Name: _____

Section: _____

Today's Goals:

- (TBA)

Today's Question(s)

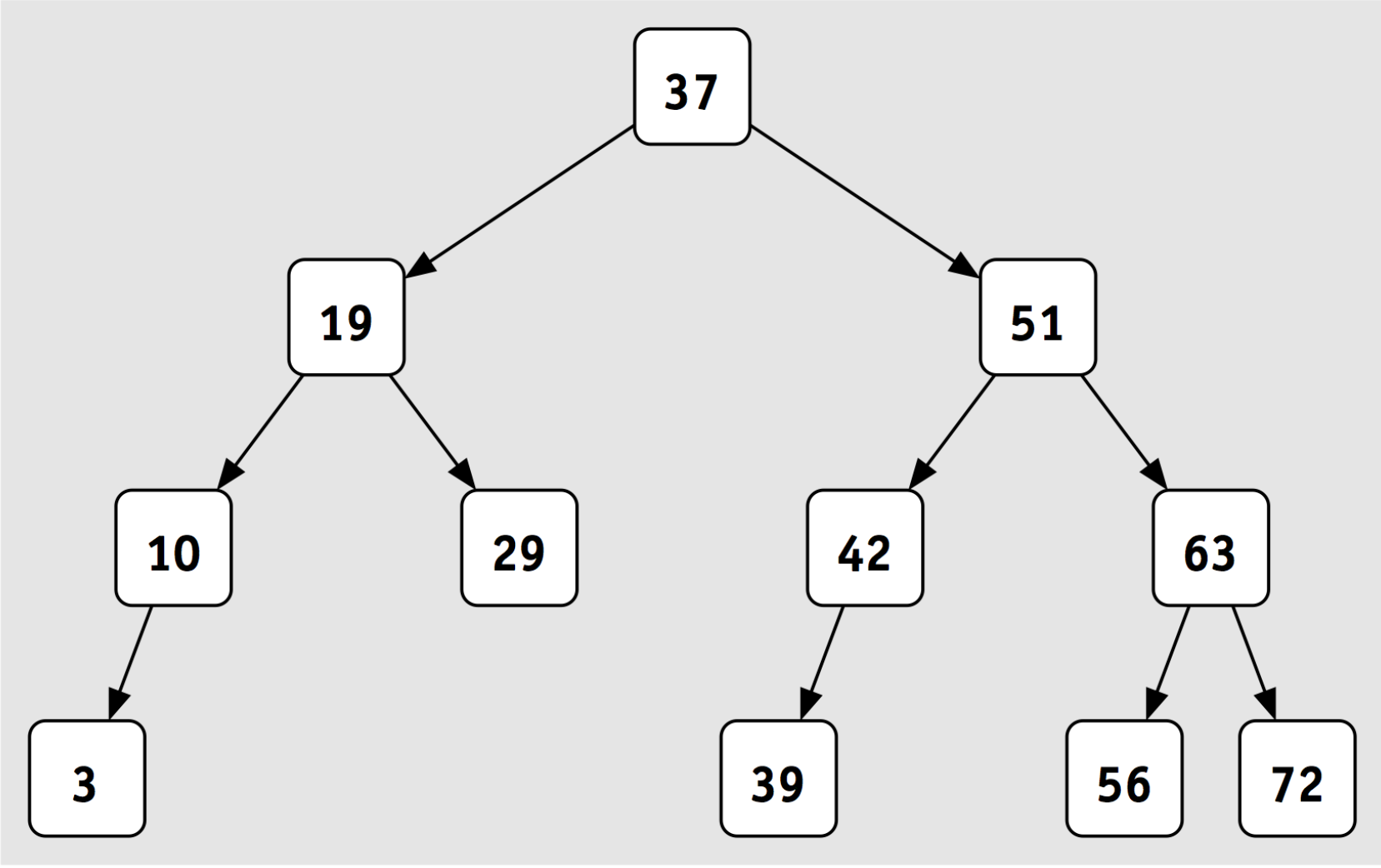
(TBA)

QOTD:

Goals: * Implement insert in two different ways * Motivate and define tree rotations * Explain how randomized trees work

