How long to process a sequence of searches?

If access frequencies are known in advance and initial tree is arbitrary but fixed, an optimum binary search tree (Knuth-style) minimizes the total search time.

What if access frequencies are not known in advance?

What if tree is allowed to change during the sequence?
Total time for a sequence of accesses

= total search time
  (sum of 1+ depth of accessed item, when accessed)

+ total number of rotations
  (between searches arbitrary rotations can be done)
Goal: Compare the minimum-cost off-line strategy with (simple) on-line strategies.

Can an on-line strategy (no future knowledge) achieve a performance within a constant factor of that of the optimum off-line strategy (access requests known in advance)?
Balanced trees have $O(\log n)$ worst-case access, insert, delete time.

Optimum trees minimize average access time for fixed, known probabilities, independent accesses. Construction takes $O(n^2)$ time, $O(n)$ time for approximately optimum tree.

Biased trees are approximately optimum, allow fast insertion, deletion, require known probabilities (but could keep frequency counts) are somewhat complicated.
A Self-Adjusting Search Tree
Previous Self-Adjusting Heuristic

1. **Move to root**: do single rotations all along access path.

   ![Diagrams showing example of move to root](Image)

2. **Single exchange**: do one rotation at parent of accessed node.

   ![Diagrams showing example of single exchange](Image)

Both are $O(n)$ per operation, even amortized.
Bad Examples

MTR

SE
Splaying: Sleator and Tarjan (1985)

Rotate each edge along an access path.

Perform rotations in pairs, roughly bottom-up.

Access path is (roughly) halved, other nodes can move down, but only by a few steps.
Cases of Splaying

zig

zig-zig

zig-zag
Step by Step Examples

\[
\begin{array}{c}
\rightarrow 1 \\
\rightarrow 1 \\
\rightarrow 1 \\
\end{array}
\]
EXAMPLES

1 2 3 4 5 6 7 8 9 10
splay

1 2 3 4 5 6 7 8 9 10
splay

1 4 8 10 3 6 7 5
Accessed node moves to root, distance of the other nodes from the root essentially halves.
Splaying in sequential order

\[
\begin{align*}
\text{average} & = 3^{2/3} 
\end{align*}
\]
What is Known

Let $m$ be the number of accesses, $n$ the number of nodes. Assume $m \geq n$.

Total time for $m$ accesses $= O(m \log n)$: matches bound for balanced trees.

Total time for any access sequence is within a constant factor of that for an optimum static tree.

Total time for $n$ accesses, one per item, in symmetric order, is $O(n)$. 

+ 25
Potential: define the total weight of a node to be the sum of the individual weights of its descendants, including itself.

The potential of a tree is the sum of the (base-two) logarithms of the weights of its nodes.

\[ \Phi = \sum_{i=1}^{n} \log_2 \left( +w_i \right) \]
Access Lemma

For any assignment of positive weights to items, the amortized time to access item \( i \) is at most

\[ 3 \log \left( \frac{W}{w_i} \right) + 1 \]

where \( W = \text{total weight} \) and the cost of an access is the depth of the accessed node.

Note: The item weights are parameters of the analysis, not of the algorithm.
Let \( tw(x) = \text{sum of weights of all items in subtree of } x \)

rank of \( x = r(x) = \log_2 tw(x) \)

We shall show:

amortized time of a splay step at \( x \) is

\[
\leq 3 \left( r'(x) - r(x) \right) + 1 \text{ if zig}
\]

\[
\uparrow \quad \uparrow
\]

after \quad before

Then total amortized time of splay is

\[
\leq 3 \left( r_{\text{final}}(x) - r_{\text{initial}}(x) \right) + 1
\]

\[
\leq 3 \left( \log W - \log w_i \right) + 1
\]

\[
\leq 3 \log \left( \frac{W}{w_i} \right) + 1
\]
Analysis of Case 2 (zig-zig) Step

Amortized time of step

\[
= 1 + r'(y) + r'(z) - r(x) - r(y)
\]

\[
\leq 1 + r'(x) + r'(z) - 2r(x) \quad \text{since} \quad r'(x) \geq r'(y), \quad r(y) \geq r(x)
\]

\[
\leq 3(r'(x) - r(x)) \quad \text{iff}
\]

\[
2r'(x) - r(x) - r'(z) \geq 1.
\]

But \( r'(x) \geq \max \{r(x), r'(z)\} \). Also, \( tw(x) + tw'(z) \leq tw'(x) \).

Thus \( \min \{tw(x), tw'(z)\} \leq tw'(x)/2 \). I.e. \( r'(x) \geq \min \{r(x), r'(z)\} + 1 \).

\[
r(x) = \log tw(x)
\]
Corollaries

Balance Theorem
The total time for m accesses in an n-node tree is \( O((m+n) \log (n+2)) \).

Static Optimality Theorem
If every item is accessed at least once, the total access time is \( O(m + \sum_{i=1}^{\infty} q_i \log (m/q_i)) \), where \( q_i \) is the access frequency of item \( i \).
Extension of argument shows that self-adjusting
trees are as efficient (to within a
constant factor) as optimum trees, over
a sequence of operations.
Static Finger Theorem
The total access time is
\[ o(n \log n + \sum_{j=1}^{m_3} \log(d(i,j,f)+2)) \]
where \( f \) is any fixed item, \( ij \) is the item accessed during the \( j^{th} \) access, and \( d(i,i') \) is the (symmetric-order) distance between items \( i \) and \( i' \).
"Working Set" Theorem

The total access time is

\[ o(n \log n + \sum_{j=1}^{m} \log(t(i, j)+2)) \]

where \( t(i, j) \) is the number of different items accessed before access \( j \) since the last access of item \( i \).
Access lemma holds for variants of splaying, including top-down and move half-way to root methods. For the latter, the constant factor is 2.
Thm. Total time to access all items once, in symmetric order, using splaying = $O(n)$.

(any initial tree)
Conjecture

Dynamic Optimality

For any access sequence, splaying minimizes the total access time to within a constant factor among dynamic binary search tree algorithms, assuming unit cost per rotation and access cost equal to depth.

(Initial tree is given or $+O(n)$ term)