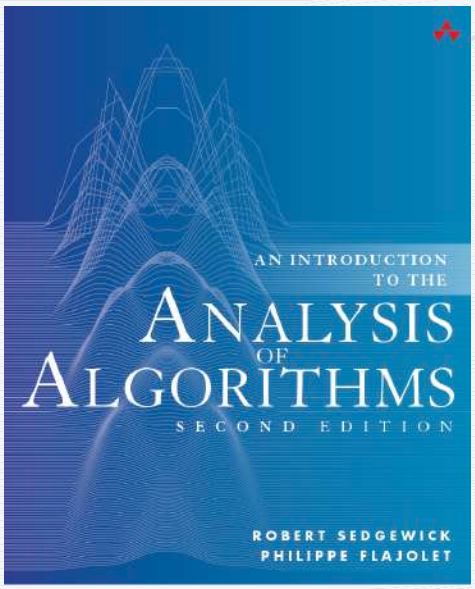


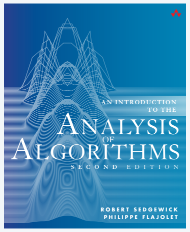
http://aofa.cs.princeton.edu

6. Trees

Review

First half of class

- Introduced analysis of algoritihms.
- Surveyed basic mathematics needed for scientific studies.
- Introduced analytic combinatorics.



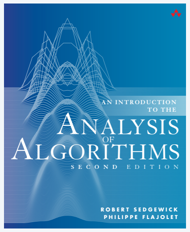
1	Analysis of Algorithms
2	Recurrences
3	Generating Functions
4	Asymptotics
5	Analytic Combinatorics

Note: Many applications beyond analysis of algorithms.

Orientation

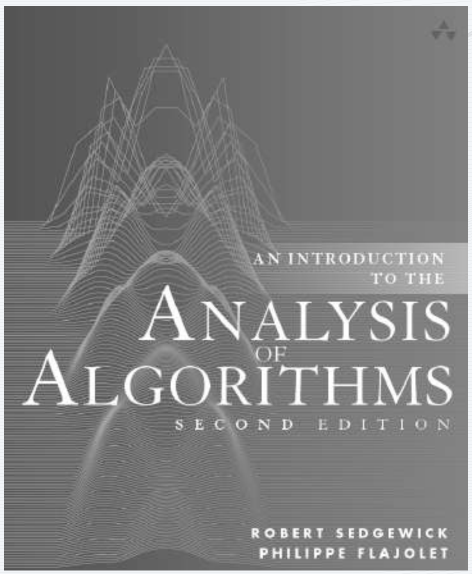
Second half of class

- Surveys fundamental combinatorial classes.
- Considers techniques from analytic combinatorics to study them .
- Includes applications to the analysis of algorithms.



chapter	*combinatorial classes*	*type of class*	*type of GF*
6	Trees	unlabeled	OGFs
7	Permutations	labeled	EGFs
8	Strings and Tries	unlabeled	OGFs
9	Words and Mappings	labeled	EGFs

Note: Many more examples in book than in lectures.



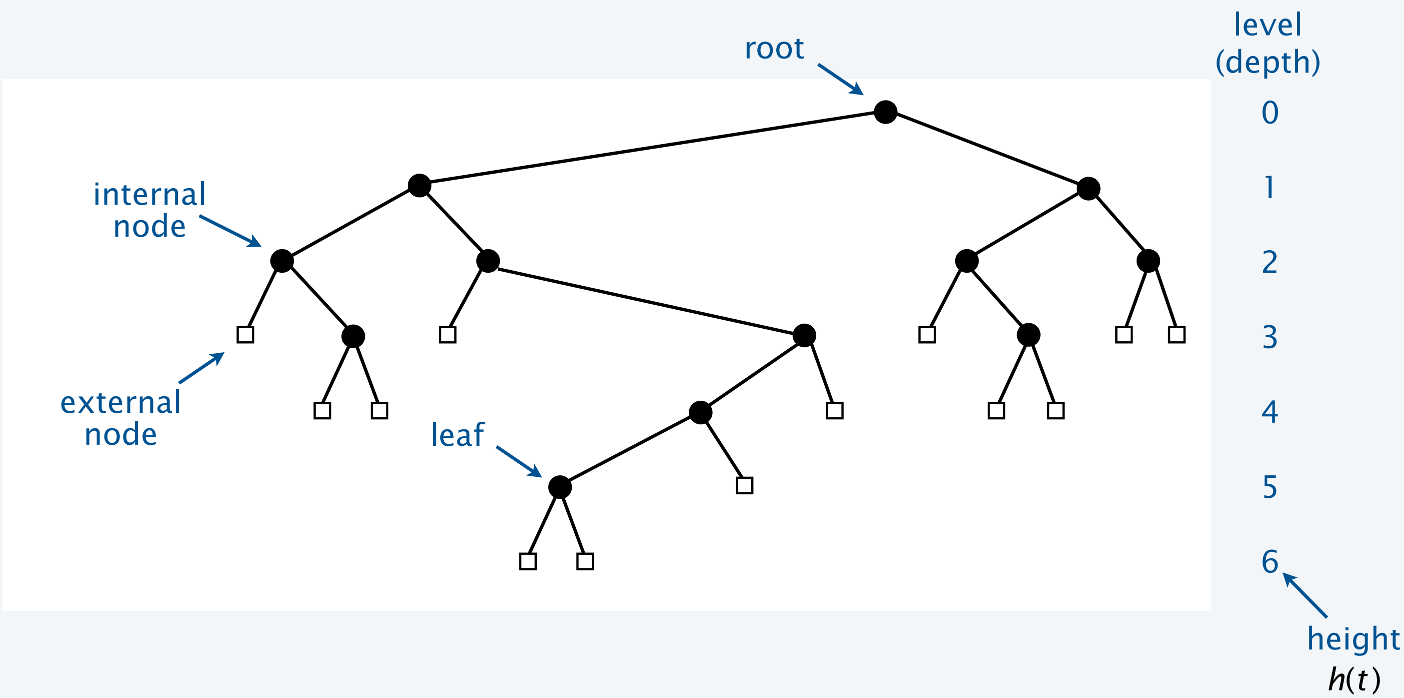
http://aofa.cs.princeton.edu

6. Trees

- **Trees and forests**
- Binary search trees
- Path length
- Other types of trees

Anatomy of a binary tree

Definition. A *binary tree* is an external node or an internal node and two binary trees.

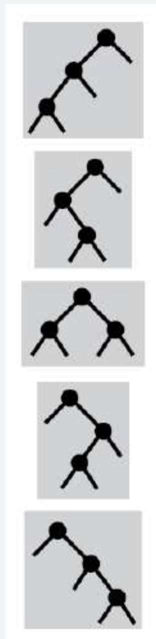


Binary tree enumeration (quick review)

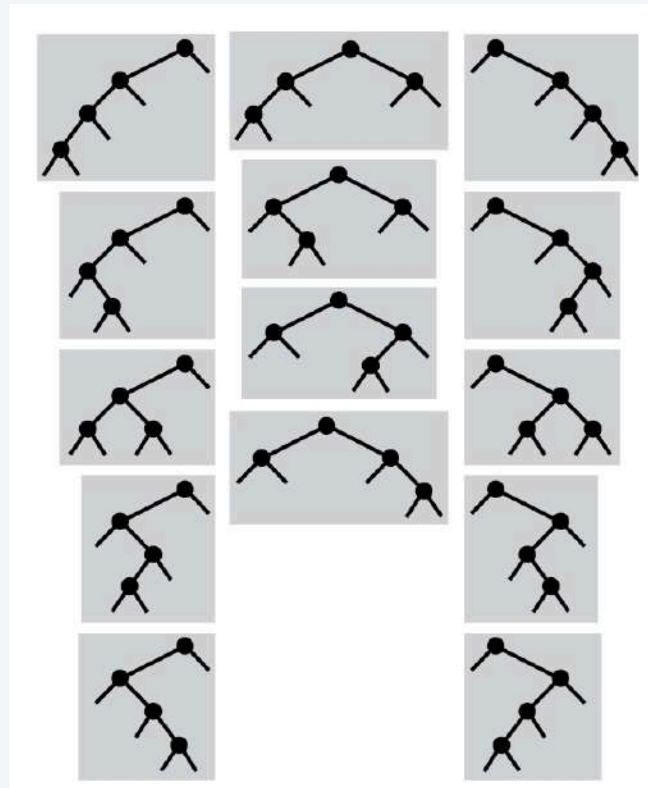
How many binary trees with N nodes?

