Graduate Algorithms
CS673-2016F-03
Heaps

David Galles
Department of Computer Science
University of San Francisco
03-0: Heap Definition

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

Valid Heap
03-2: Heap Examples

Invalid Heap
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.
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
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
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!
A Complete Binary Tree can be stored in an array:
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 $\left\lfloor \frac{(i - 1)}{2} \right\rfloor$
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
Building a Heap

Build a heap out of $n$ elements

- Start with an empty heap
- Do $n$ insertions into the heap

MaxHeap $H = \text{new MaxHeap}();$
for($i=0 < i<A.\text{size}(); i++)
    $H.\text{insert}(A[i]);$

Running time?
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? \( O(n \lg n) \) – is this bound tight?
Total time: \[ c_1 + \sum_{i=1}^{n} c_2 \log i \]
Total time: \( c_1 + \sum_{i=1}^{n} c_2 \lg i \)

\[
c_1 + \sum_{i=1}^{n} c_2 \lg i \geq \sum_{i=n/2}^{n} c_2 \lg i
\]

\[
\geq \sum_{i=n/2}^{n} c_2 \lg(n/2)
\]

\[
= (n/2)c_2 \lg(n/2)
\]

\[
= (n/2)c_2((\lg n) - 1)
\]

\[
\in \Omega(n \lg n)
\]

Running Time: \( \Omega(n \lg n) \)
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 $\lfloor i/2 \rfloor$
**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 \( \lfloor i/2 \rfloor \)

```plaintext
for(i=n/2; i>=0; i--)
    siftdown(i);
```
How many swaps, worst case? If every siftdown has to swap all the way to a leaf:

\[
\begin{align*}
\frac{n}{4} \text{ elements} & \quad 1 \text{ swap} \\
\frac{n}{8} \text{ elements} & \quad 2 \text{ swaps} \\
\frac{n}{16} \text{ elements} & \quad 3 \text{ swaps} \\
\frac{n}{32} \text{ elements} & \quad 4 \text{ swaps} \\
\ldots & \\
\text{Total # of swaps:}
\end{align*}
\]

\[
\frac{n}{4} + 2\frac{n}{8} + 3\frac{n}{16} + 4\frac{n}{32} + \ldots + (\log n)\frac{n}{n}
\]
How can we use a heap to sort a list?
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:
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: $\Theta(n \ lg \ n)$