TEL AVIV UNIVERSITY
Department of Computer Science
0368.4281 – Advanced topics in algorithms
Spring Semester, 2013/2014

**Homework 1, March 6, 2014**

**Due on Thursday March 20. Please put a hardcopy in my mailbox and keep a copy of the homework!**

1. a) Let $T$ be an arbitrary binary search tree with the items (keys) $B_1, \ldots, B_n$. Define $B_0 = -\infty$ and $B_{n+1} = \infty$. Suppose that we always search items $B \neq B_i$ and the probability that we search an item $B \in (B_i, B_{i+1})$ is $q_i$ for $0 \leq i \leq n$. The expected search time using $T$ is $\sum_{i=0}^{n} q_i a_i$ where $a_i$ is depth of the null pointer (or "dummy" leaf) representing the gap $(B_i, B_{i+1})$. Prove that the expected search time using $T$ is at least $H = \sum q_i \log_2 \frac{1}{q_i}$.
b) We serve a sequence of $m$ searches drawn from the distribution above using a splay tree by splaying at the last node of each search (this is not the "dummy" leaf but the "real" node right before the search fell off the tree.) Prove that the expected search time is $O(m(1+H))$. (Note that this requires a simple modification in the definition of the weights used by the access lemma, and a slight modification in the static optimality theorem that follows from it. So please revisit both the access lemma and the static optimality theorem and describe the changes needed precisely. Then formally prove the statement above.)

2. We insert an item $B$ into a splay tree $T$ ($B \notin T$) by searching for $B$ first, replacing the "dummy" leaf where the search ends by a new node $v$ containing $B$, and splaying at $v$. Use the same potential as in class and prove that the amortized time of adding $B$ is $O(\log n)$ where $n$ is the number of items in the tree when we perform the insertion. (Hint: use the weight function that gives weight 1 to any item.)

3. Show how to extend dynamic trees to support the operation $evert(v)$. This operation changes the actual tree containing $v$ such that following the operation $v$ is the root of this tree. The directions of all the edges from $v$ to the previous root of the tree are reversed. An efficient implementation should run in $O(\log n)$ time. Do not hurt the logarithmic running time of any of the other operations.

4. Assume that each edge $e$ of a dynamic tree has a fixed *weight*, $w(e)$, associated with it, which is given with $e$ when $e$ is inserted by a link operation and never changes (do not confuse $w(e)$ with the cost of $e$, $c(e)$, which is a different thing). Show how to extend dynamic trees to support the operation $sum_w(v)$ that returns $\sum_{e \in P} w(e) \cdot c(e)$ where $P$ is the path from $v$ to the root. Note that for the special case where $w(e) = 1$ for every $e$, $sum_w(v)$ is just the sum of the costs of the edges on the path from $v$ to the root. An efficient implementation should run in $O(\log n)$ time. Do not hurt the logarithmic running time of any of the other operations.