$T_1 = 1$

$T_2 = 2$



$T_3 = 5$



$T_4 = 14$

Symbolic method: binary trees

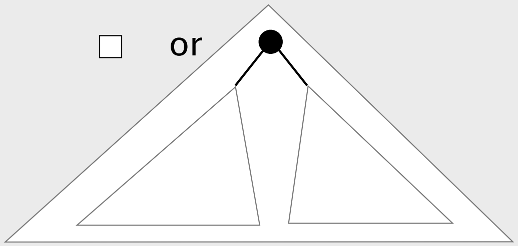
How many binary trees with N nodes?

Class	T, the class of all binary trees
Size	$\lvert t \rvert$, the number of internal nodes in t
OGF	$T(z) = \sum_{t \in T} z^{\lvert t \rvert} = \sum_{N \geq 0} T_N z^N$

Atoms

type	*class*	*size*	*GF*
external node	$Z_\square$	0	1
internal node	$Z_\bullet$	1	z

“a binary tree is an external node or an internal node connected to two binary trees”



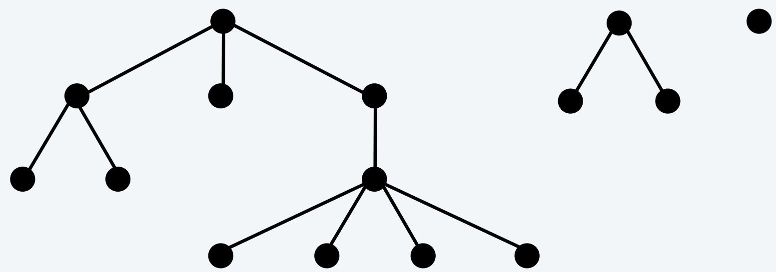
Construction $\quad T = Z_\square + T \times Z_\bullet \times T$

OGF equation $\quad T(z) = 1 + zT(z)^2$

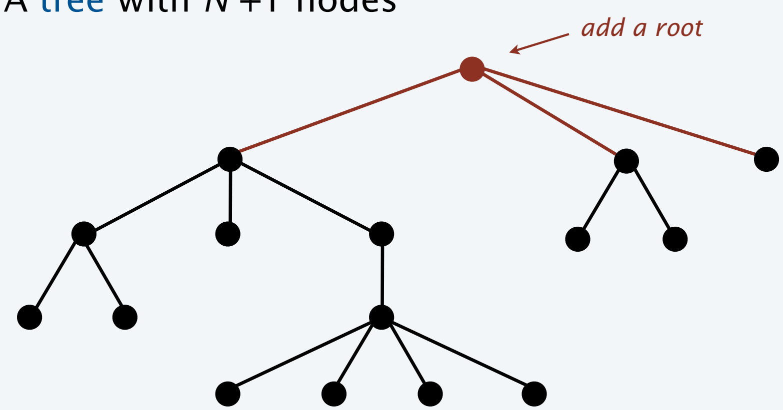
$$[z^N]T(z) = \frac{1}{N+1}\binom{2N}{N} \sim \frac{4^N}{\sqrt{\pi N^3}}$$

Forest and trees

Each forest with N nodes corresponds to



A tree with $N+1$ nodes



$$[z^N]F(z) = [z^{N+1}]G(z)$$
$$zF(z) = G(z)$$

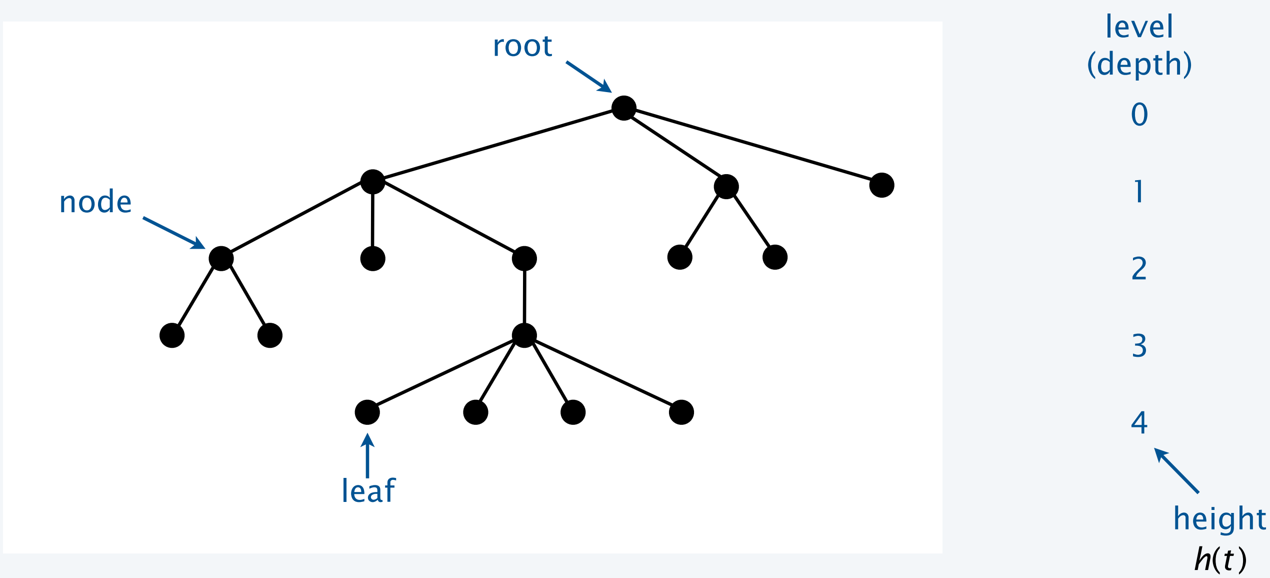
GF that enumerates forests

GF that enumerates trees

Anatomy of a (general) tree

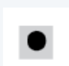
Definition. A *forest* is a sequence of disjoint trees.

Definition. A *tree* is a node (called the *root*) connected to the roots of trees in a forest.

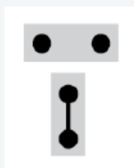


Forest enumeration

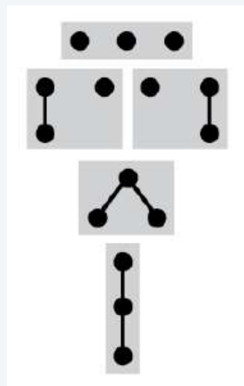
How many forests with N nodes?



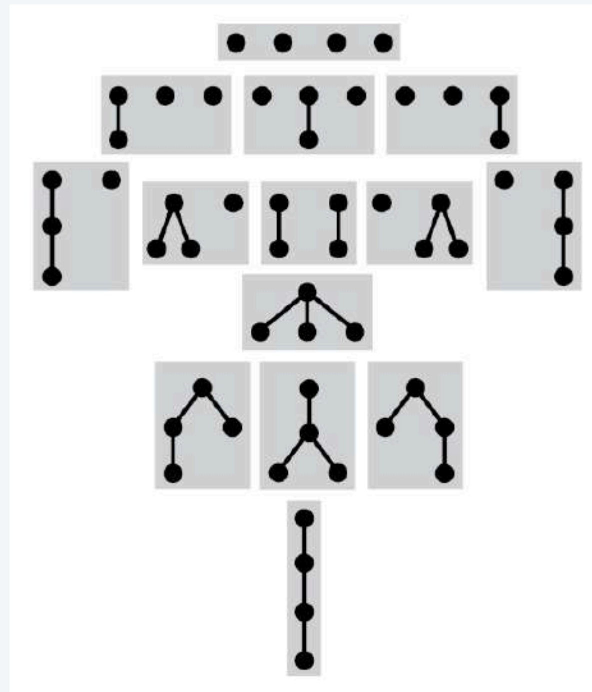
$F_1 = 1$



$F_2 = 2$



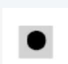
$F_3 = 5$



$F_4 = 14$

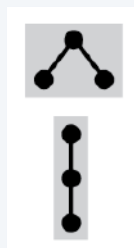
Tree enumeration

How many trees with N nodes?

