Splay Trees

**Question 1.** We saw AVL trees, which guaranteed a search time of $O(\log n)$, where $n$ is the number of nodes in the tree. Is it reasonable to optimize to guarantee the worst-case lookup time is $O(\log n)$, or are there other things we should be worried about?

In a splay tree, our goal is that recently-used data should be near the top. We add and search as per a normal binary search tree, except when we’re done, we’re going to splay it to the root.

If we just inserted $x$ and it is now a child of the root (whether because we inserted it there or because it splayed up to that location), the adjustment is easy:

Before

```
   p
   / \
  x   T_0
    /   /
   T_1  T_2
```

After

```
   x
   / \
   p   T_0
    /   /
   T_1  T_2
```
Otherwise, if $x$ is not a child of the root (and is also not a child of the root), we’re going to bring it up two steps. There are two possibilities (four if you count the mirrors):

Before

```
  p
 /|
/  |
  g
```

After

```
  x
 /
/ |
 x
```

Question 2. Consider the following splay tree; what does it look like after `find(1)`?

```
6
|
5 7
 |
4
 |
3
 |
2
 |
1
```

Question 3. From the tree that resulted, what does it look like after `find(3)`?

Question 4. Suppose we start with an initially empty splay tree and then insert the values $1, 2, \ldots, n$ in order and then call `find(1)`. What is the total running time? What is the amortized time per operation?

The “Splay Trees Are Awesome” Conjecture: there is a conjecture that, for any sequence of binary search tree operations, splay trees are asymptotically as fast as any other implementation (basic, AVL, Red/Black, etc). Some sets can be done in less than $O(\log n)$ time, and the conjecture is that if one binary search tree implementation can do it, so can splay trees.