Other/Better Criteria?
- Expected case: Some keys more popular than others
- Self-adjusting: Tree adapts as popularity changes

How to design/analyze?
- Splay Tree: A self-adjusting binary search tree
  - No rules! (yay anarchy!)
    - No balance factors
    - No limits on tree height
    - No color/levels/priorities
  - Amortized efficiency:
    - Any single op - slow
    - Long series - efficient on avg.

Intuition: Let T be an unbalanced BST, suppose we access its deepest key

Recap: Lots of search trees
- Unbalanced BSTs
- AVL Trees
- 2-3, Red-black, AA Trees
- Treaps & Skip lists

Focus: Worst-case or randomized expected case

Lesson:
Different combinations of rotations can:
- bring given node to root
- significantly change (improve) tree structure.

Splay Trees 1

Final

Tree's height has reduced by ~ half!

Idea I: Rotate "a" to top
(Future accesses to "a" fast)

...final result:

Idea II: Rotate 2 at a time - upper + lower

still unbalanced
ZigZag(p): [LL case]

Splay (Key x):
Node p ← find (x) [nearest node]
while (p ≠ root) {
    if (p is child of root) zig(p)
    else /* p has grand parent */ zigzag(p)
}

insert (x):
splay (x)
g = new Node (x)
if (root.key < x) 
x.left = root
x.right = root.right
root.right = null
else ... symmetrical...

find (x):
splay (x)
if (root.key == x) 
    found!
else not found

Example: splay (3)

Subtrees A, C move up↑

ZigZigLPf: [LR case]

Subtrees A, C move up↑

ZigLPf: [L case]

Subtree A moves up↑

Subtrees C, E of p move up↑

Finals
Splay Trees III

**Dynamic Finger Theorem:**
- Key: $x_1, \ldots, x_n$. We perform
  accesses $x_{i_1}, x_{i_2}, \ldots, x_{i_m}$ Let
  $\Delta_j = i_j - i_{j-1}$ distance
  between consecutive
  items.
- Thm: Total access time is
  \[ O(m+n \log n + \sum_{j=1}^m (1 + \log \Delta_j)) \]

**Static Optimality:**
- Suppose key $x_i$ is accessed with
  prob $p_i$. \((\sum p_i = 1)\)
- Information Theory:
  - Best possible binary search
    tree answers queries in
    expected time $O(H)$ where
    \[ H = \sum p_i \log \frac{1}{p_i} \]
    Entropy
- Given a seq. of $m$ ops. on splay
tree with keys $x_1, \ldots, x_n$, where
  $x_i$ is accessed $q_i$ times. Let
  $p_i = q_i / m$. Then total time is
  \[ O(m \sum p_i \log \frac{1}{p_i}) \]