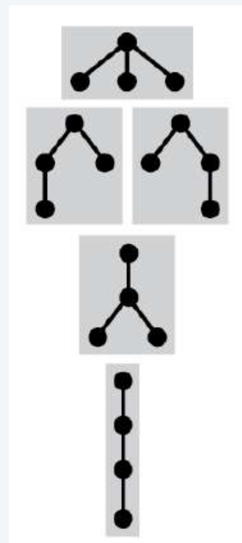


$G_1 = 1$

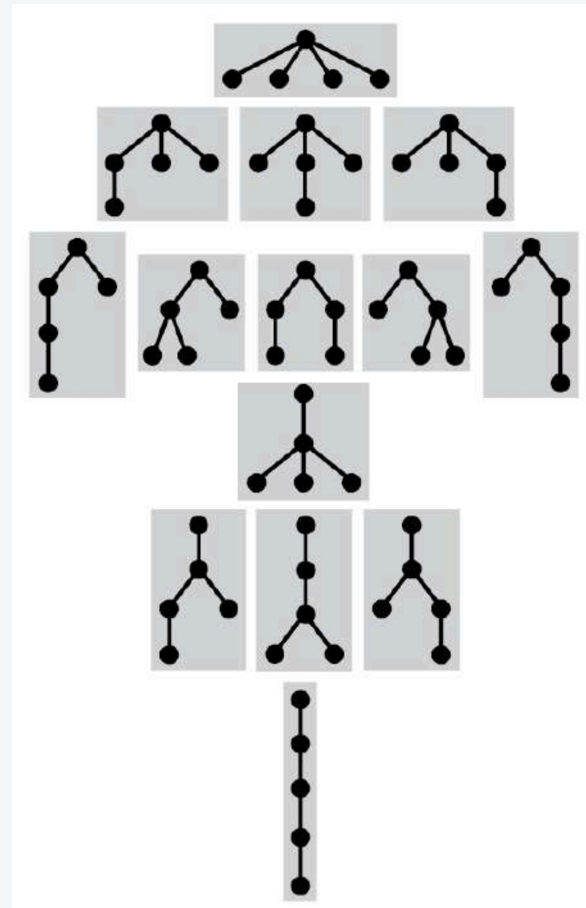
$G_2 = 1$



$G_3 = 2$



$G_3 = 5$



$G_4 = 14$

Symbolic method: forests and trees

How many forests and trees with N nodes?

Class	F, the class of all forests
Size	$\|f\|$, the number of nodes in f

Class	G, the class of all trees
Size	$\|g\|$, the number of nodes in g

Atoms

type	*class*	*size*	*GF*
node	Z	1	z

Construction $$F = SEQ(G) \quad \text{and} \quad G = Z \times F$$

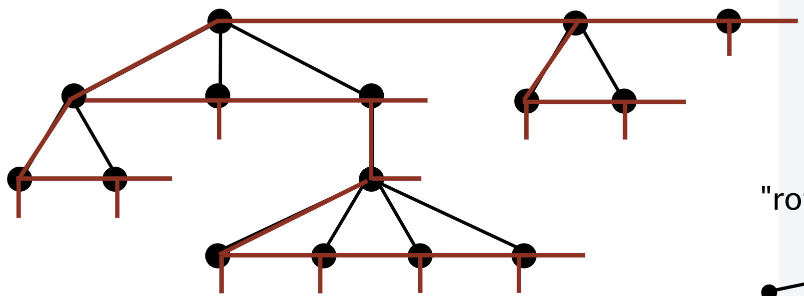
OGF equations $$F(z) = \frac{1}{1 - G(z)} \quad \text{and} \quad G(z) = zF(z)$$

Solution $$F(z) - zF(z)^2 = 1$$

Extract coefficients $$F_N = T_N = \frac{1}{N+1}\binom{2N}{N} \sim \frac{4^N}{\sqrt{\pi N^3}} \qquad G_N = F_{N-1} \sim \frac{4^{N-1}}{\sqrt{\pi N^3}}$$

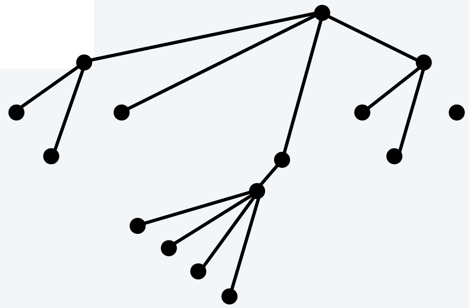
Forest and binary trees

Each forest with N nodes corresponds to

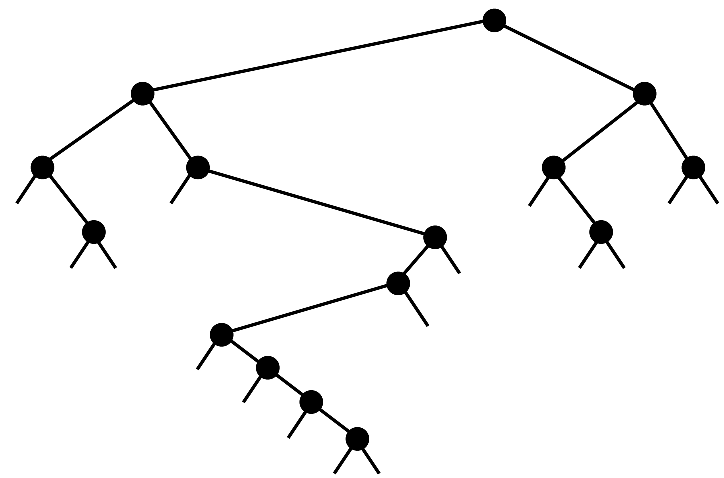


Connect each node to its
- left child
- right sibling

"rotation" correspondence



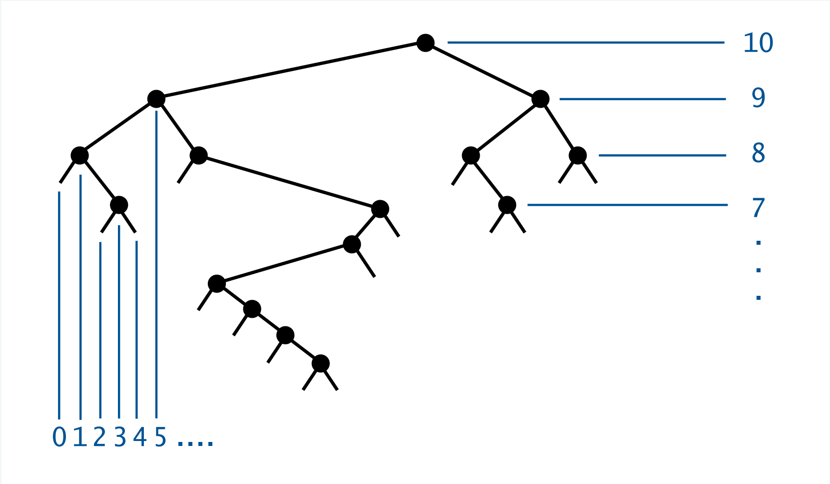
A binary tree with N nodes



Aside: Drawing a binary tree

Approach 1:

