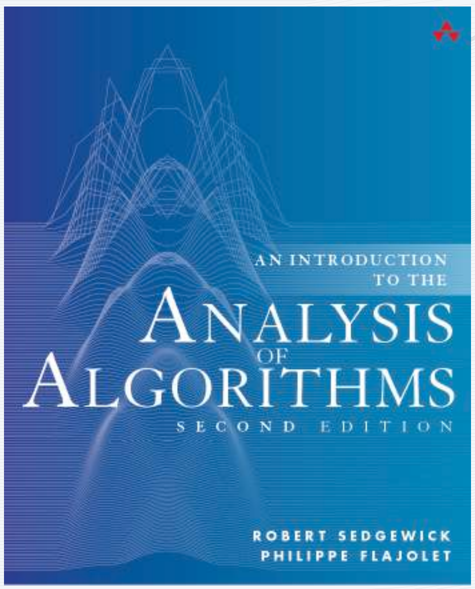
ANALYTIC COMBINATORICS

PART ONE

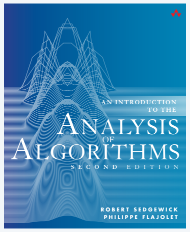


http://aofa.cs.princeton.edu

6. Trees

Review

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



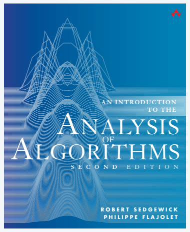
1	Analysis of Algorithms
2	Recurrences
3	Generating Functions
4	Asymptotics
5	Analytic Combinatorics

Note: Many applications beyond analysis of algorithms.

Orientation

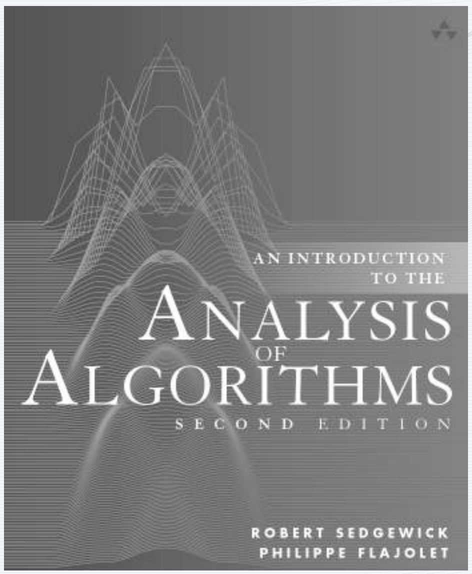
Second half of class

- 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.



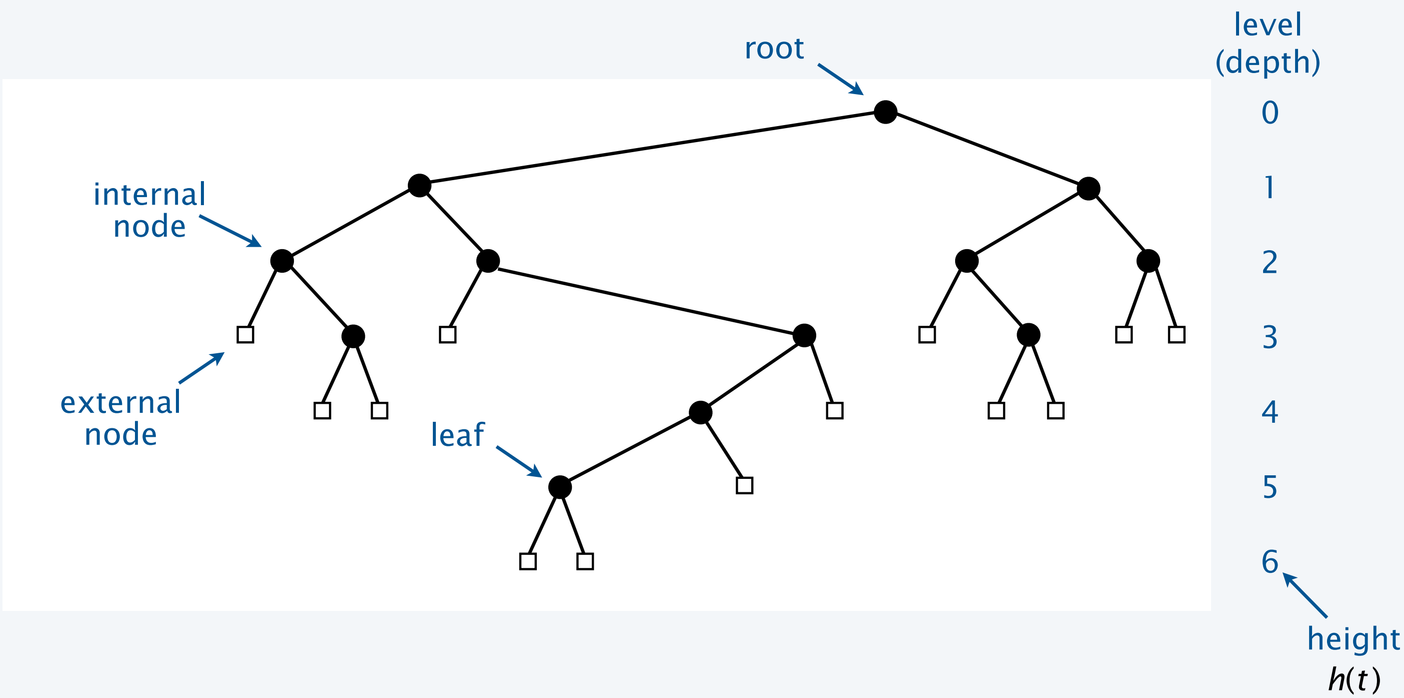
http://aofa.cs.princeton.edu

6. Trees

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

Anatomy of a binary tree

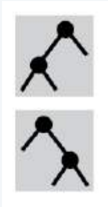
Definition. A *binary tree* is an external node or an internal node and two binary trees.



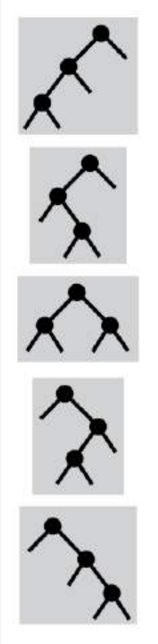
Binary tree enumeration (quick review)

How many binary trees with N nodes?

