**03-0: Heap Definition**

- Complete Binary Tree
- Heap Property
  - Max Heap:
    - For every subtree in a tree, each value in the subtree is \( \geq \) value stored at the root of the subtree
  - Min Heap:
    - For every subtree in a tree, each value in the subtree is \( \leq \) value stored at the root of the subtree

**03-1: Heap Examples**

```
20
8   15
7   4   14
5   2   1   3
```

Valid Heap

```
20
8   15
7   9   4   14
5   2   1   3
```

Invalid Heap

**03-2: Heap Examples**

```
20
8   15
7   4   14
5   2   1   3
```

**03-3: Heap Insert**

- There is only one place we can insert an element into a heap, so that the heap remains a complete binary tree
- Inserting an element at the “end” of the heap might break the heap property
03-4: **Heap Insert**

- There is only one place we can insert an element into a heap, so that the heap remains a complete binary tree
- Inserting an element at the “end” of the heap might break the heap property
  - Swap the inserted value up the tree

03-5: **Heap Remove Largest**

- Removing the Root of the heap is hard
- Removing the element at the “end” of the heap is easy

03-6: **Heap Remove Largest**

- Removing the Root of the heap is hard
- Removing the element at the “end” of the heap is easy
  - Move last element into root
    - May break the heap property

03-7: **Heap Remove Largest**

- Removing the Root of the heap is hard
- Removing the element at the “end” of the heap is easy
  - Move last element into root
    - Shift the root down, until heap property is satisfied

03-8: **Representing Heaps**

- Represent heaps using pointers
  - Need to add parent pointers for insert to work correctly
  - Space needed to store pointers – 3 per node – could be greater than the space need to store the data in the heap!
  - Memory allocation and deallocation is slow
- There is a better way!

03-9: **Representing Heaps**

A Complete Binary Tree can be stored in an array:
03-10: **CBTs as Arrays**

- The root is stored at index 0
- For the node stored at index $i$:
  - Left child is stored at index $2 \times i + 1$
  - Right child is stored at index $2 \times i + 2$
  - Parent is stored at index $\lfloor (i - 1)/2 \rfloor$

03-11: **CBTs as Arrays**

Finding the parent of a node

```java
int parent(int n) {
    return (n - 1) / 2;
}
```

Finding the left child of a node

```java
int leftchild(int n) {
    return 2 * n + 1;
}
```

Finding the right child of a node

```java
int rightchild(int n) {
    return 2 * n + 1;
}
```

03-12: **Building a Heap**

Build a heap out of $n$ elements

03-13: **Building a Heap**

Build a heap out of $n$ elements
• Start with an empty heap
• Do \( n \) insertions into the heap

```java
MaxHeap H = new MaxHeap();
for(i=0 < i<A.size(); i++)
    H.insert(A[i]);
```

Running time?

03-14: **Building a Heap**
Build a heap out of \( n \) elements

• Start with an empty heap
• Do \( n \) insertions into the heap

```java
MaxHeap H = new MaxHeap();
for(i=0 < i<A.size(); i++)
    H.insert(A[i]);
```

Running time? \( O(n \log n) \) – is this bound tight?

03-15: **Building a Heap** Total time: \( c_1 + \sum_{i=1}^{n} c_2 \log i \)

03-16: **Building a Heap** Total time: \( c_1 + \sum_{i=1}^{n} c_2 \log i \)

\[
c_1 + \sum_{i=1}^{n} c_2 \log i \geq \sum_{i=n/2}^{n} c_2 \log i
\geq \sum_{i=n/2}^{n} c_2 \log(n/2)
= (n/2)c_2 \log(n/2)
= (n/2)c_2((\log n) - 1)
\in \Omega(n \log n)
\]

Running Time: \( \Theta(n \log n) \)

03-17: **Building a Heap**
Build a heap from the bottom up

• Place elements into a heap array
• Each leaf is a legal heap
• First potential problem is at location \( \left\lfloor i/2 \right\rfloor \)

03-18: **Building a Heap**
Build a heap from the bottom up

• Place elements into a heap array
• Each leaf is a legal heap
• First potential problem is at location \( \left\lfloor i/2 \right\rfloor \)
for(i=n/2; i>=0; i--)
siftdown(i);

03-19: Building a Heap

How many swaps, worst case? If every sift down has to swap all the way to a leaf:

- $n/4$ elements 1 swap
- $n/8$ elements 2 swaps
- $n/16$ elements 3 swaps
- $n/32$ elements 4 swaps

... Total # of swaps:

$$n/4 + 2n/8 + 3n/16 + 4n/32 + \ldots + (\lg n)n/n$$

03-20: Heapsort

- How can we use a heap to sort a list?

03-21: Heapsort

- How can we use a heap to sort a list?
  - Build a max-heap out of the array we want to sort (Time $\Theta(n)$)
  - While the heap is not empty:
    - Remove the largest element
    - Place this element in the “empty space” just cleared by the deletion

Total time:

03-22: Heapsort

- How can we use a heap to sort a list?
  - Build a max-heap out of the array we want to sort (Time $\Theta(n)$)
  - While the heap is not empty:
    - Remove the largest element
    - Place this element in the “empty space” just cleared by the deletion

Total time: $\Theta(n \lg n)$