Heaps

- A heap can be seen as a complete binary tree:

What makes a binary tree complete?
Is the example above complete?
Heaps

- A heap can be seen as a complete binary tree:
Heaps

- In practice, heaps are usually implemented as arrays:

\[ A = 16 \ 14 \ 10 \ 8 \ 7 \ 9 \ 3 \ 2 \ 4 \ 1 = \]

![Heap Tree Representation]

1. 16
2. 14
   - 14
     - 8
     - 7
   - 10
     - 9
     - 3
3. 2
4. 4
5. 1

A hierarchical structure where each node is greater than or equal to its children, ensuring the heap property is maintained.
Heaps

- To represent a complete binary tree as an array:
  - The root node is $A[1]$
  - Node $i$ is $A[i]$
  - The parent of node $i$ is $A[i/2]$ (note: integer divide)
  - The left child of node $i$ is $A[2i]$
  - The right child of node $i$ is $A[2i+1]$

$$A = [16, 14, 10, 8, 7, 9, 3, 2, 4, 1]$$
Referencing Heap Elements

• So...

Parent(i) { return \lfloor i/2 \rfloor; }
Left(i) { return 2*i; }
right(i) { return 2*i + 1; }

• An aside: How would you implement this most efficiently?
The Heap Property

• Heaps also satisfy the *heap property*:
  \[ A[\text{Parent}(i)] \geq A[i] \quad \text{for all nodes } i > 1 \]
  - In other words, the value of a node is at most the value of its parent
  - *Where is the largest element in a heap stored?*

• Definitions:
  - The *height* of a node in the tree = the number of edges on the longest downward path to a leaf
  - The height of a tree = the height of its root
Heap Height

• What is the height of an n-element heap? Why?

Number of node in full binary tree of height h

\[
2^0 + 2^1 + 2^2 + \ldots + 2^h = 2^{h+1} - 1
\]

\[
2^h \leq n \leq 2^{h+1} - 1
\]

Taking log we get

\[
h \leq \log(n), \log(n + 1) \leq h + 1
\]

\[
\log(n + 1) - 1 \leq h \leq \log(n)
\]

\[
h = \text{floor} \ (\log(n))
\]
Heap Height

- **What is the height of an n-element heap? Why?** $\Theta(\log(n))$

- This is nice: basic heap operations take at most time proportional to the height of the heap
Heap Height

- Heapify
- Build-heap
- Heapsort
Heap Operations: Heapify()

- **Heapify()**: maintain the heap property
  - Given: a node \( i \) in the heap with children \( l \) and \( r \)
  - Given: two subtrees rooted at \( l \) and \( r \), assumed to be heaps
  - Problem: The subtree rooted at \( i \) may violate the heap property (How?)
  - Action: let the value of the parent node “float down” so subtree at \( i \) satisfies the heap property
    - What do you suppose will be the basic operation between \( i, l, \) and \( r \)?
Heap Operations: Heapify()

Heapify(A, i)
{
    l = Left(i); r = Right(i);
    if (l <= heap_size(A) && A[l] > A[i])
        largest = l;
    else
        largest = i;
    if (r <= heap_size(A) && A[r] > A[largest])
        largest = r;
    if (largest != i)
        Swap(A, i, largest);
    Heapify(A, largest);
}

How to maintain heap property.
Suppose property is violated at A[i]
Heapify(A, 2) Example

Assumes that prior to violation of heap property
As node A[2] the array is indeed a heap.
Heapify(A,2) Example

A = 16 4 10 14 7 9 3 2 8 1
Heapify(A,2) Example

A = 16 14 10 4 7 9 3 2 8 1
Heapify(A,2) Example

A = 16 14 10 4 7 9 3 2 8 1
Heapify(A,4) Example

A = [16, 14, 10, 4, 7, 9, 3, 2, 8, 1]
Heapify(A,4) Example

A = 16 14 10 8 7 9 3 2 4 1
Heapify(A, 4) Example

A = [16, 14, 10, 8, 7, 9, 3, 2, 4, 1]
Heapify(A, 9) Example

\[
A = 16 \ 14 \ 10 \ 8 \ 7 \ 9 \ 3 \ 2 \ 4 \ 1
\]
Analyzing Heapify(): Informal

• Aside from the recursive call, what is the running time of \texttt{Heapify()}?

• How many times can \texttt{Heapify()} recursively call itself?

• What is the worst-case running time of \texttt{Heapify()} on a heap of size \( n \)?