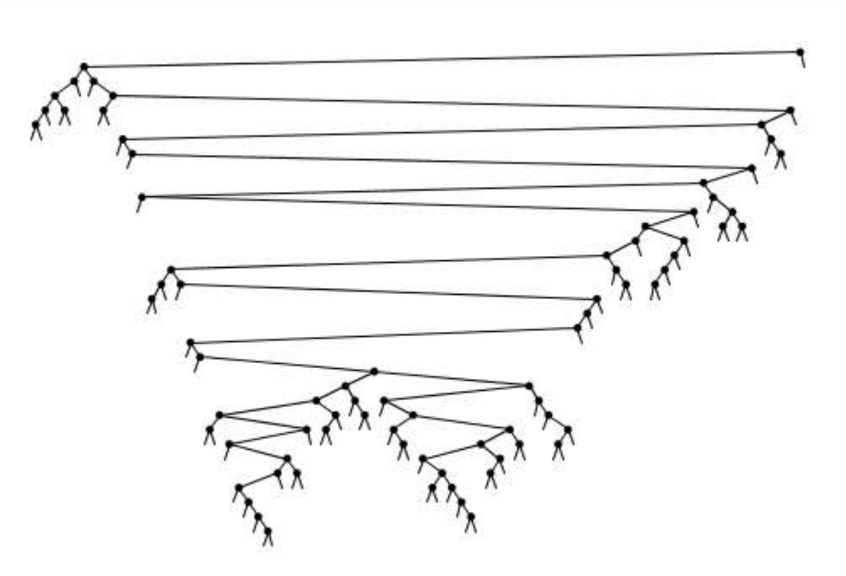
- y-coordinate: height minus node depth
- x-coordinate: inorder node rank



Design decision:

Reduce visual clutter by omitting external nodes

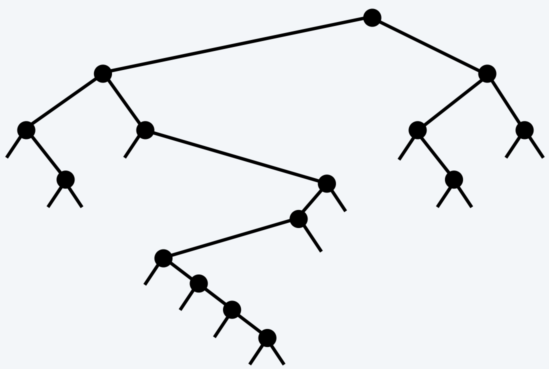
Problem: distracting long edges

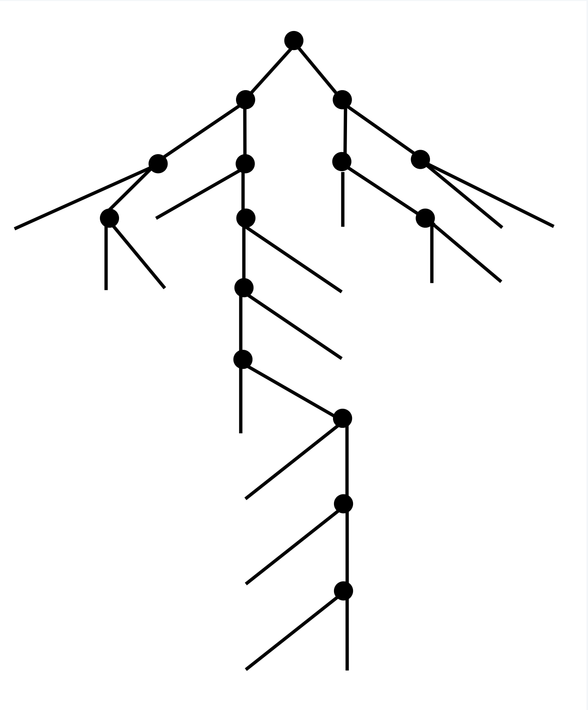


Aside: Drawing a binary tree

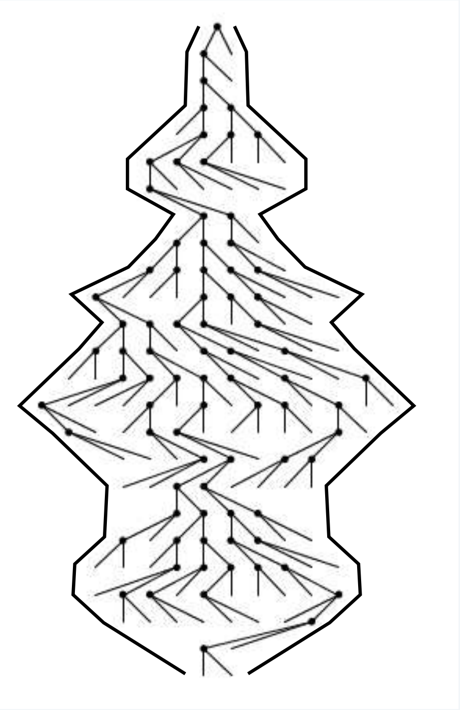
Approach 2:

- y-coordinate: height minus node depth
- x-coordinate: centered and evenly spaced by level

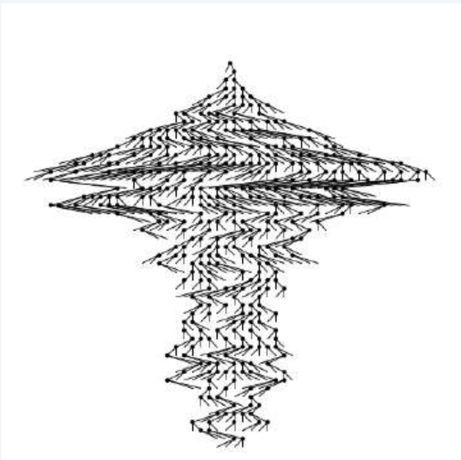


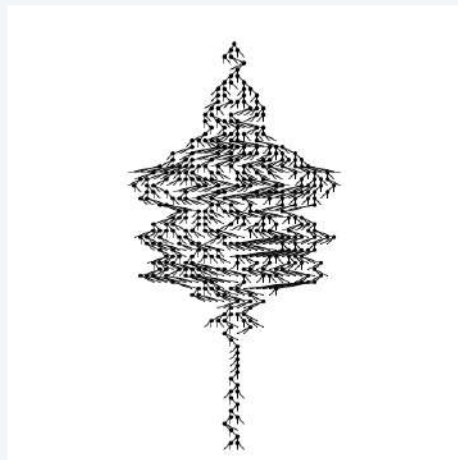


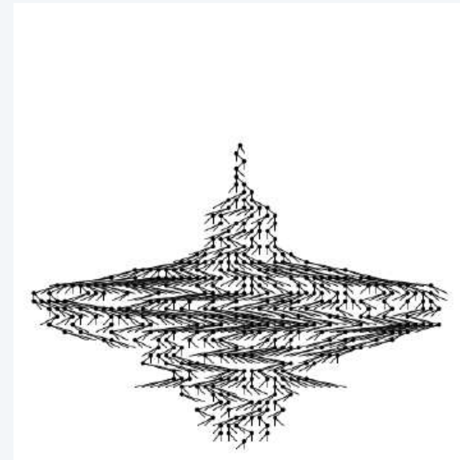
Drawing shows tree *profile*

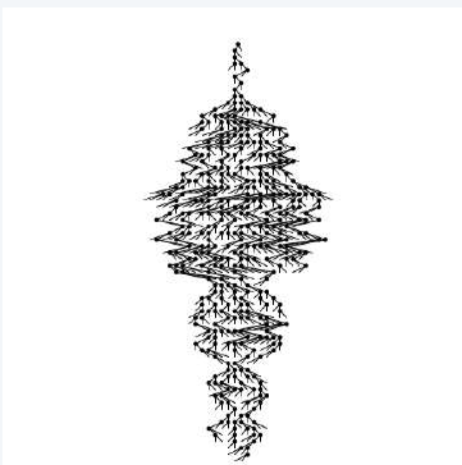


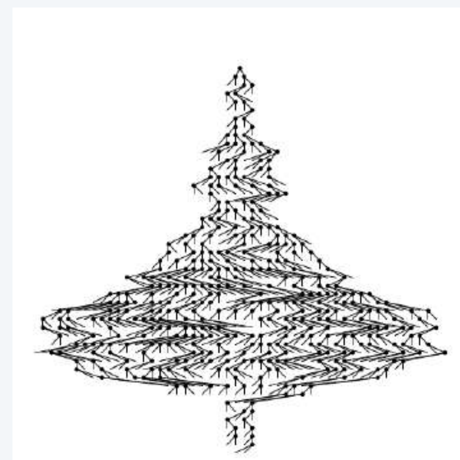
Typical random binary tree shapes (400 nodes)