Reminder: Binary Search Trees (BSTs)

- Each node has at most 2 children
- All nodes in left subtree are less than parent
- All nodes in right subtree are greater than parent



Reminder: Tree Encoding

```
class StringBinarySearchTree {
public:
  ...
private:
  struct Node {
    string      value_;
    Node* left_;
    Node* right_;
  };
  Node* root_;
};
```

Reminder: Tree Lookup

```
bool lookup(const string& key) {
  return lookupNode(root_, key);
}

bool lookupNode(const Node* node, const string& key) {
    if (node == nullptr)
        return false;
    else if (key < node->value_)
        return lookupNode(node->left_, key);
    else if (node->value_ < key)
        return lookupNode(node->right_, key);
    else
        return true;
```

Insertion

```
void StringTree::insert(const string& key) {
    nodeInsert(root_, key);
}

void StringTree::nodeInsert(Node*& node, const string& key)
    if (node == nullptr) {
        node = new Node{key};
    }
    else if (key < node->value_)
        insertNode(node->left_, key);
    else if (node->value_ < key)
        insertNode(node->right_, key);
    else
        return; // Duplicate is undefined behavior
}
```

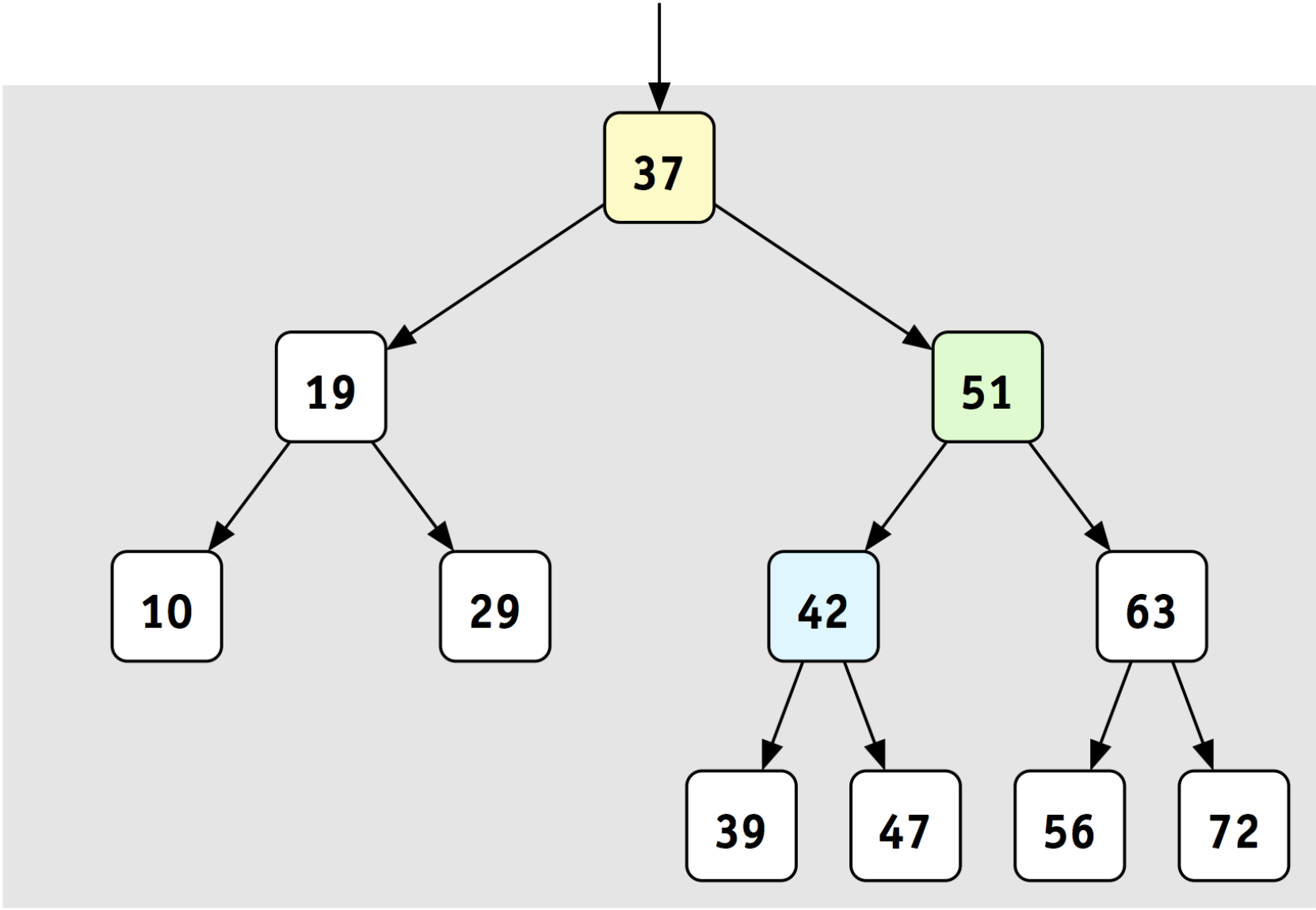
Practice: Create a tree from one of these

