Revisiting FindMin

- Application: Find the smallest (or highest priority) item quickly
  - Operating system needs to schedule jobs according to priority instead of FIFO
  - Event simulation (bank customers arriving and departing, ordered according to when the event happened)
  - Find student with highest grade, employee with highest salary etc.

Priority Queue ADT

- Priority Queue can efficiently do:
  - FindMin (and DeleteMin)
  - Insert
- What if we use...
  - Lists: If sorted, what is the run time for Insert and FindMin? Unsorted?
  - Binary Search Trees: What is the run time for Insert and FindMin?

Less flexibility  →  More speed

- Lists
  - If sorted: FindMin is $O(1)$ but Insert is $O(N)$
  - If not sorted: Insert is $O(1)$ but FindMin is $O(N)$
- Balanced Binary Search Trees (BSTs)
  - Insert is $O(\log N)$ and FindMin is $O(\log N)$
- BSTs look good but...
  - BSTs are efficient for all Finds, not just FindMin
  - We only need FindMin
Better than a speeding BST

• We can do better than Balanced Binary Search Trees.
  › Very limited requirements: Insert, FindMin, DeleteMin. The goals are:
  › FindMin is $O(1)$
  › Insert is $O(\log N)$
  › DeleteMin is $O(\log N)$

Binary Heaps (minimum)

• A binary heap is a binary tree (NOT a BST) that is:
  › Complete: the tree is completely filled except possibly the bottom level, which is filled from left to right
  › Satisfies the heap order property:
    • every node is less than or equal to its children
    • In particular, the root node is always the smallest node

Binary Heaps (maximum)

• A binary heap is a binary tree (NOT a BST) that is:
  › Complete: the tree is completely filled except possibly the bottom level, which is filled from left to right
  › Satisfies the heap order property
    • every node is greater than or equal to its children
    • In particular, the root node is always the largest node

Heap order property

• A heap provides limited ordering information
• Each path is sorted, but the subtrees are not sorted relative to each other
  › A binary heap is NOT a binary search tree

These are all valid binary heaps (minimum)
Binary Heap vs Binary Search Tree

Binary Heap

5
10
94
97
24

Parent is less than both left and right children

Binary Search Tree

94
10
97
5
24

Parent is greater than left child, less than right child

Structure property

• A binary heap is a complete tree
  › All nodes are in use except for possibly the right end of the bottom row

Examples

Array Implementation of Heaps

• Root node = A[1]
• Children of A[i] are in A[2i], A[2i + 1]
  › Proof: By induction on i
• Keep track of current size N (number of nodes)
FindMin and DeleteMin

- **FindMin:** Easy!
  - Return root value $A[1]$
  - Run time $=$ ?

- **DeleteMin:**
  - Delete (and return) value at root node
  - How can we delete?

Maintain the Structure Property

- We now have a “Hole” at the root
  - Need to fill the hole with another value
- When we get done, the tree will have one less node and must still be complete

Maintain the Heap Property

- The last value has lost its node
  - we need to find a new place for it
- We can do a simple insertion sort operation to find the correct place for it in the tree

DeleteMin: Percolate Down

- Copy smaller child up and go down one level
- Done if both children are $\geq$ item or reached a leaf node
- What is the run time?
Percolate Down

Maintain the Heap Property

• First, the new value goes to A[N+1] (and N is increased)
• Next, we find the correct place for it in the tree

Insert: Percolate Up

• Start at last node and keep comparing with parent A[i/2]
• If parent larger, copy parent down and go up one level
• Done if parent ≤ item or reached top node A[1]
• Run time?

Insert: Done

• Run time?

PercUp

• Define PercUp which percolates new entry to correct spot.
• Note: the parent of i is i/2

\[
\text{PercUp}(i : \text{integer}, x : \text{integer}): \{ \\
\quad \ldots \\
\}
\]
Sentinel Values

- Every iteration of Insert needs to test:
  - if it has reached the top node A[1]
  - if parent \( \leq \) item
- Can avoid first test if A[0] contains a very large negative value
  - sentinel \(-\infty\) < item, for all items
- Second test alone always stops at top

<table>
<thead>
<tr>
<th>value</th>
<th>-\infty</th>
<th>2</th>
<th>3</th>
<th>8</th>
<th>7</th>
<th>4</th>
<th>10</th>
<th>9</th>
<th>11</th>
<th>9</th>
<th>6</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>index</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
</tr>
</tbody>
</table>

Binary Heap Analysis

- Space needed for heap of at most MaxN nodes: O(MaxN)
  - An array of size MaxN, plus a variable to store the current size N, plus an array slot to hold the sentinel
- Time
  - FindMin: O(1)
  - DeleteMin and Insert: O(log N)
  - BuildHeap from N inputs : O(N)

Build Heap

```plaintext
BuildHeap {
    for i = N/2 to 1 by -1 PercDown(i, A[i])
}
```

Build Heap

Build Heap { N=11

```
5
/   \\n3   10
/ \   / \ \
2  8  9  4
```

\[ \begin{array}{c}
\text{Build Heap} \\
\text{with } N=11 \\
\text{results in:}
\end{array} \]
Build Heap

Analysis of Build Heap

- Assume $N = 2^k - 1$ (a full tree of height $k$)
  - Level 1: $k - 1$ steps for 1 item
  - Level 2: $k - 2$ steps for 2 items
  - Level 3: $k - 3$ steps for 4 items
  - In general: Level $i$: $k - i$ steps for $2^{i-1}$ items
  - Until Level $k - 1$: 1 step for $2^{k-2}$ items

Total Steps: $\sum_{i=1}^{k-1} (k - i)2^{i-1} = 2^k - k - 1$

$= O(N)$

Other Heap Operations

- Find($X, H$): Find the element $X$ in heap $H$ of $N$ elements
  - What is the running time? $O(N)$
- FindMax($H$): Find the maximum element in $H$
- Where FindMin is $O(1)$
  - What is the running time? $O(N)$
- We sacrificed performance of these operations in order to get $O(1)$ performance for FindMin
- How can we support FindMax in $O(1)$?
  - Hint: double time and space complexity..

Other Heap Operations

- DecreaseKey($P, \Delta$): Decrease the key value of node at position $P$ by a positive amount $\Delta$, e.g., to increase priority
  - First, subtract $\Delta$ from current value at $P$
  - Heap order property may be violated
  - so percolate up to fix
  - Running Time: $O(\log N)$
Other Heap Operations

- **IncreaseKey**(P, Δ): Increase the key value of node at position P by a positive amount Δ, e.g., to decrease priority
  - First, add Δ to current value at P
  - Heap order property may be violated
  - so percolate down to fix
  - Running Time: O(log N)

- **Delete**(P): E.g. Delete a job waiting in queue that has been preemptively terminated by user
  - Use **DecreaseKey**(P, ∞) followed by **DeleteMin**
  - Running Time: O(log N)

Other Heap Operations

- **Merge**(H1, H2): Merge two heaps H1 and H2 of size O(N). H1 and H2 are stored in two arrays.
  - Can do O(N) Insert operations: O(N log N) time
  - Better: Copy H2 at the end of H1 and use **BuildHeap**. Running Time: O(N)

PercUp Solution

```java
PercUp(i : integer, x : integer): {
    if i = 1 then A[1] := x
    else if A[i/2] < x then
        A[i] := x;
        else
            A[i] := A[i/2];
            Percup(i/2, x);

```