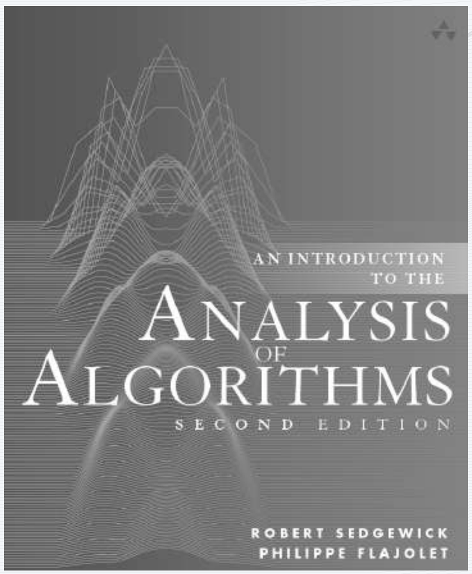








Challenge: characterize analytically



http://aofa.cs.princeton.edu

6. Trees

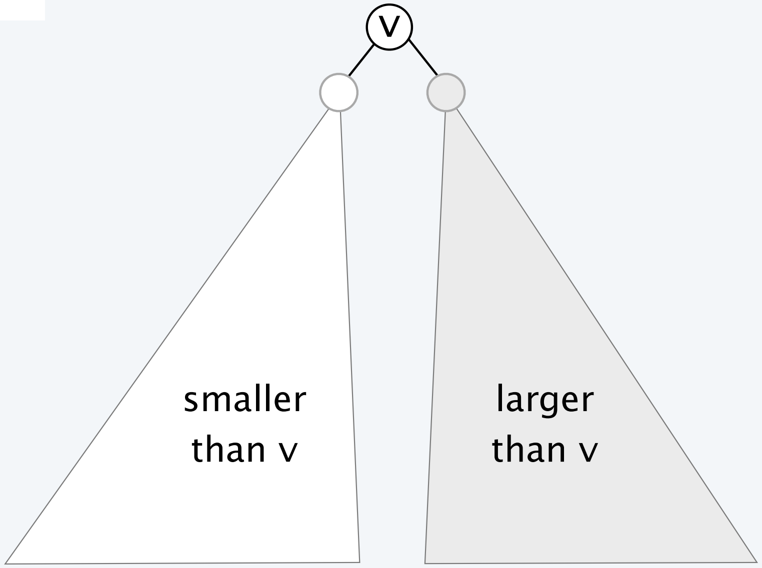
- Trees and forests
- **Binary search trees**
- Path length
- Other types of trees

Binary search tree (BST)

Fundamental data structure in computer science:

- Each node has a key, with comparable values.
- Keys are all distinct.
- Each node's left subtree has smaller keys.
- Each node's right subtree has larger keys.

Section 3.2



BST representation in Java

Java definition: A BST is a reference to a root Node.

A Node is comprised of four fields:

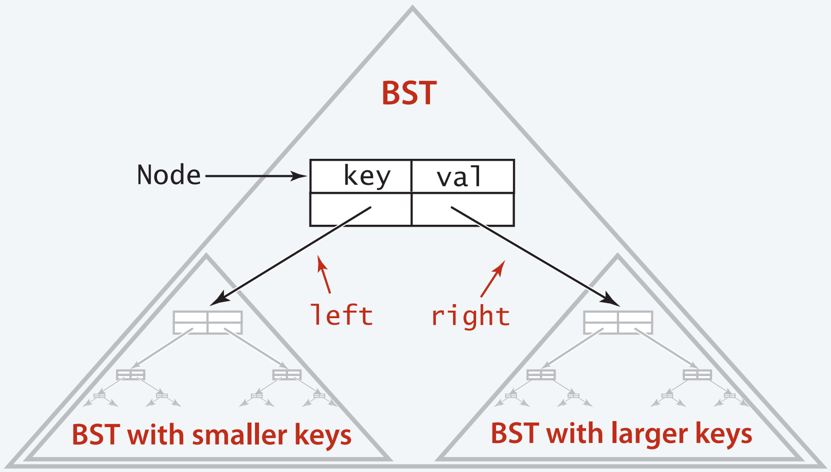
- A Key and a Value.
- A reference to the left and right subtree.

smaller keys larger keys

Notes:

- Key and Value are generic types.
- Key is Comparable.

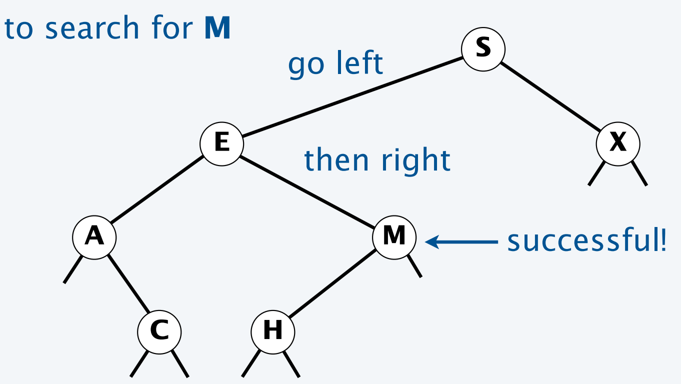
```
private class Node
{
   private Key key;
   private Value val;
   private Node left, right;
   public Node(Key key, Value val)
   {
      this.key = key;
      this.val = val;
   }
}
```

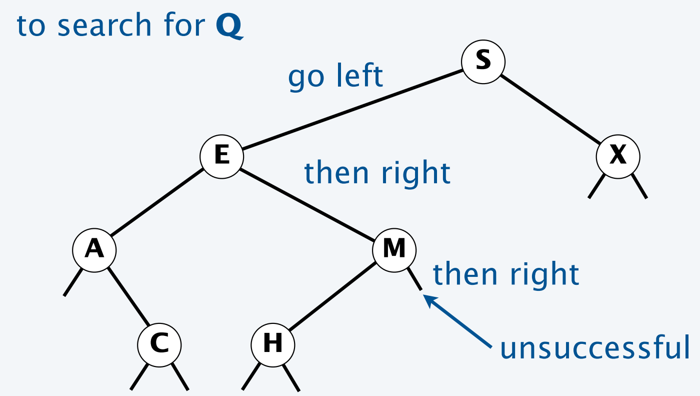


Binary search tree

BST implementation (search)

```
public class BST<Key extends Comparable<Key>, Value>
{
   private Node root;

   private class Node
   {  /* see previous slide */  }

   public Value get(Key key)
   {
      Node x = root;
      while (x != null)
      {
         int cmp = key.compareTo(x.key);
         if      (cmp  < 0) x = x.left;
         else if (cmp  > 0) x = x.right;
         else if (cmp == 0) return x.val;
      }
      return null;
   }

   public void put(Key key, Value val)
   {  /* see next slide */  }

}
```

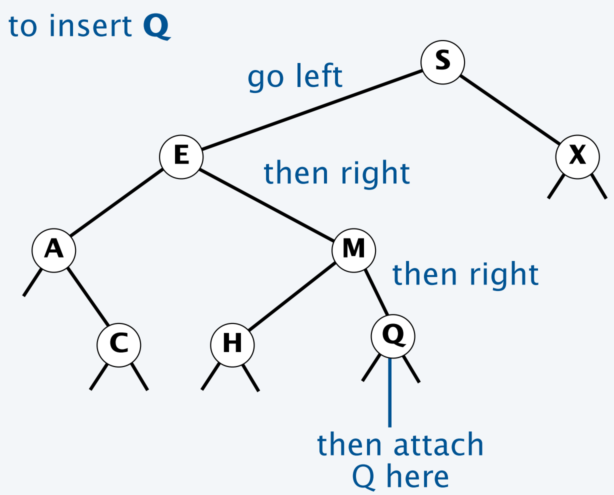




BST implementation (insert)