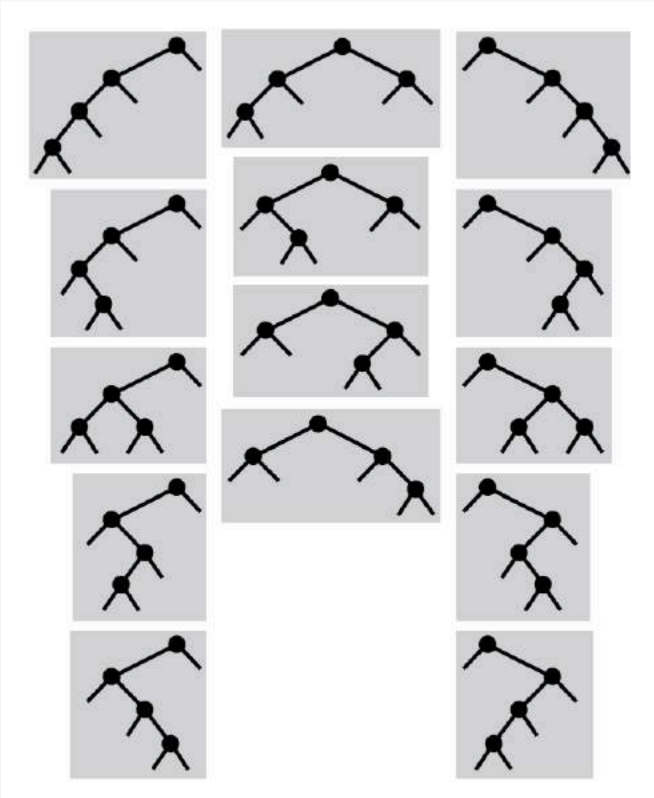
$T_1 = 1$



$T_2 = 2$



$T_3 = 5$



$T_4 = 14$

Symbolic method: binary trees

How many binary trees with N nodes?

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

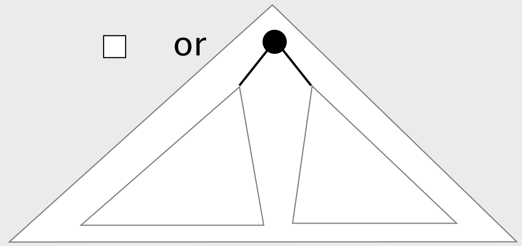
Atoms

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

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

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

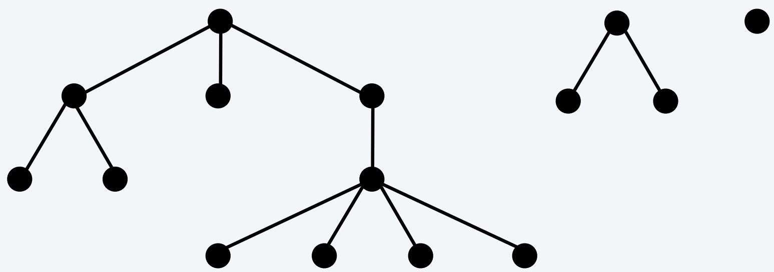
"a binary tree is an external node or an internal node connected to two binary trees"



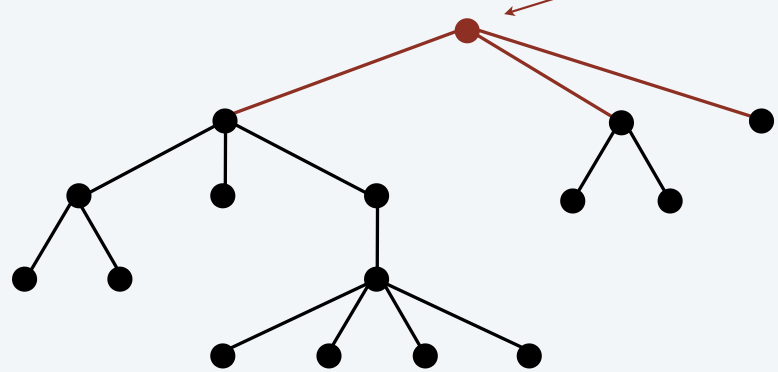
$$[z^N]T(z) = \frac{1}{N+1}\binom{2N}{N} \sim \frac{4^N}{\sqrt{\pi N^3}}$$

Forest and trees

Each forest with N nodes corresponds to



A tree with $N+1$ nodes



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

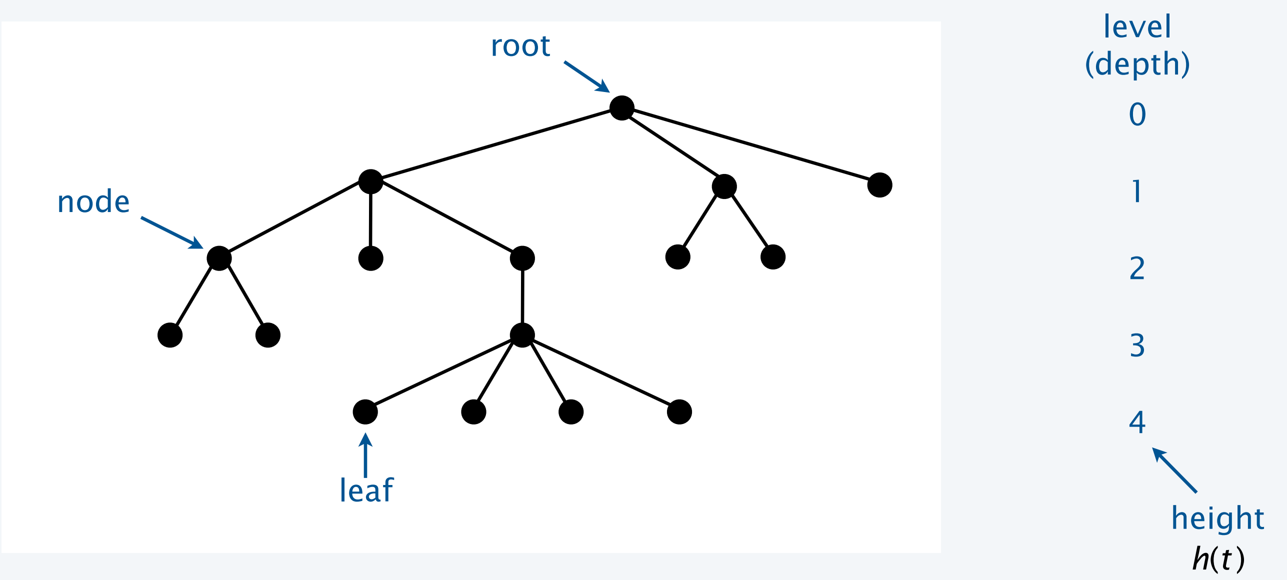
GF that enumerates forests

GF that enumerates trees

Anatomy of a (general) tree

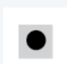
Definition. A *forest* is a sequence of disjoint trees.

