Computer Science COMP-250 Homework #5 *v*2.0
Due Sunday April 16th, 2016

1. **Linear-time sorting**

[20%] Consider a situation where you are given an array A of n positive integers, each of these integers $A[i]$ being bounded by a polynomial n^k. Show that Radix sort using k rounds, combined with Counting sort, will allow you to sort these numbers in time $O(n)$.

2. **Min-Max**

Consider another situation where you have to operate on a large set S of (comparable) values. You have to handle three types of operations on S :

[20%]

S.min(): returns and removes the min in S
S.max(): returns and removes the max in S
S.insert(x): inserts an element x in S

Show how to implement the set S so that any of these operations can be performed in time $O(\log n)$, where n is the current number of elements in S.

3. **Radix trees**

Given two strings $a = a_0a_1...a_p$ and $b = b_0b_1...b_q$, where each a_i and each b_j is in some ordered set of characters, we say that string a is lexicographically less than string b if either

A. there exists an integer j, $0 \leq j \leq \min(p, q)$, such that $a_i = b_i$ for all $i = 0, 1, \ldots, j - 1$ and $a_j < b_j$, or

B. $p<q$ and $a_i = b_i$ for all $i = 0, 1, \ldots, p$.

For example, if a and b are bit strings, then 10100 < 10110 by rule 1 (setting j = 3) and 10100 < 101000 by rule 2. This is similar to the ordering used in English-language dictionaries.

The radix tree data structure shown in Figure 1 stores the bit strings 1011, 10, 011, 100, and 0. When searching for a key $a = a_0a_1 \ldots a_p$, we go left at a node of depth i if $a_i = 0$ and right if $a_i = 1$. Let S be a set of distinct binary strings whose lengths sum to n.

[20%] Show how to use a radix tree to sort S lexicographically in $O(n)$ time. For the example in Figure 1, the output of the sort should be the sequence 0, 011, 10, 100, 1011.

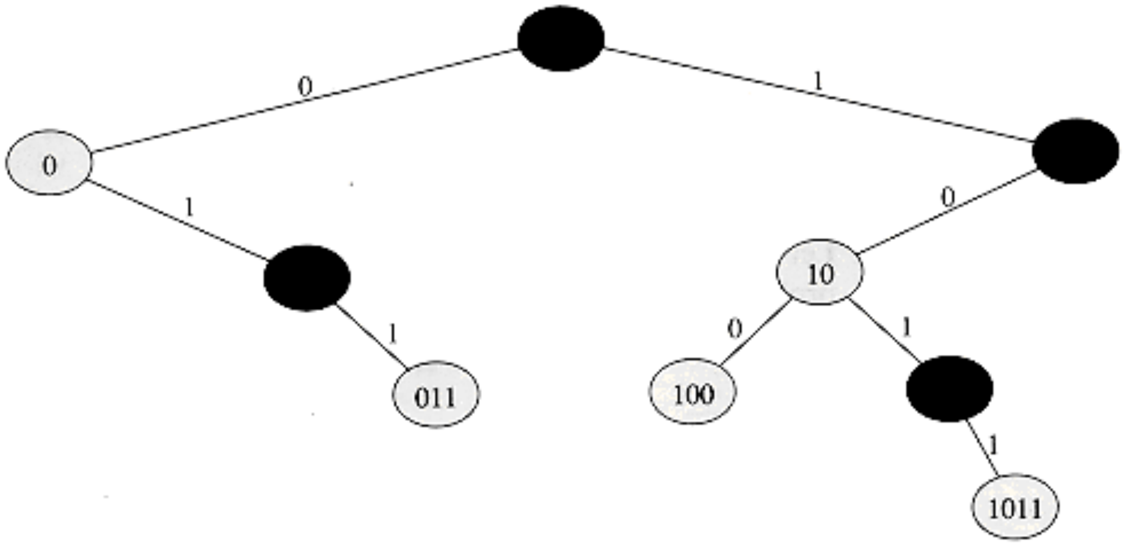

Figure 1 A radix tree storing the bit strings 1011, 10, 011, 100, and 0. Each node's key can be determined by traversing the path from the root to that node. There is no need, therefore, to store the keys in the nodes; the keys are shown here for illustrative purposes only. Nodes are heavily shaded if the keys corresponding to them are not in the tree; such nodes are present only to establish a path to other nodes.

4. Let T_n be the number of distinct ways we can organize a binary tree with n internal nodes. It is easy to observe $T_0 = T_1 = 1$. Argue that for n≥1 we have

[20%]

$$T_{n+1} = \sum_{k=0}^{n} T_k T_{n-k} .$$

Show by mathematical induction that T_n is $\Omega(2^n)$. Indeed T_n is $\Theta(4^n/n^{3/2})$. [for 10 EXTRA points] Show by mathematical induction that T_n is $\Omega(3^n)$.

5. **Professor Bunyan** thinks he has discovered a remarkable property of binary search trees. Suppose that the search for key k in a binary search tree ends up in a leaf. Consider three sets: A, the keys to the left of the search path; B, the keys on the search path; and C, the keys to the right of the search path. Professor Bunyan claims that any three keys $a \in A$, $b \in B$, and $c \in C$ must satisfy $a \leq b \leq c$. Give a smallest possible counterexample to the professor's claim.

[20%]