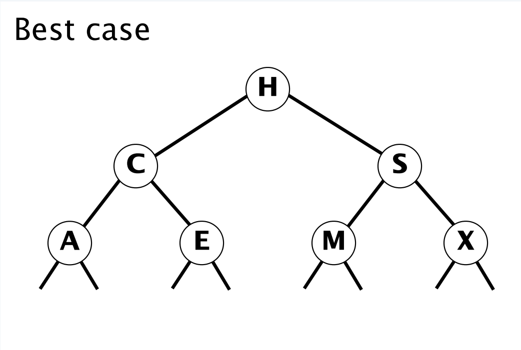
```
public void put(Key key, Value val)
{  root = put(root, key, val);  }

private Node put(Node x, Key key, Value val)
{
   if (x == null) return new Node(key, val);
   int cmp = key.compareTo(x.key);
   if      (cmp  < 0) x.left  = put(x.left,  key, val);
   else if (cmp  > 0) x.right = put(x.right, key, val);
   else if (cmp == 0) x.val = val;
   return x;
}
```

concise, but tricky, recursive code

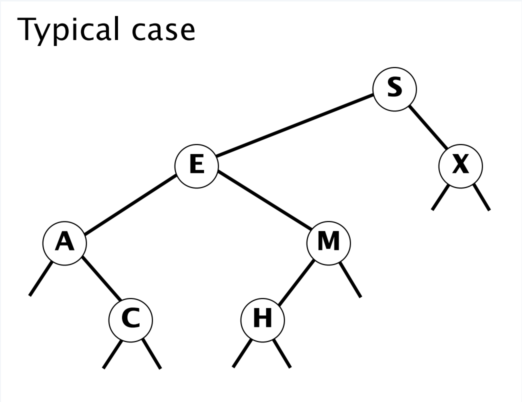


Key fact

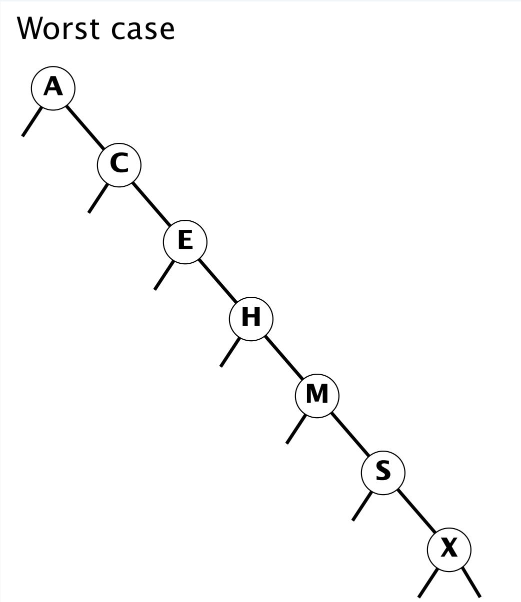
The shape of a BST depends on the order of insertion of the keys.



search cost guaranteed ~lg N



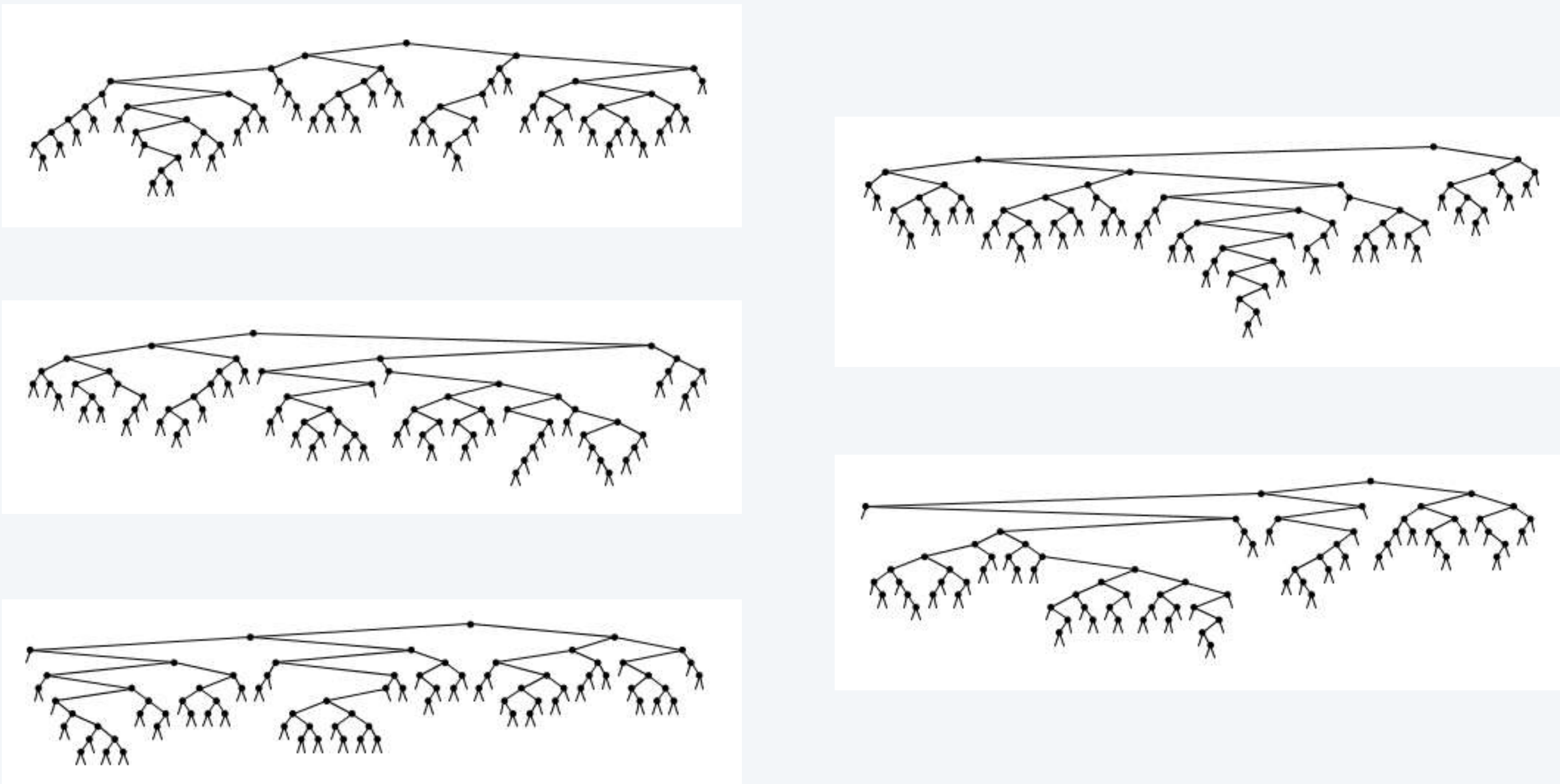
Average search cost ?



Average search cost ~N/2 (a problem)

Reasonable model: Analyze BST built from inserting keys in *random* order.