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

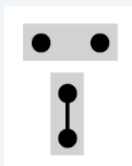


Forest enumeration

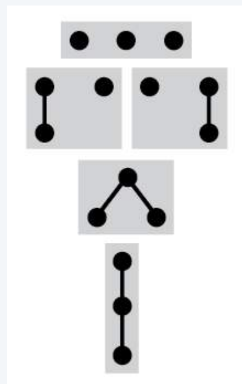
How many forests with N nodes?



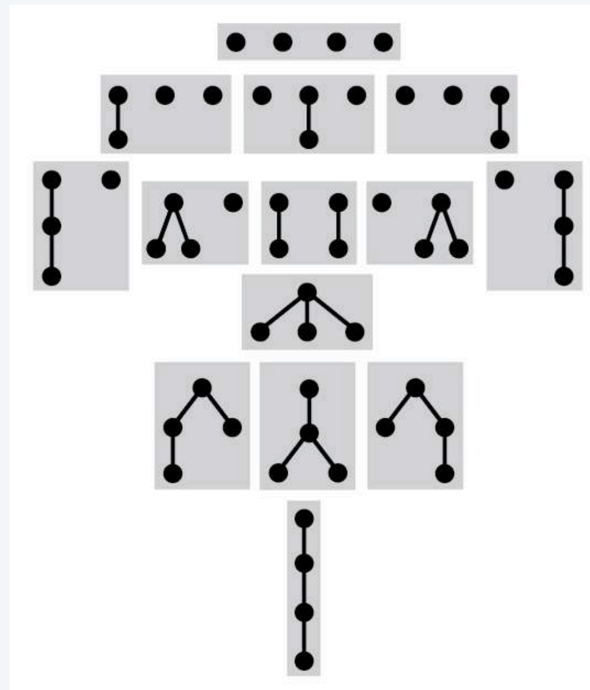
$F_1 = 1$



$F_2 = 2$



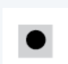
$F_3 = 5$



$F_4 = 14$

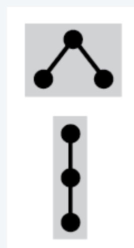
Tree enumeration

How many trees with N nodes?

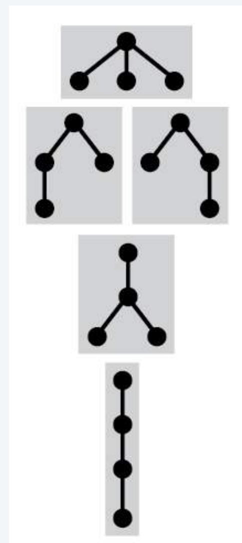


$G_1 = 1$

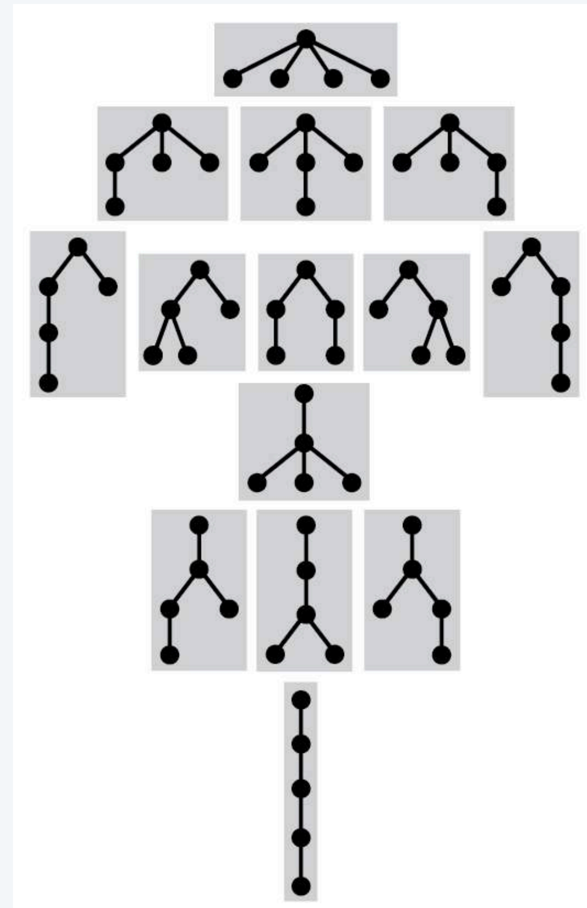
$G_2 = 1$



$G_3 = 2$



$G_3 = 5$



$G_4 = 14$

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

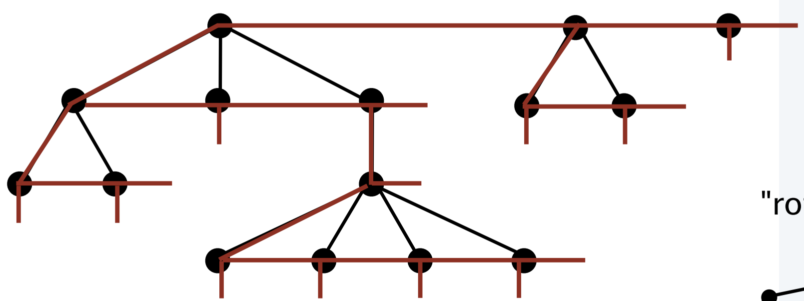
OGF equations
$$F(z) = \frac{1}{1 - G(z)} \quad \text{and} \quad G(z) = zF(z)$$