- D B F C G E A
- A B C D E F G
- D C B A E F G

Rearranging Trees

Rearrange the tree to make 51 the root (in constant time)

Rotations



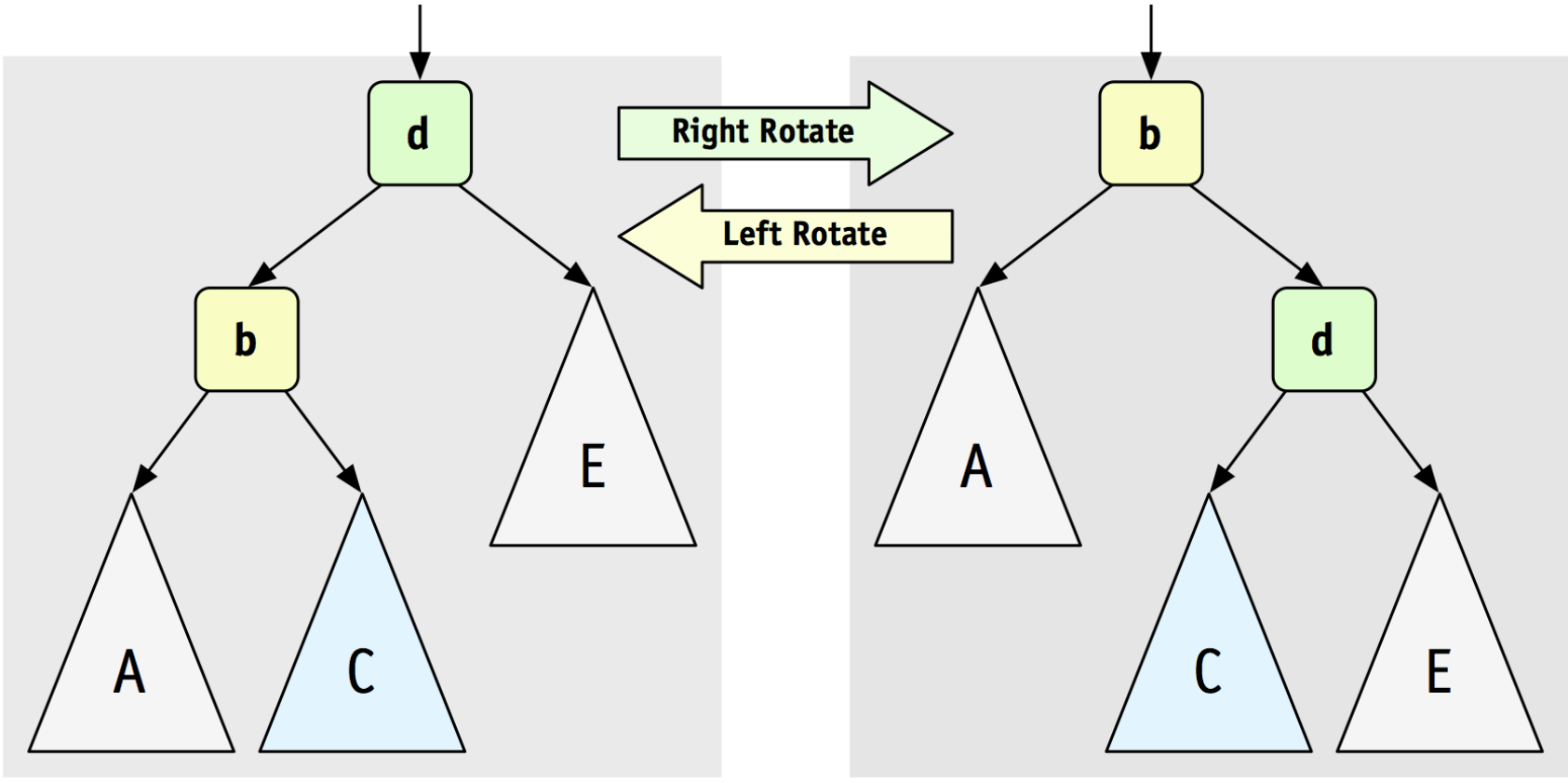
Rotation Code

```
void rotateRight(Node * & top) {
  Node * b = top->left_;     // b is d's left child
  top->left_ = b->right_;    // C becomes left child of d
  b->right_ = top;           // d becomes right child of b
  top = b;                   // top is now b
}
```

