University of Waterloo
CS 466 — Advanced Algorithm
Spring 2013
Problem Set 3
Siwei Yang - 20258568

1. [8 marks: Using Doubling Binary Search] This question finishes up the proof that the approximation to the optimal binary search tree can be found in linear time if the probabilities of access to keys (and for unsuccessful searches for values between consecutive key values) are given in order by the key values. Having made this introduction we pose the problem in a straightforward mathematical form.

   a. [3 marks] State the recurrence for the time, T(n), for the algorithm outlined below:

      - You take time $\lg i$ to break a problem of size n into 2 subproblems of sizes i-1 and n-i.
      - You have no control over i, other than it is an integer in the range $[1, \frac{n}{2}]$.
      - Problems of size 0 and 1 take no time.

      We can conclude the recurrence series of this algorithm as:

$$T(n) = \begin{cases} T(i-1) + \lg i + T(n-i), & \text{if } n \geq 2 \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

   b. [5 marks] Show that the solution to the recurrence (and so runtime of the algorithm) above is O(n). below:

      Let the base case be:

$$T(n) \leq 2 * n - \lg(n+1) \tag{2}$$

      which is true for n = 1.

      Then, for the induction step, **assuming equation 2 holds for** $n < k$, for a problem of size k, we have:

$$T(i-1) + \lg i \leq 2 * i \tag{3}$$

$$T(i-1) + \lg i + T(k-i) \le 2 * (k-1) - \lg (k-i+1) \tag{4}$$

Since we have i in range [1,$\frac{k}{2}$], we have 2 * (k-i+1) ¿ k + 1. Thus, $\lg (k+1) - \lg (k-i+1) \le 1$. From that fact, part of equation 4 have this:

$$\lg (k-i+1) - 2 = -\lg 2 * (k-i+1) - 1 \le 2 * k - \lg (k+1) - 1 \tag{5}$$

Therefore we can rewrite equation 4 as:

$$T(i-1) + \lg i + T(k-i) \le 2 * k - \lg (k+1) - 1 \tag{6}$$

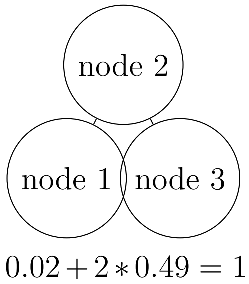
From above, we proved **equation 2 holds for** $n \le k$. Since the base case is true for n = 1, **we have equation 2 true for all positive n which means T(n) is in O(n)**.

2. [4 marks: Near-optimal Binary Search Tree] This question also deals with the approximately optimal binary search tree heuristic. The method we discussed works as follows.

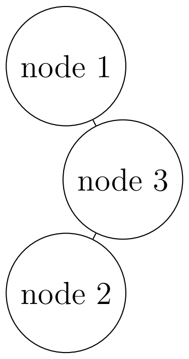
   - Choose the root by a greedy heuristic and recursively solve the problem for the left and right subtrees.
   - The root chosen has the key value such that the maximum of the probability of being in its left subtree and the probability of being in its right subtree is minimized.

   Consider ther following probability distribution:

Interval	q0	p1	q1	p2	q2	p3	q3
Probability	0	0.49	0	0.02	0	0.49	0

   following the heuristic we have a tree like this: 

the amortized cost for accessing this tree is $2 * 0.49 + 0.02 + 2 * 0.49 = 1.98$.



However, the optimal search tree have the following form:
the amortized cost for accessing this tree is $0.49{+}3{*}0.02{+}2{*}0.49 = 1.53$.