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

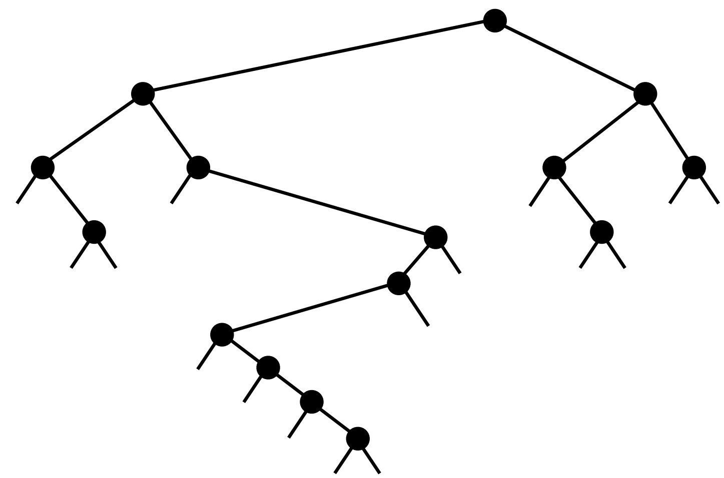
Each forest with N nodes corresponds to



Connect each node to its
- left child
- right sibling

"rotation" correspondence

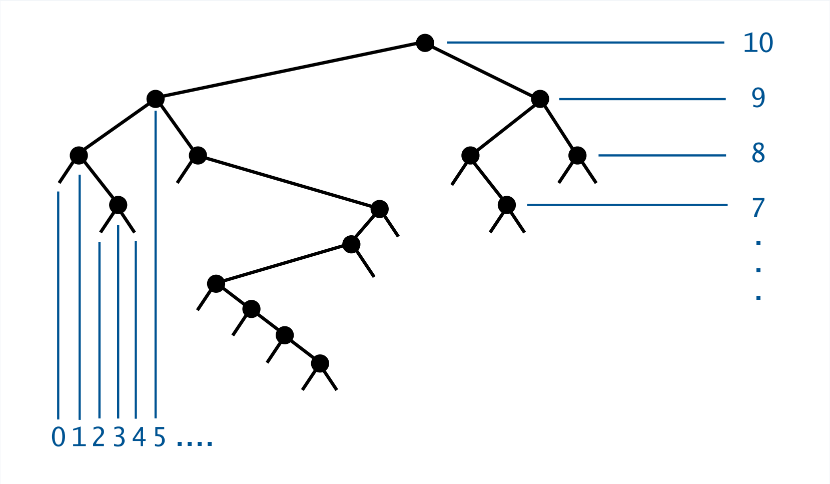
A binary tree with N nodes



Aside: Drawing a binary tree

Approach 1:

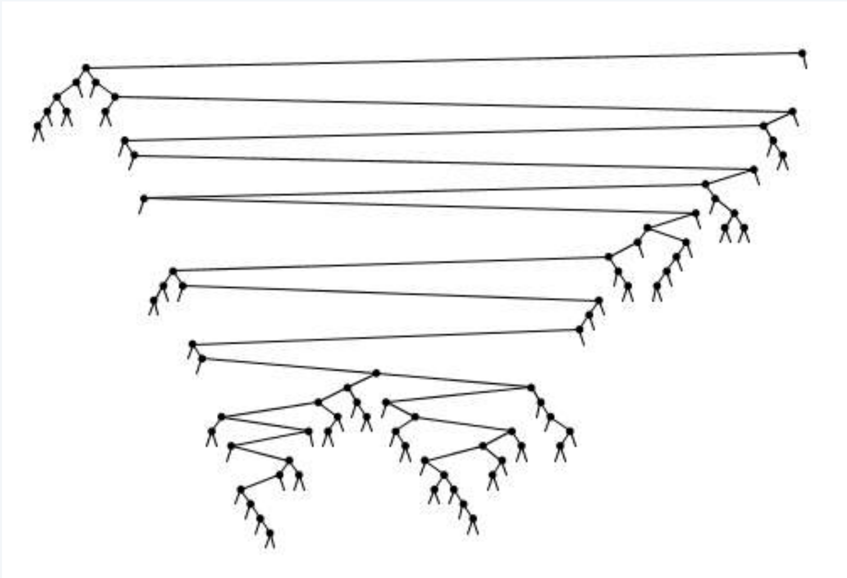
- y-coordinate: height minus node depth
- x-coordinate: inorder node rank



Design decision:

Reduce visual clutter by omitting external nodes

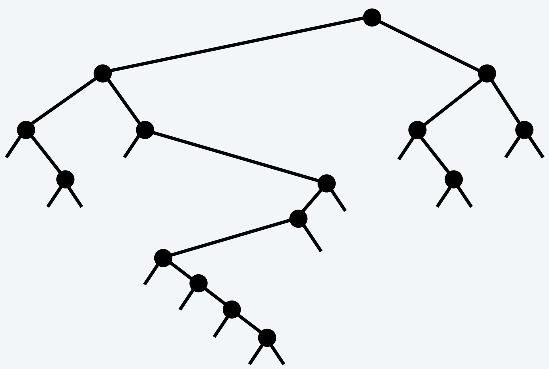
Problem: distracting long edges

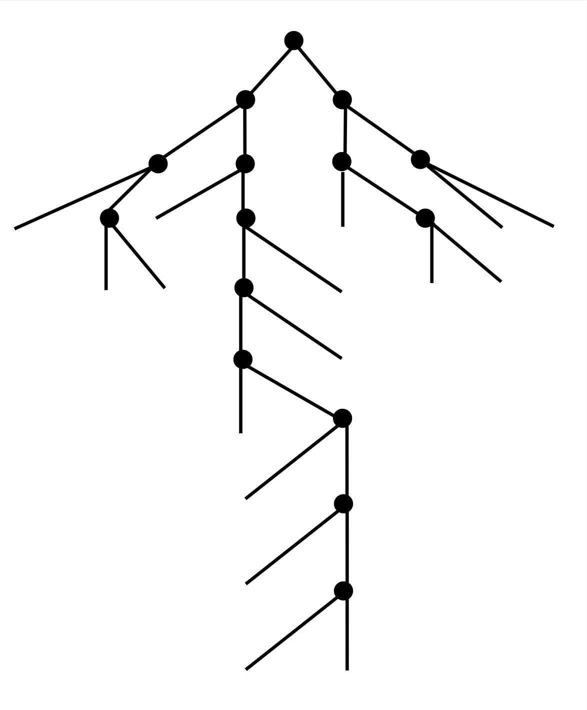


Aside: Drawing a binary tree

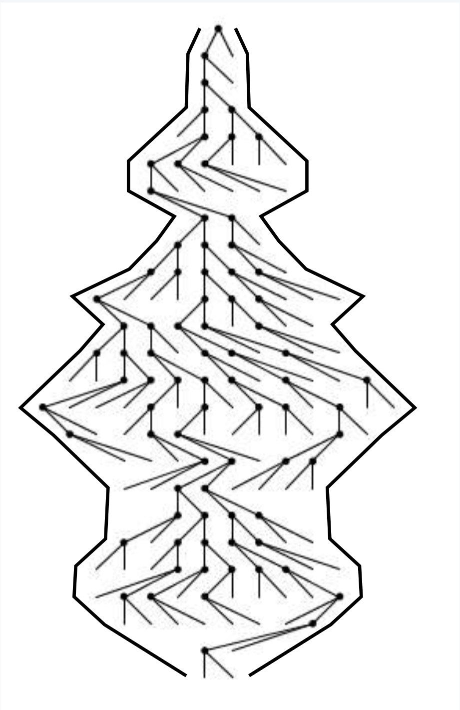
Approach 2:

