Par: ext. 3 days to Tuesday

Note: in a CBT:

\[#\text{nodes at depth } d\] = 2^d

\[\therefore \text{ #nodes in a CBT of height } h \text{ is}\]

\[n = \sum_{d=0}^{h} 2^d = \frac{2^{h+1} - 1}{2 - 1} = 2^{h+1} - 1\]

\[\therefore 2^{h+1} = n + 1\]

\[\therefore h+1 = \log(n+1)\]

\[\therefore h = \log(n+1) - 1\]
An Almost Complete Binary Tree (ACBT) is a Binary Tree that is filled at all levels except possibly the bottom, which is filled left to right.

Example:

True # of nodes n in an ACBT of height h satisfies:
\[
2^h - 1 < n \leq 2^{h+1} - 1
\]

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

\[
h \leq \lfloor \log(n) \rfloor < h+1
\]

\[
h = \lfloor \log(n) \rfloor
\]
6.1 Heaps

A Binary Heap is an array used to represent a CBT. Each node corresponds to an array element.

Ex.

```
  25
  /\  \\
 2 /  3
  \  /
 17 20
 /\ /\  /
8 11 8 12
 /\ /\  /
5 10 6 15
```

```
A = [25, 17, 20, 11, 16, 15, 12, 5, 8, 10, 2, 6, 
     8, 9, 12, 11, 10, 11, 12, 15, 14, 15, 16]
```

length(A) = 16, heapSize(A) = 12
Array has 2 attributes

- length[A]
- headSize[A]


Helper functions:

\[
\text{parent}(i) = \left\lfloor \frac{i}{2} \right\rfloor \quad (i \geq 2)
\]

\[
\text{left}(i) = 2i
\]

\[
\text{right}(i) = 2i + 1
\]

* max - heap property: \( A[\text{parent}(i)] \geq A[i] \)

* min - heap property: \( A[\text{parent}(i)] \leq A[i] \)
Heap operations

- HeaAdity ( )
- BuildHeap ( )
- Heapsort ( )

Priority Queue ops (ADT)
  - Insert ( )
    - ExtractMax
    - IncreaseKey
    - Maximum
6.2 Heapsify

Pre:

Goal of Heapsify(A, i) is to make the subtree rooted at i into a valid heap.
\textbf{Heapify}(A, i) \quad \text{Pre: subtrees at left(i) and right(i) are valid heaps}

1.) \( l = \text{left}(i) \)
2.) \( r = \text{right}(i) \)
3.) if \( l \leq \text{heapSize}[A] \) and \( A[l] > A[i] \)
4.) \( \text{largest} = l \)
5.) \text{else}
6.) \( \text{largest} = i \)
7.) if \( r \leq \text{heapSize}[A] \) and \( A[r] > A[\text{largest}] \)
8.) \( \text{largest} = r \)
9.) if \( \text{largest} \neq i \)
10.) \( A[i] \leftrightarrow A[\text{largest}] \) (swap)
11.) \text{Heapify}(A, \text{largest})

Let \( T(n) \) denote the worst-case run time of \text{Heapify}() when run on a subtree (rooted at \( i \)) consisting of \( n \) nodes. Then

\[ T(n) = \text{height of subtree at } i = \lceil \log n \rceil \]

\[ = \Theta(\log n) \]

\[ = \Theta(h) \]
6.3 Build Heap

Ex

Build Heap \( A \)

1. \( n = \text{heapSize} [A] = \text{length} [A] \)
2. for \( i = \lfloor \frac{n}{2} \rfloor \) down to 1
3. Heapify \( A, i \)

Routine \( \mathcal{O}(n) = \Theta(n) \)
6.4 Heapsort

Heapsort (A)

1.) BuildHeap(A)

2.) for i = length[A] down to 2


4.) heapSize[A] --

5.) Heapify(A, 1)

Runtime: \( T(n) = \Theta(n) + (n-1) \Theta(\log n) \)

\[ = \Theta(n \log n) \]
6.5 Priority Queues

Priority Queue AAT: maintain a finite set \( S \) of elements (records), each with an associated attribute called \( \text{key} \).

\[ x = (\ldots, x, \ldots, \text{key}, \ldots) \in S \]

satellite data

\[ S = \{ x, \ldots, x, \ldots \} \]

This set is in a state
A max priority has following ops.

- **Insert** \( S, x \) : inserts a new \( x \) into \( S \).

- **Max** \( S \) : returns record with largest key.

- **ExtractMax** \( S \) : returns \& deletes record with largest key.

- **IncreaseKey** \( S, x, k \) : change \( key[x] \) to \( k \) if \( k > key[x] \), otherwise do nothing (book considers \( k \leq key[x] \) an error.)
max-heap implementation:

HeapMaximum(A)


Runtime: $\Theta(1)$