Rotation Code: Left

```
void rotateLeft(Node * & top) {
  Node * d = top->right_;   // d is b's left child
  top->right_ = d->left_;   // C becomes right child of b
  d->left_ = top;           // b becomes left child of d
  top = d;                  // top is now d
}
```

Rotations



Class Exercise

Write insertAtRoot, which

- inserts a value at a leaf, then
- "bubble the value up" to the root

Hint: *use rotations*

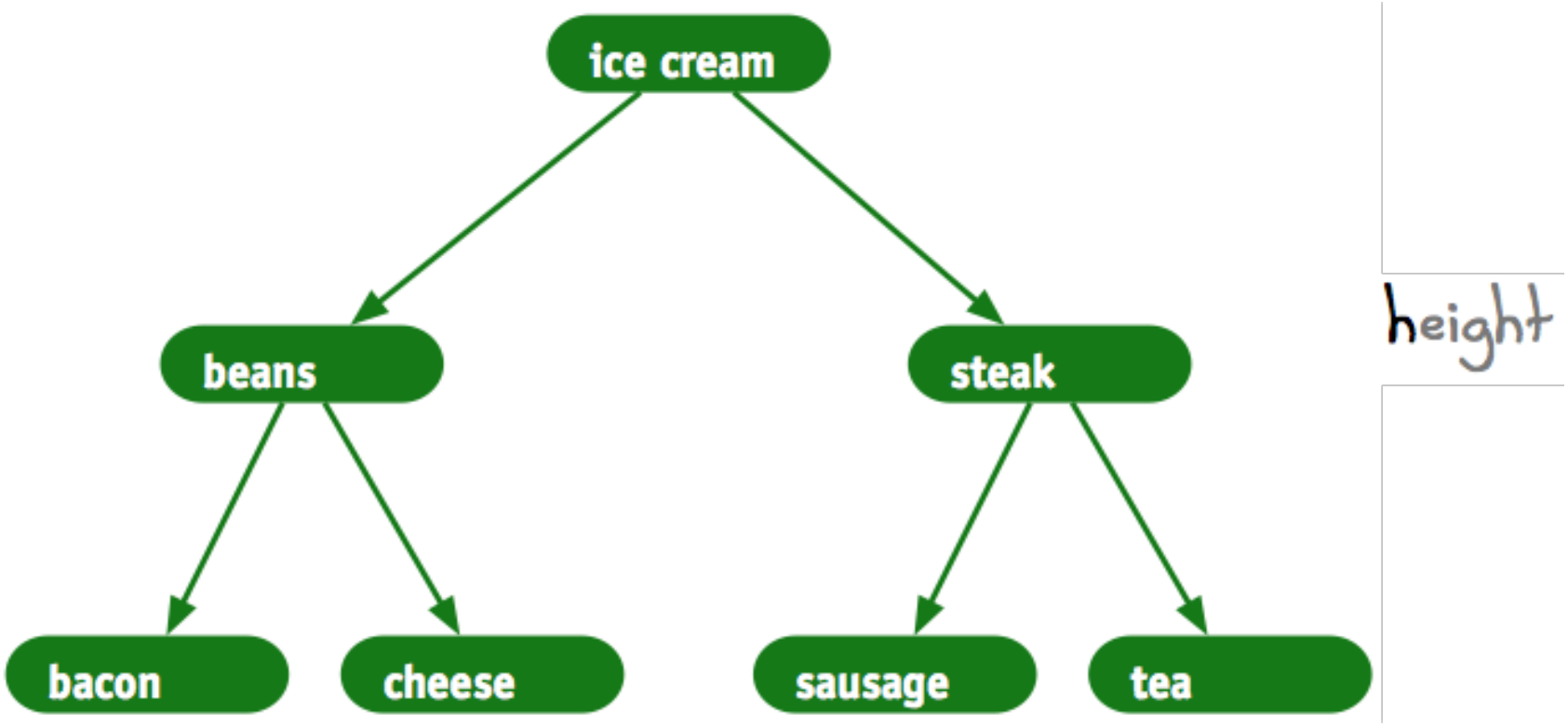
insertAtRoot

```
void insertNodeAtRoot(Node * & node,
                      const T & key) {




}
```

