Heaps

Introduction

Some heap facts:

- A heap is a complete binary tree.
- For a max-heap, a parent’s value is always greater than or equal to the values of its children.
- For a min-heap, a parent’s value is always less than or equal to the values of its children.
- There is no relationship between the values of sibling nodes.

We will assume a max-heap unless otherwise stated.

Problems:

- 7.1-1
- 7.1-2
- 7.1-4 (all elements distinct)
- 7.1-5
- 7.1-6

Correcting point errors in a heap

Often in the course of manipulating a heap, heap ordering is disrupted. Typically, these are point errors. That is to say, only one value in the heap is incorrect; absent that error, the heap has the correct ordering.

There are two types of point errors that arise during some heap operations. One is that a parent value is modified and turns out to be too small. Since it is a point error, we know that the modified value needs to move lower down in the heap. The other is that a node gains a child that has a value that is larger than it should be. In this case, the child value needs to move up.

To remedy the first case, the heapify or pushDown operation for max-heaps pushes a small value down the heap until it resides at its proper level (mutatis mutandis for min-heaps). The sub-heap rooted at the original node will be a heap after this operation completes:

```javascript
function heapify(i) { // i is the ith node
    var next = i;
    if (hasLeft(i) && value(left(i)) > value(next))
        next = left(i);
    if (hasRight(i) && value(right(i)) > value(next))
        next = right(i);
    if (next != i)
        {
            swapValues(i, next);
            heapify(next);
        }
}
```
To correct the second case, the fixHeap or bubbleUp operation for max-heaps moves a large value up the heap until it resides at its proper level (mutatis mutandis for min-heaps).

```plaintext
function fixHeap(i)
{
    if (hasParent(i) && value(i) > value(parent(i)))
    {
        swapValues(i, parent(i));
        fixHeap(parent(i));
    }
}
```

Problems:
• 7.2-1
• 7.2-3
• 7.2-4

**Constructing heaps**

The buildHeap operation is:
• a routine which turns an unordered complete tree into a max-heap
• works from the bottom up

```plaintext
function buildHeap(h)
{
    for each node n in h from rightmost leaf to root by level
    heapify(n)
}
```

Some questions:
• How would you traverse an array-based heap?
• How would you traverse a tree-based heap?
• Can you ignore the leaves?
• The obvious running time for buildHeap is $O(n \log n)$. Is this bound tight?

Problems:
• 7.3-1
• 7.3-3