Homework 5 Solutions

Question 1

Insert the n elements into an AVL tree. Each insertion takes $O(\log n)$ time, and there are n insertions, so the total time for this step is $O(n \log n)$.

Now read-out the elements using an inorder traversal, which takes $O(n)$ time.

The total time is $O(n \log n) + O(n) = O(n \log n)$.

Question 2

The modified red-black tree satisfies all the properties of a red-black tree *except* the root is red. Therefore, it satisfies:

1. Every external node is black.
2. The children of a red node are black.
3. All external nodes have the same black depth, which is the number of black ancestors, minus one. We subtract one because the external node itself should not be included in the depth.

Now, suppose we have a valid modified red-black tree with red root and change its root to black. Property (1) still holds since we have not

changed the color of any external nodes. Property (2) still holds since we have not introduced any new red nodes. Changing the root to black has changed the black depth of every external node (it has gone up by one), but since the black depth for *every* external node has gone up by one, they all still have the same black depth, and property (3) is satisfied.

Finally, since we have changed the root to black, the one remaining property of a “normal” red-black tree is satisfied, and we conclude that the tree is a valid red-black tree.

Question 3

The smallest red-black tree with black height k would have no red nodes since red nodes increase the size without increasing the black height. Since the black depth is the same for each exernal node, it must be that the tree is perfect. A perfect tree of height k has $2^{k+1} - 1$ nodes.

You can prove this by induction on the height k starting with the base case $k = 0$, which would be a tree with a single node and therefore with size $1 = 2^{0+1} - 1$.

The largest red-black tree will have as many red nodes as possible, which would be every other node. This doubles the height since for every black node on a path from the root to an external node there will

be one red node for each black node, not including the external node itself. Moreover, the tree will be perfect since, otherwise, it would not be the largest tree possible. Therefore, the size of the largest tree is $2^{2k+1} - 1$.

This, too, can be proven by induction.

Question 4

Consider a heap of height h. Since a heap is complete, the smallest it can be is a perfect tree of height h - 1 plus one child with depth h. Therefore, the smallest it can be is $2^{h-1+1} - 1 + 1 = 2^h$. Therefore, $2^h <= n$ and taking logs we have:

```
h <= log n
```

Since h is an integer, it follows that h <= floor(log n).

Question 5

The smallest element must be an external node; otherwise it would have a child with larger values than itself. The element must be in the last half of the array since the last internal node is at index floor(n/2). Why is that? First, let i = floor(n/2); then the left child of i is at index 2 * floor(n/2) <= n, which is a valid index in the array, so the node at index i is internal.

Now suppose i = floor(n/2) + 1; then 2 * { floor(n/2) + 1 } = 2 * floor(n/2) + 2. If n is even, then this is n + 2, which is beyond the end of the array. If n is odd, then this is jsut 2 * floor((n –1)/2) + 2 = n - 1 + 2 = n + 1, which is also beyond the end of the array. In either case, the node at index i must be external as it has no children. Therefore, the external nodes are all in indices floor(n/2) + 1 to n.

Question 6

(1) The representation of a d-ary heap in an array is essentially the same as for a binary heap: the elements of the tree are written in order, starting with the root, then the depth-one elements (left-to-right), then the depth-two elements, etc.

(2) The kth child, k = 1, 2, ..., d, of node i in a d-ary heap has index:

```
child(k, i) = d * (i - 1) + k + 1
```

You can prove this by induction:

Base Case: When i = 1 (first element in the heap), we have

```
child(k, 1) = k + 1
```

That is, the children of i are the d values starting at position 2, which is correct for a heap.

Now, suppose the formula is correct for indices 1..*i*; show it is true for *i* + 1.

```
child(k, i + 1) = d * (i + 1 - 1) + k + 1
    = d * (i - 1) + k + 1 + d
    = child(k, i) + d by the inductive hypothesis.
```

But this just says that the *k*th child of *i* + 1 is *d* positions after the *k*th child of *i*, which is correct for a *d*-ary heap.

We have to invert the expression for child(*k*, *i*) to get the formula for parent(*i*). Here's one way to think about that:

- Root is at index 1
- Children of root are at indices 2 … 2 + (*d*–1)
- Next block of children are at 2 + (*d*–1) + 1 = 2 + *d* … 2 + *d* + (*d*–1) = 2*d* + 1.
- Next block: 2*d* + 2 … 2*d* + 2 + (d–1) = 3*d* + 1
- Next block: 3*d* + 2 … 4*d* + 1

You can see the pattern: the children of index *i* are at indices

```
(i-1)d + 2 ... i d + 1
```

We need a formula that takes any index in this range and returns *i*. If we subtract two from any number in this range, we get

```
(i-1)d ... i d - 1
```

Dividing by *d* and taking the floor gives *i* - 1. Therfore, the parent of index *c* is

```
parent(c) = floor( (c - 2) / d ) + 1
```

(3) The number of nodes in a perfect *d*-ary tree of height *h* is

```
1 + d + d^2 + ... + d^h
```

Now we can use essentially the same approach as in Problem 3. The smallest the tree could be is a perfect tree of depth *k* - 1 plus a single node at depth *k*, so

```
( 1 + d + d^2 + ... + d^{k-1} ) + 1 = 2 + d + d^2 + ... + d^{k-1}
<= n
```

It follows that

```
log_d (d^{k-1}) <= log_d n

(k - 1) <= log_d n

k <= 1 + log_d n
```

Or in asymptotic notation, the height is O(log *n*).