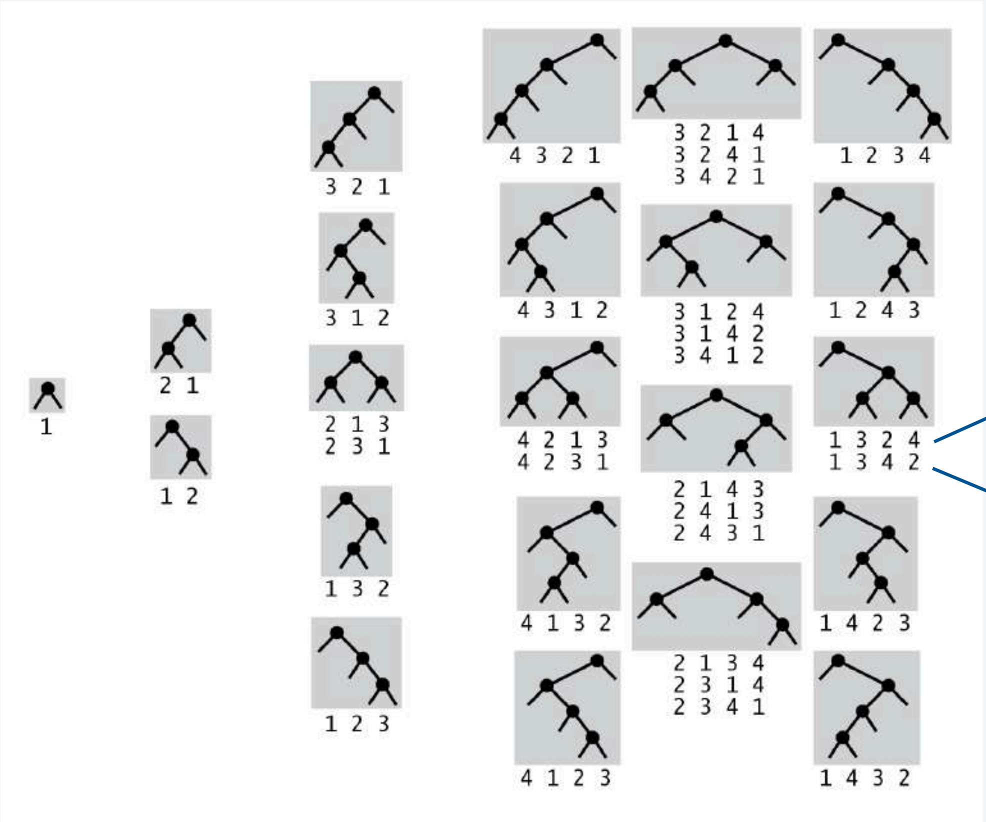
Typical random BSTs (80 nodes)

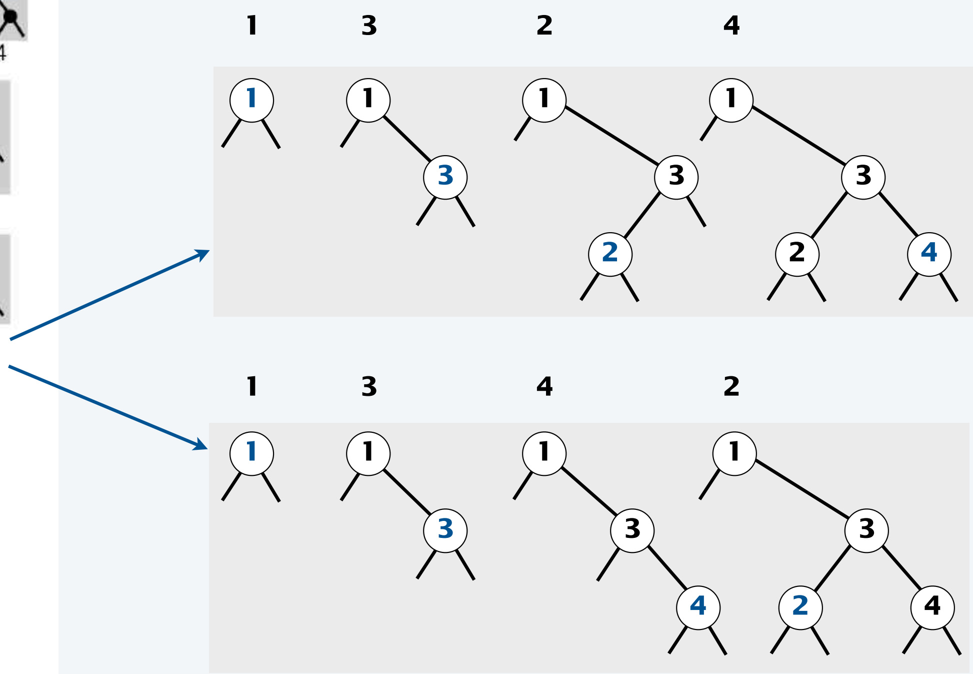


Challenge: characterize analytically (explain difference from random binary trees)

BST shape

is a property of *permutations*, not trees (!)



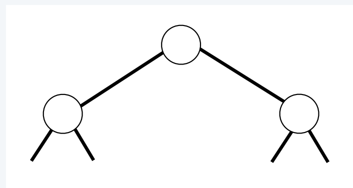


Note: Balanced shapes are more likely.

Mapping permutations to trees via BST insertion

Q. How many permutations map to this tree?

"result in this tree shape when inserted into an initially empty BST"

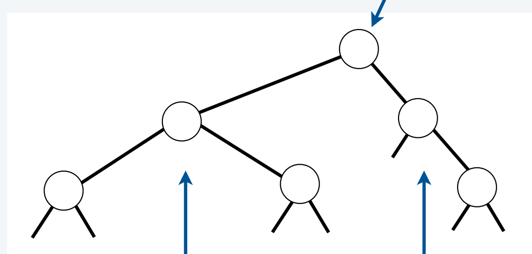


2	1	3
2	3	1

A. 2

Q. How many permutations map to *this* tree?

root must be 4



1, 2, and 3 on the left

5 and 6 on the right

ways to mix left and right

A. $\binom{5}{2} \cdot 2 \cdot 1 = 20$

perms mapping to left subtree

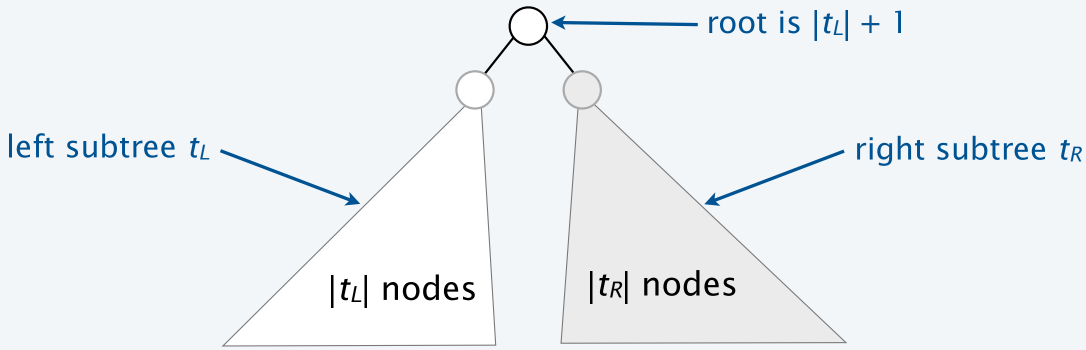
perms mapping to right subtree

4	2	1	3	5	6
4	2	1	5	3	6
4	2	1	5	6	3
4	2	5	1	3	6
4	2	5	1	6	3
4	2	5	6	1	3
4	5	2	1	3	6
4	5	2	1	6	3
4	5	2	6	1	3
4	5	6	2	1	3

4	2	3	1	5	6
4	2	3	5	1	6
4	2	3	5	6	1
4	2	5	3	1	6
4	2	5	3	6	1
4	2	5	6	3	1
4	5	2	3	1	6
4	5	2	3	6	1
4	5	2	6	3	1
4	5	6	2	3	1

Mapping permutations to trees via BST insertion

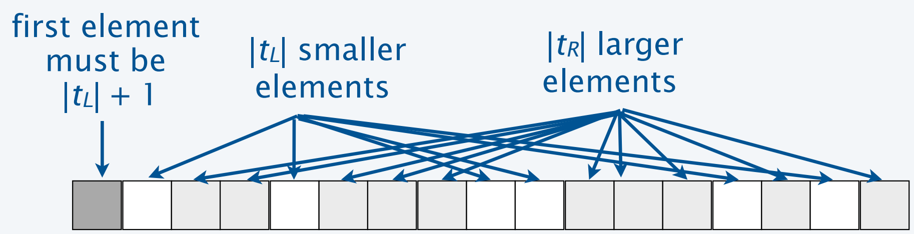
Q. How many permutations map to a general binary tree t ?



