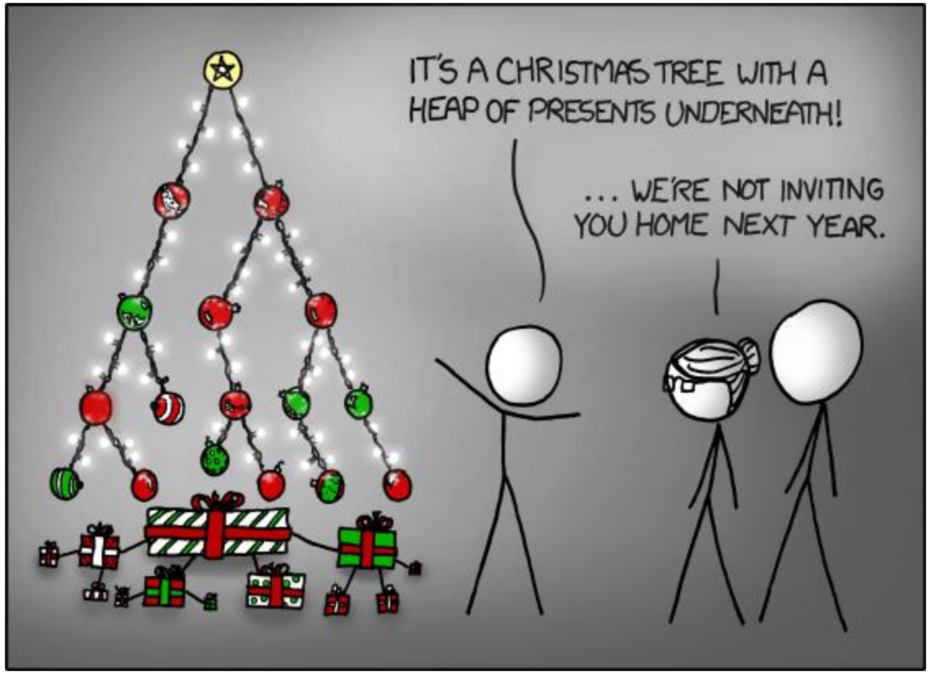
Wrap Up

- In this lecture we talked about
  - Concept of Heaps: binary tree with different rules/ordering
    - All nodes larger (or smaller) than their parents
    - Tree kept completely full and balanced
  - Efficient for Priority Queues
  - Basic operations: insert, delete max(min)
  - Array representations of heaps

- Next up
  - Application of Priority Queues: Huffman coding

Suggested Complimentary Readings

- Data Structure and Algorithms in C++: Chapter 6.1-6.4
- Introduction to Algorithms: Chapter 6.1-6.3, 6.5



Acknowledgement

- This slide builds on the hard work of the following amazing instructors:
  - Andrew Hilton (Duke)
  - Mary Hudachek-Buswell (Gatech)