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

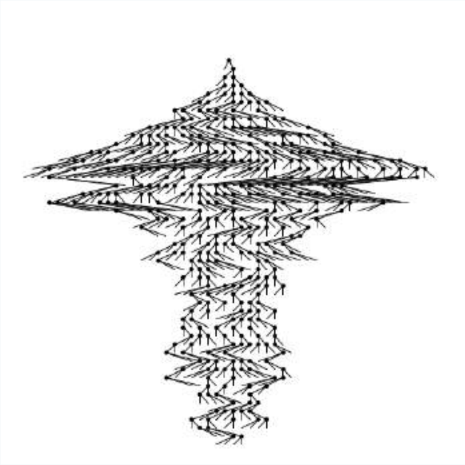


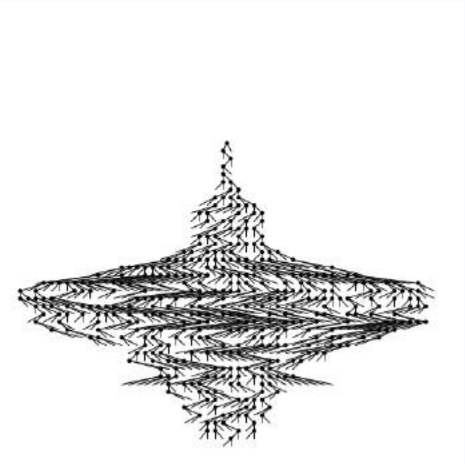


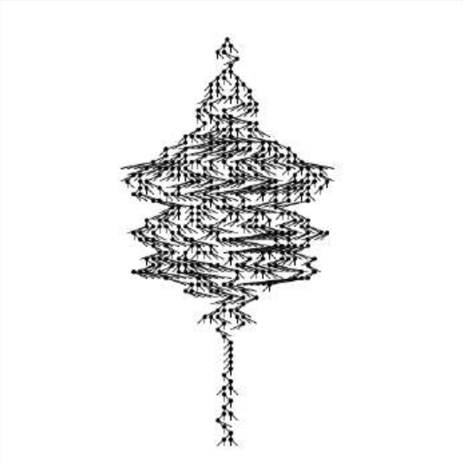
Drawing shows tree *profile*

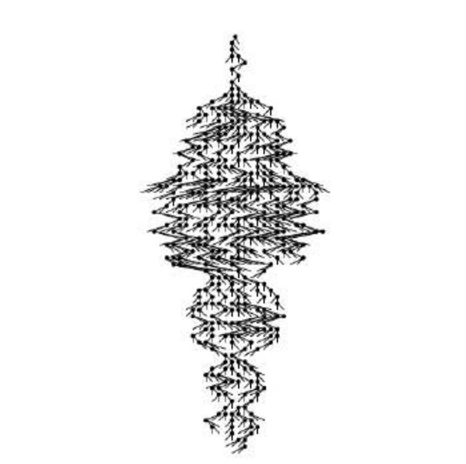


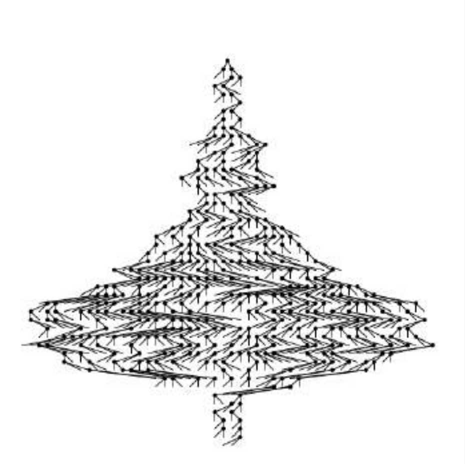
Typical random binary tree shapes (400 nodes)







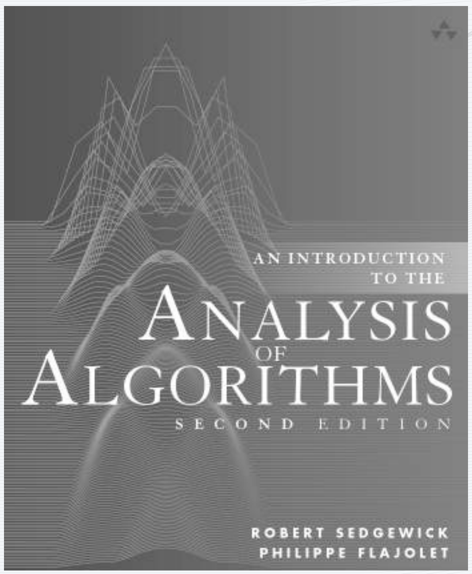




Challenge: characterize analytically

ANALYTIC COMBINATORICS

PART ONE



http://aofa.cs.princeton.edu

6. Trees

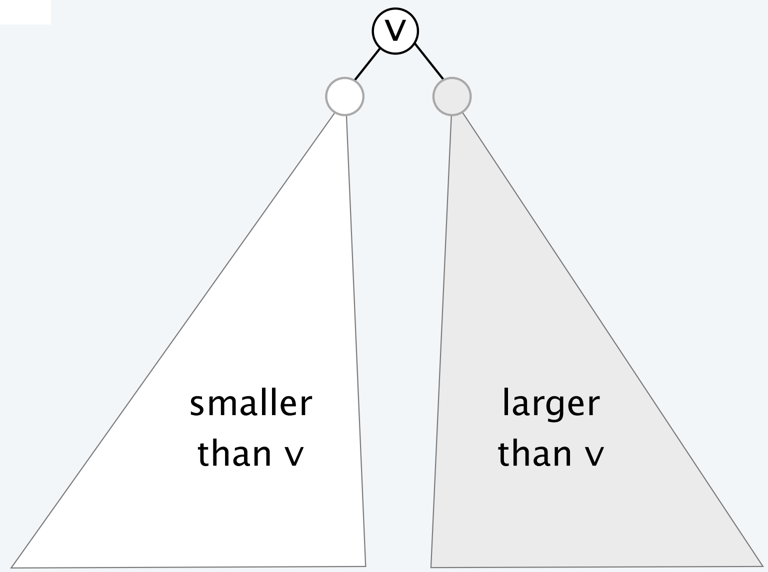
- Trees and forests
- **Binary search trees**
- Path length
- Other types of trees