Does this fix our stick problem?

- A B C D E F G

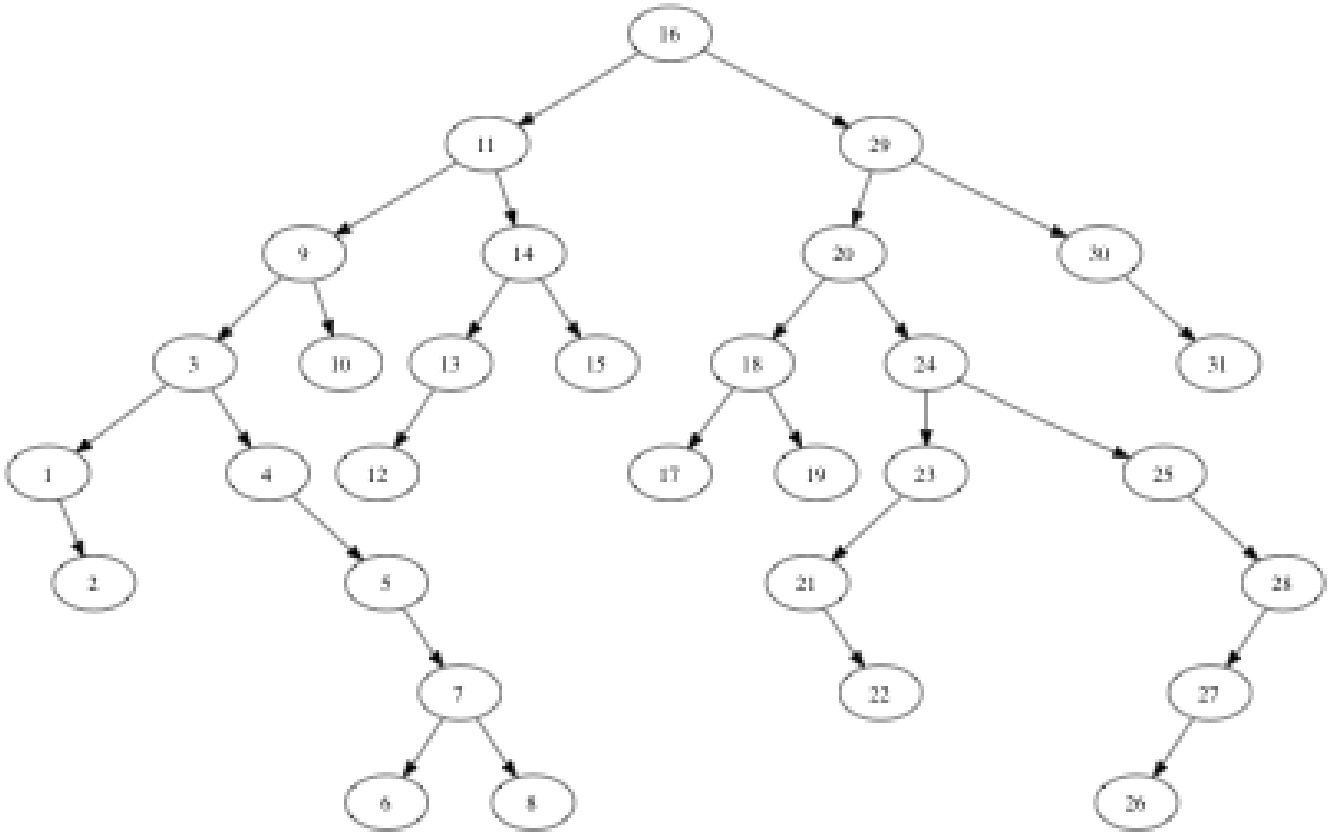
Perfect trees: definitions and properties



