Tim Cheeseman
CS 260-003
Summer 2014
Assignment 4 Sample Solution

1)

The maximum height of a binary tree is n - 1, in the case where all nodes lie along a single path with n nodes (length n - 1) from the root node to a single leaf node.

The minimum height of a binary tree occurs when each level is filled such that the depth of any leaf node is no more than one level deeper than all other leaf nodes. For each height h, there will be at most 2^h nodes (a single node at the root has height 0, and at most $2^0 = 1$ nodes, a full tree with height 2 has $2^0+2^1 = 3$ nodes, etc.), and a full tree of height h will have $n = \sum_{i=0}^{h} 2^i = 2^{h+1} - 1$ nodes. Therefore, if we solve for h, a full tree with n nodes will have a height of $log_2(n+1) - 1$. To account for the possibility of non-full trees, we must round up to account for the partially filled level: $\lceil log_2(n+1) - 1 \rceil$.

2)

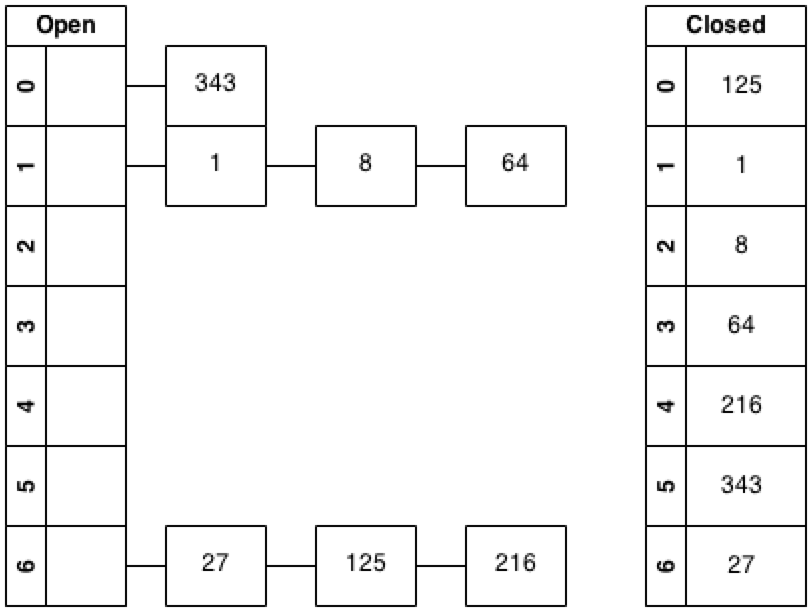
The maximum height of a tree is again n - 1, in the case where all nodes lie along a single path with n nodes (length n - 1) from the root node to a single leaf node.

The minimum height of a tree with n nodes is 1, where there is a root node with n - 1 children.

3)

	Sorted List	**Open Hash Table**[1]
MAKENULL	O(1)	[2] O(1)
UNION	O(n+m)	[3] $O((n^2+m^2)/B)$
INTERSECTION	[4] O(min(n, m))	[5] $O((n \cdot m)/B)$
MEMBER	[6] $O(\log_2(n))$	[7] O(n/B)
MIN	O(1)	O(n)
INSERT	[8] O(n)	[9] O(n/B)
DELETE	[10] O(n)	[11] O(n/B)

4)



[1] Assume B buckets.
[2] If you count initializing each bucket to a null list header, then O(B).
[3] n+m calls to INSERT.
[4] We stop when we reach the end of either list.
[5] For each element in one table, check MEMBER with other table, then INSERT if true
[6] Assuming sorted array (O(1) random access) and binary search. O(n) without (e.g. linked list).
[7] Assuming a good hashing algorithm. Absolute worst case is O(n).
[8] Even in a sorted array with log(n) time to find the element, the shifting after insert takes O(n) worst case.
[9] Assuming a good hashing algorithm. Absolute worst case is O(n).
[10] Even in a sorted array with log(n) time to find the element, the shifting after delete takes O(n) worst case.
[11] Assuming a good hashing algorithm. Absolute worst case is O(n).

5)

- h1(x) is not good because, with a hash table with B buckets, there might be a string whose length is greater than B. Also, we can’t assume that string lengths are evenly distributed so we’d have poorly distributed hash values.
- h2(x) is not good because it is non-deterministic. The same value would hash to a potentially different hash each time, making all operations that would be O(n/B) take O(n) time, as we’d have to check every element for MEMBER, INSERT, etc.

6)

A data structure that fits these requirements is the bit-vector implementation of sets described in section 4.3. If we represent a set as an array of length n of boolean values, then we can insert or delete from the set in O(1) time by just accessing that index in the array and setting the value to true or false, respectively.