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:

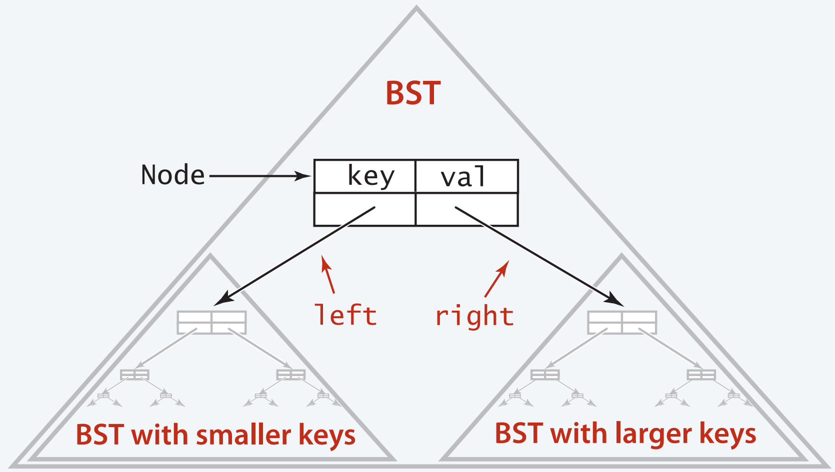
- A Key and a Value.
- A reference to the left and right subtree.

smaller keys   larger keys

Notes:

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

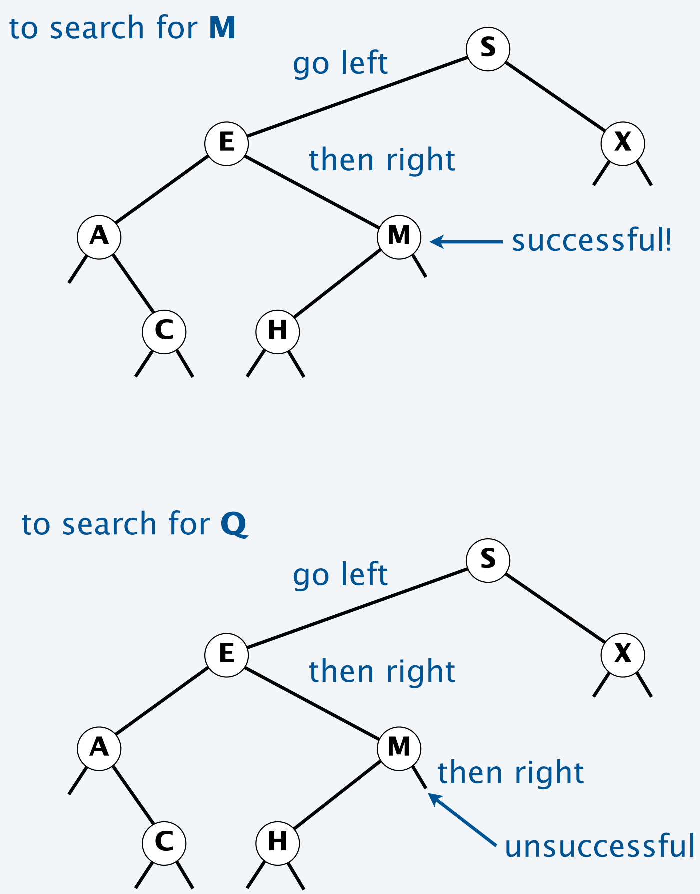
```
private class Node
{
   private Key key;
   private Value val;
   private Node left, right;
   public Node(Key key, Value val)
   {
      this.key = key;
      this.val = val;
   }
}
```



Binary search tree

BST implementation (search)

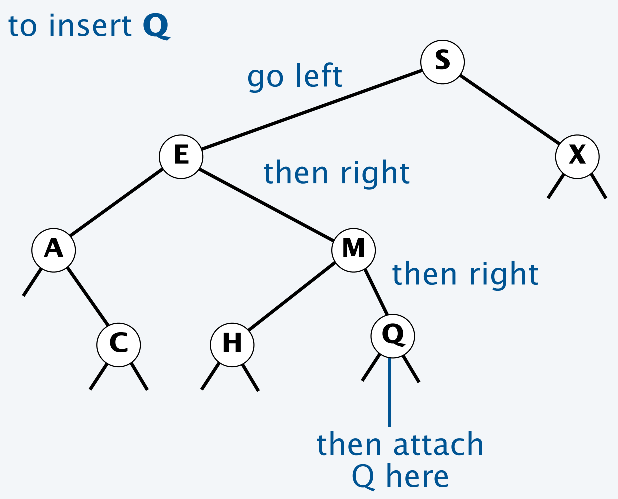
```
public class BST<Key extends Comparable<Key>, Value>
{
   private Node root;

   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)

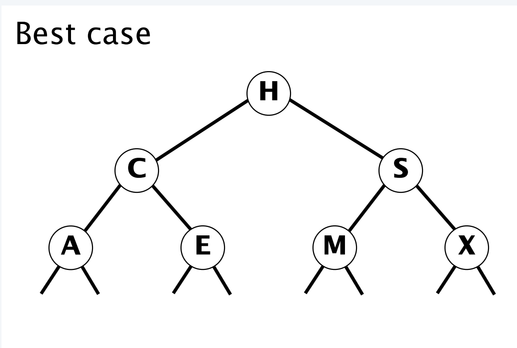
```
public void put(Key key, Value val)
{  root = put(root, key, val);  }