Random Trees

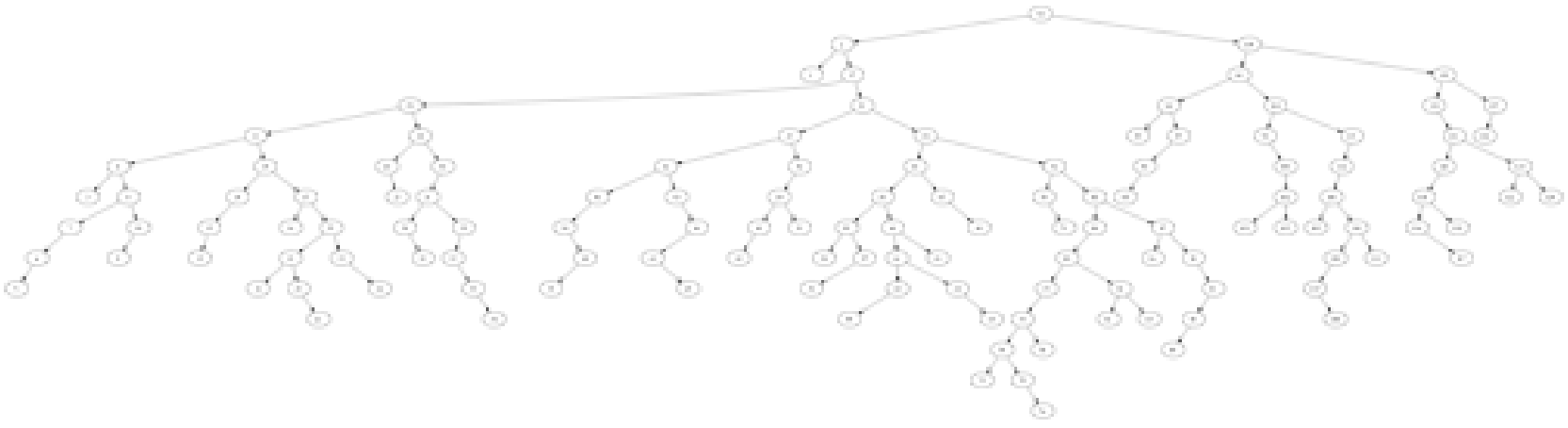
What do they look like?

What are their properties?

