Cosc-320
Heap Operations

Introduction

These notes on heaps expound on Chapter 14 of “Data Structures with C++ using STL 2nd Edition,” William Ford and William Topp, Prentice-Hall, 2002.)

◇ **Definition**: A binary max-heap is a complete binary tree in which, *at every node*, the value at the node is no less than the value at either child

◇ **Definition**: A binary min-heap is a complete binary tree in which, *at every node*, the value at the node is no greater than the value at either child

• Note: The text calls these *max-heap* and *min-heap*, leaving off the “binary” part.

• The usual implementation of a heap is as an array or vector. Any array or vector can represent a complete binary tree. The values are taken in index-order and put in the tree in level-order.

The array `{15, 10, 3, 4, 7, 2}` represents the complete binary tree (and heap)

```
        15
       /   \
     10     3
    /  \   /
   4    7  2
```

Given the index, i, of an element in a vector, the parent of that element has index $(i-1)/2$, its left child has index $2i+1$, and its right child has index $2i+2$. Thus, in the vector above, the node with value 10 is at index 1. Its parent is at $(1-1)/2 = 0$, its left child is at $2 \times 1 + 1 = 3$ and its right child is at $2 \times 1 + 2 = 4$.

The Operations

The operations on a heap and their worst-case asymptotic performance as a function of the number of elements, n is:

push	$O(\lg n)$
pop	$O(\lg n)$
top	$O(1)$
Construct from random vector	$O(n)$

top

The `top` operation just requires finding the element at index 0 of the vector. This is clearly in $O(1)$.

push

The `push` operation inserts a new element into the heap. The operation is implemented in two stages:

1. Insert the new element at the back of the vector (at index n). This is in $O(1)$, worst case.
2. Move the element up the tree ("percolate-up" or "heapify") by iteratively swapping it with its parent until it is in the correct heap location (no bigger than its parent for a max-heap). `push` is in $O(\lg n)$, worst case, since the maximum distance the new element might move is from the lowest level to the root level. Since this is a complete binary tree, that distance is in $O(\lg n)$.

Thus, the `push` operation is in $O(1 + \lg n) = O(\lg n)$.

pop

The `pop` operation removes the root element from the tree. It is done in two stages:

1. Exchange the root value with the value at index $n-1$ (the rightmost node on the lowest level). This is in $O(1)$, worst case.
2. "Re-heapify" (or "sift-down") the tree by moving the new root value down until it is in the correct heap location (no smaller than either child for a max-heap). When moving it down, swap it with the larger of its two children. As with `push`, the new root might move from the root level to the lowest level, a distance in $O(\lg n)$.

Thus, the `pop` operation is in $O(1 + \lg n) = O(\lg n)$.

Constructing a Heap from Scratch

Given an arbitrary vector (values not in heap order), the vector can be made into a heap in $O(n)$, worst case. Iteratively visit each node from the one at index $(n-2)/2$ up to the root node with index 0. In each iteration, move the value at that index down the tree until it is in its correct heap location. The text calls each such operation "adjustHeap." It's the same operation used in `pop` to move the root down to its correct location.

An arbitrary vector can be put into heap order in $O(n)$ steps using the procedure above. A complete binary tree of height h has 2^{h-1} nodes at level $h-1$ (the level above the lowest).

In the procedure above, there are at most:

1 swap per node at level $h-1$ (just swap with child below),

2 swaps per node at level $h-2$ (swap with child and grandchild),

and so forth, down to

h swaps of the root at level 0.

Thus, the total number of swaps, worst case, is

$$0 \times 2^h + 1 \times 2^{h-1} + 2 \times 2^{h-2} + \cdots + h \times 2^0 = \sum_{i=0}^{h} 2^i (h-i)$$

$$\sum_{i=0}^{h} 2^i (h-i) = h \sum_{i=0}^{h} 2^i - \sum_{i=0}^{h} i2^i = h(2^{h+1}-1) - ((h-1)2^{h+1}+2) = 2^{h+1} - h - 2$$

The expression $2^{h+1} - h - 2$ is in $O(2^h)$. Note that a complete binary tree of height h has $2^{h+1} - 1$ nodes, which is also in $O(2^h)$. Thus, the worst-case number of swaps is on the same order as the number of nodes in the tree.