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

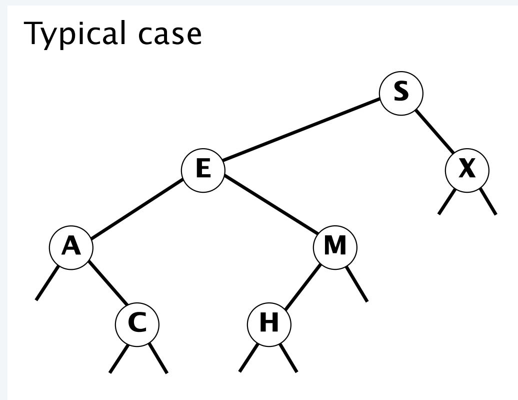


Key fact

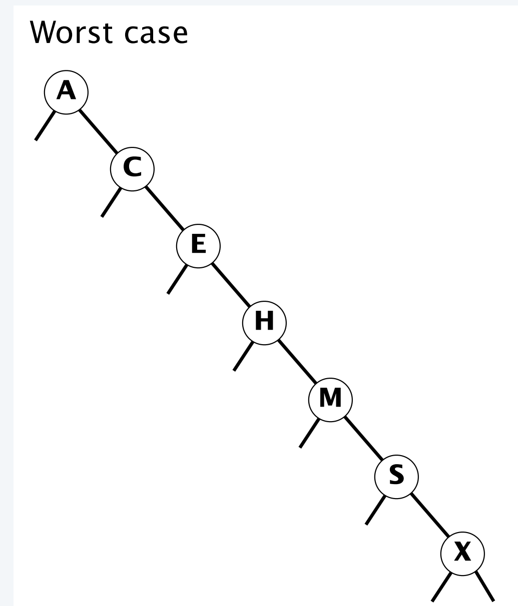
The shape of a BST depends on the order of insertion of the keys.



search cost guaranteed ~lg N



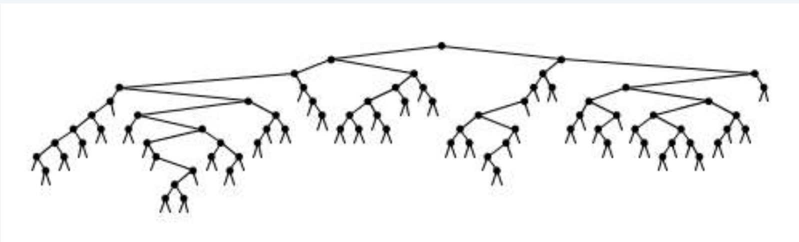
Average search cost ?

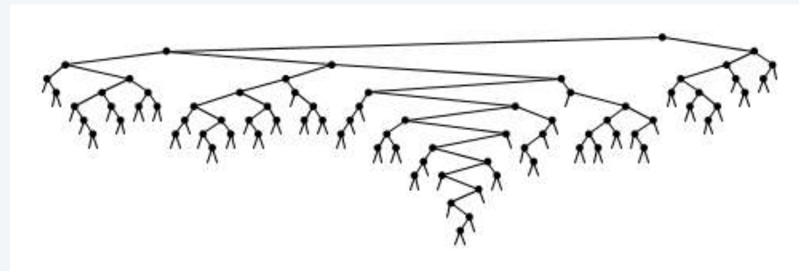


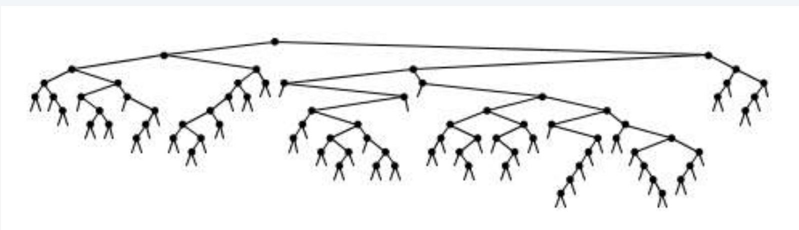
Average search cost ~N/2 (a problem)

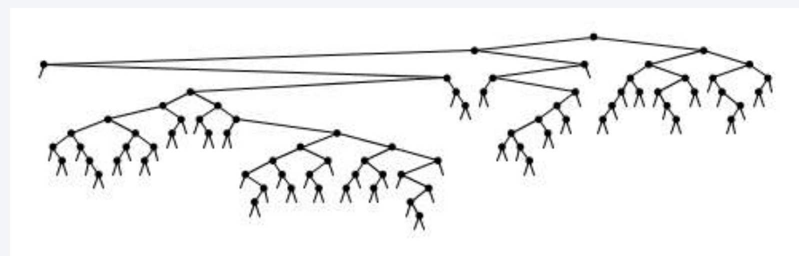
Reasonable model: Analyze BST built from inserting keys in *random* order.

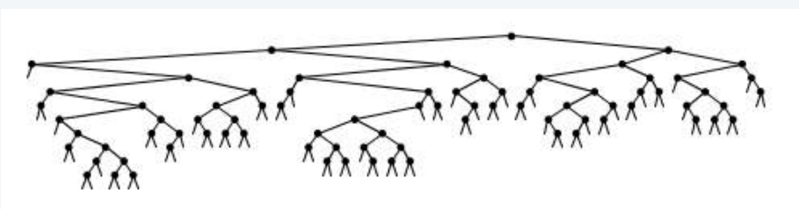
Typical random BSTs (80 nodes)