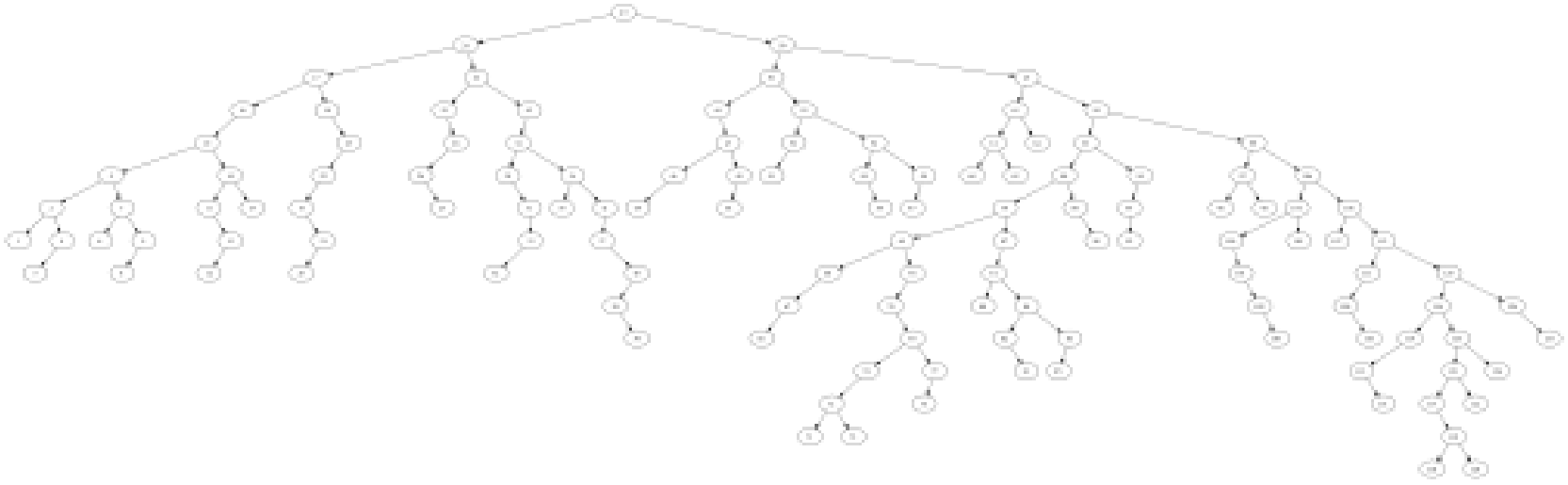


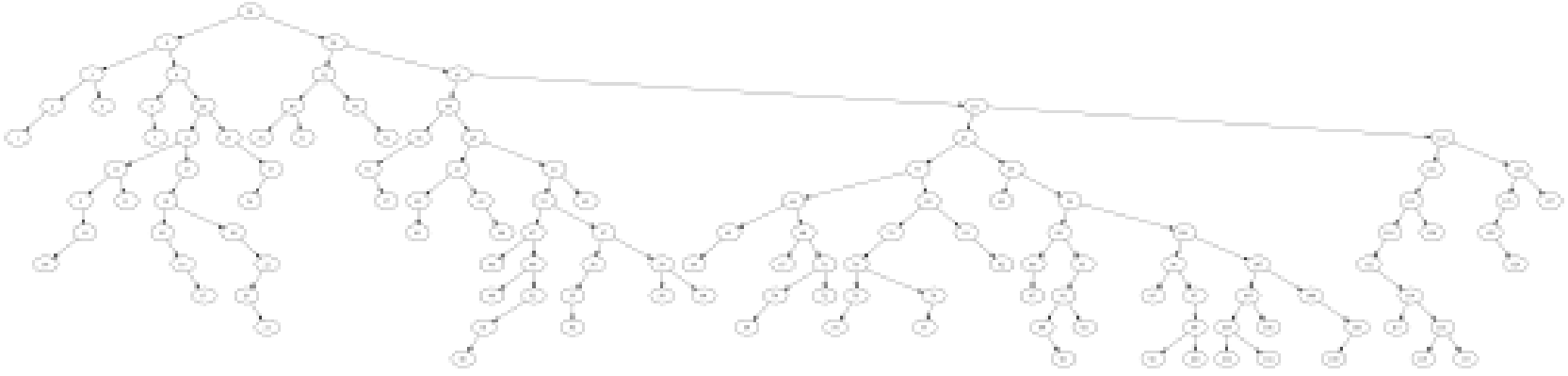
Random Info

Suppose I build a tree from a random permutation of 127 items. . .



What can we say about the likely trees we'll make. . . ?





With 127 nodes, how many possible random orders?

How many possible trees?

With 63 nodes, how many possible random orders?

How many possible trees?

Unsuccessful Search

Compared to perfection:

$$\lim_{n\to\infty} \frac{2H_{n+1} - 2}{\log_2(n+1)} = 2\log 2 \approx 1.38629$$

Random Trees Are Good!

Observe that *if keys come in in random order*, the tree will tend to be reasonably balanced (about 38% worse than a perfectly balanced tree).

Randomized Trees

Simulate random insertion by

-
-

But how often should we do each?