A. Let P_t be the number of perms that map to t

$$P_t = \binom{|t_L| + |t_R|}{|t_L|} \cdot P_{t_L} \cdot P_{t_R}$$

much, much larger when $t_L \approx t_R$ than when $t_L \ll t_R$
(explains why balanced shapes are more likely)

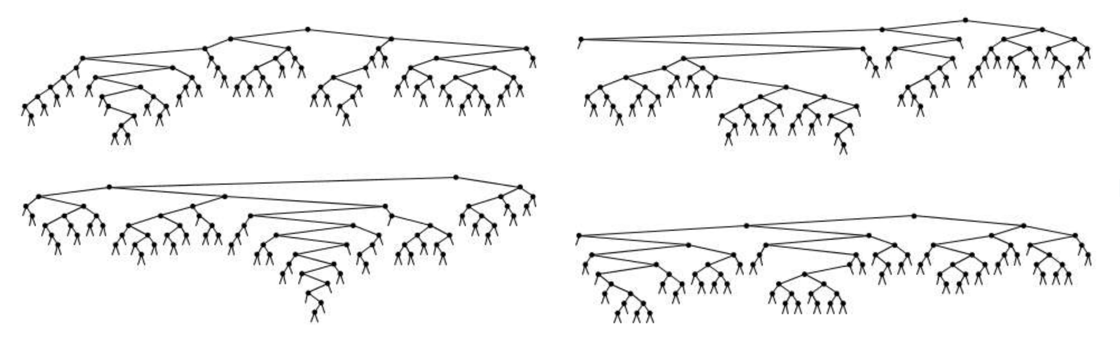


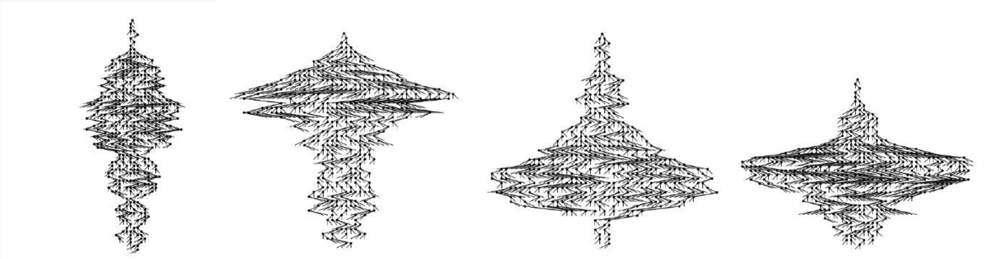
Two binary tree models

that are fundamental (and fundamentally different)

BST model

- Balanced shapes much more likely.
- Probability root is of rank k: $1/N$.





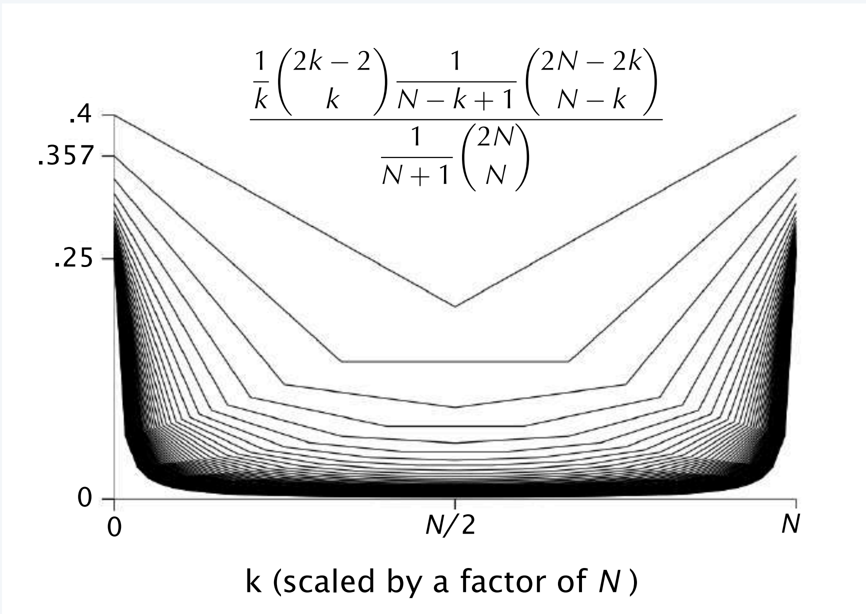
Catalan model

- Each tree shape equally likely.
- Probability root is of rank k:

$$\frac{\frac{1}{k}\binom{2k-2}{k}\frac{1}{N-k+1}\binom{2N-2k}{N-k}}{\frac{1}{N+1}\binom{2N}{N}}$$

Catalan distribution

Probability that the root is of rank k in a randomly-chosen binary tree with N nodes.




```
public static double[][] catalan(int N)
{
   double[] T = new double[N];
   double[][] cat = new double[N-1][];
   T[0] = 1;
   for (int i = 1; i < N; i++)
      T[i] = T[i-1]*(4*i-2)/(i+1);

   cat[0] = new double[1];
   cat[0][0] = 1;
   for (int i = 1; i < N-1; i++)
   {
      cat[i] = new double[i];
      for (int j = 0; j < i; j++)
         cat[i][j] = T[j]*T[i-j-1]/T[i];
   }
   return cat;
}
```

Note: Small subtrees are extremely likely.

Ex. Probability that at least one of the two subtrees is empty: ~1/2

Aside: Generating random binary trees

```
public class RandomBST
{
   private Node root;
   private int h;
   private int w;

   private class Node
   {
      private Node left, right;
      private int N;
      private int rank, depth;
   }

   public RandomBST(int N)
   {  root = generate(N, 0);  }

   private Node generate(int N, int d)
   { // See code at right. }

   public static void main(String[] args)
   {
      int N = Integer.parseInt(args[0]);
      RandomBST t = new RandomBST(N);
      t.show();
   }
}
```

stay tuned

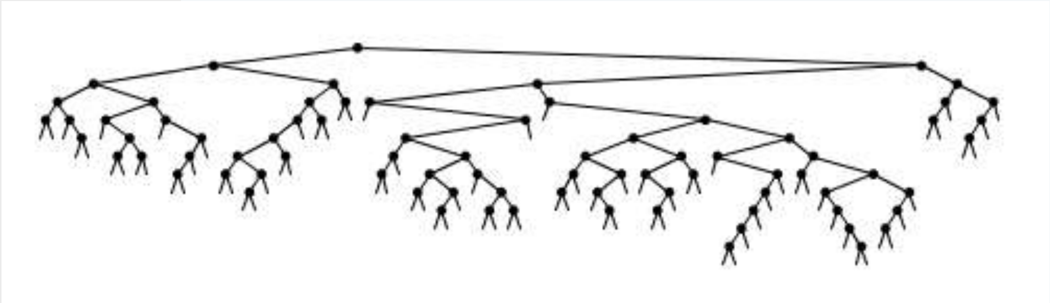
Note: “rank” field includes external nodes: x.rank = 2*k+1

```
private Node generate(int N, int d)
{
   Node x = new Node();
   x.N = N; x.depth = d;
   if (h < d) h = d;
   if (N == 0) x.rank = w++;   else
   {
      int k = // internal rank of root
      x.left = generate(k-1, d+1);
      x.rank = w++;
      x.right = generate (N-k, d+1);
   }
   return x;
}
```

random BST:	`StdRandom.uniform(N)+1`
random binary tree:	`StdRandom.discrete(cat[N]) + 1;`

Aside: Drawing binary trees

```
public void show()
{   show(root);   }

private double scaleX(Node t)
{   return 1.0*t.rank/(w+1);   }
private double scaleY(Node t)
{   return 3.0*(h - t.depth)/(w+1);   }

private void show(Node t)
{
   if (t.N == 0) return;
   show(t.left);
   show(t.right);
   double x = scaleX(t);
   double y = scaleY(t);
   double xl = scaleX(t.left);
   double yl = scaleY(t.left);
   double xr = scaleX(t.right);
   double yr = scaleY(t.right);
   StdDraw.filledCircle(x, y, .005);
   StdDraw.line(x, y, xl, yl);
   StdDraw.line(x, y, xr, yr);
}
```



Exercise: Implement “centered by level” approach.