Another interesting consequence of Theorem 3 is that we can relate the behavior of a self-adjusting tree to that of any static tree.

**Theorem 4:** Let \( t \) be the number of comparisons that occur in a sequence of searches from the root in a static binary search tree with \( n \) nodes. The time to do the same sequence of splay operations in a self-adjusting search tree is \( O(t+n^2) \).

**Proof:** Let the root of the static tree be \( r \). Let the depth of a node \( x \) in the static tree (denoted \( d(x) \)) be the distance from \( x \) to the root, \( r \). (\( d(r) = 0 \).) We assign individual weights to the nodes as follows: For the root \( r \), \( iw(r) = 3^d \) (where \( d \) is the largest depth in the tree). For any other vertex \( x \), \( iw(x) = 3^{-d(x)} iw(r) \). With this definition the deepest node has weight 1. It is easy to show by induction that \( 3iw(x) \geq tw(x) \) for all nodes \( x \). (Here \( tw(x) \) denotes the total weight of \( x \) in the static tree.) In particular we have \( 3iw(r) \geq tw(r) \), from which it follows that \( iw(x) \geq 3^{-d(x)-1} tw(r) \). Rearranging and taking logarithms gives us

\[
(\lg 3)(d(x)+1) \geq \lg \frac{tw(r)}{iw(x)}.
\]

The left hand side of this inequality is \( \lg 3 \) times the number of comparisons needed to search for \( x \) in the static tree. The right hand side is the credit time to splay at \( x \) in a self adjusting tree with the individual weights as specified above.

It remains for us to show that the number of credits initially in the self-adjusting tree is \( O(n^2) \). It is clear that the total weight of any node in the self-adjusting tree is at most \( tw(r) \). But \( tw(r) \leq 3iw(r) = 3^{d+1} \leq 3^n \), because \( d \), the largest depth in the tree, is at most \( n-1 \). This means that the number of credits on each node is at most \( (\lg 3)n \), so the total number of credits in the tree initially is at most \( (\lg 3)n^2 \). \( \square \)

A corollary of this result is that the running time of a self-adjusting tree is within a constant factor of the running time of the optimal static tree for any particular distribution. The surprising thing about this is that the self-adjusting tree behaves this way without knowing anything about the distribution in advance. (Note however that the self-adjusting tree takes some time to "learn" the distribution. This learning time is embodied in the \( O(n^2) \) overhead.)