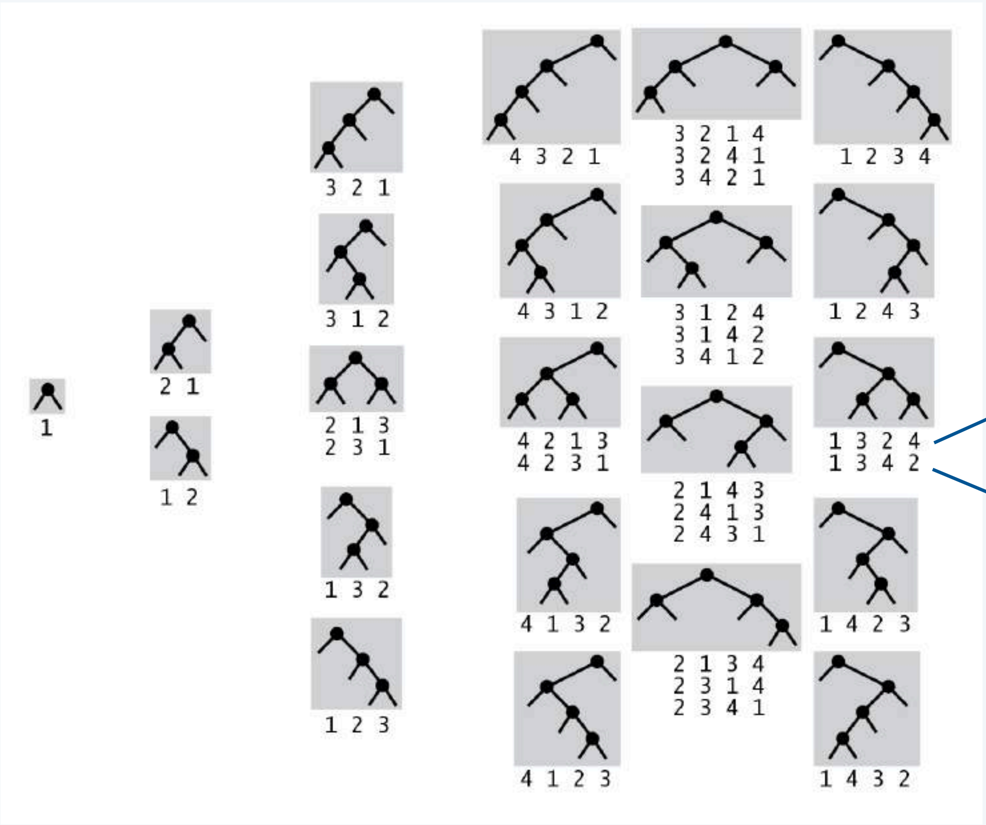


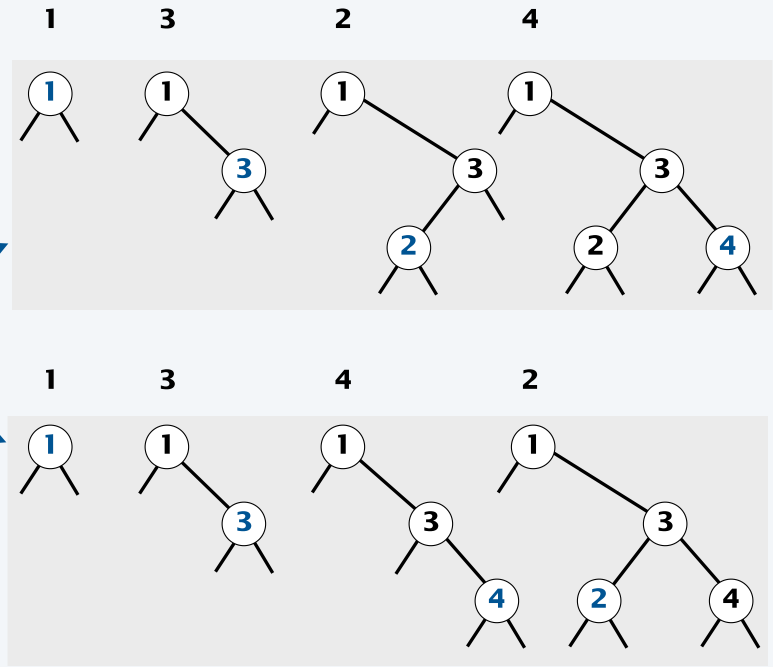


Challenge: characterize analytically (explain difference from random binary trees)

BST shape

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



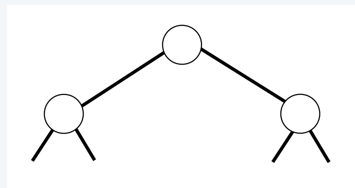


Note: Balanced shapes are more likely.

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"

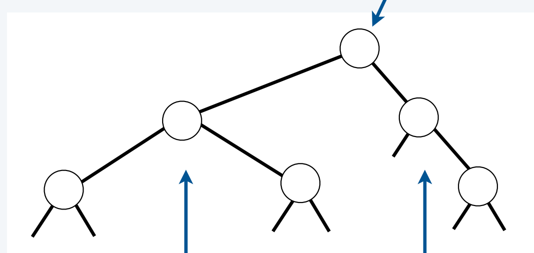


2	1	3
2	3	1

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

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

ways to mix left and right

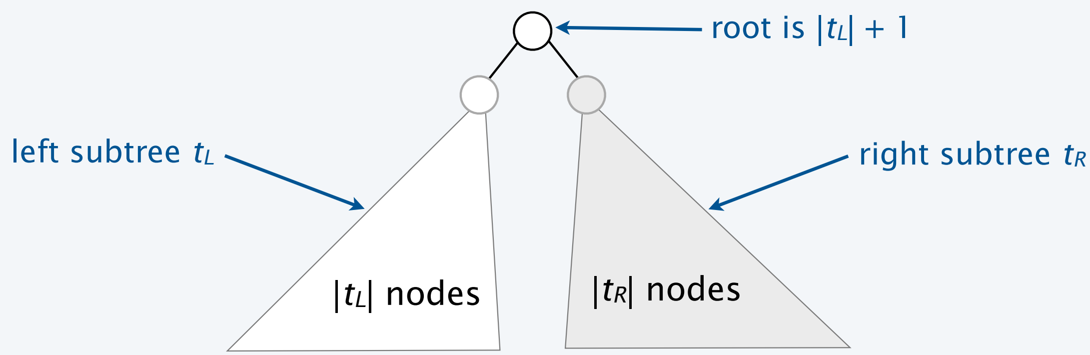
A. $\binom{5}{2} \cdot 2 \cdot 1 = 20$

perms mapping to left subtree

perms mapping to right subtree

Mapping permutations to trees via BST insertion

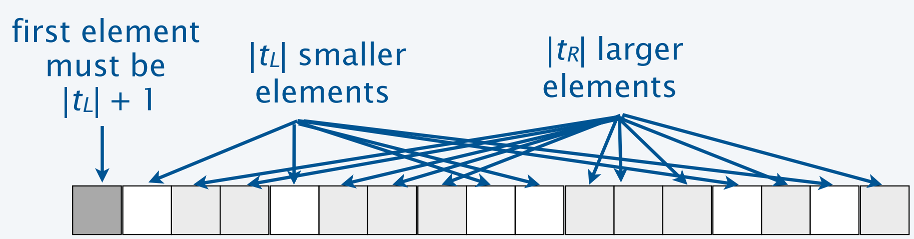
Q. How many permutations map to a general binary tree t ?



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

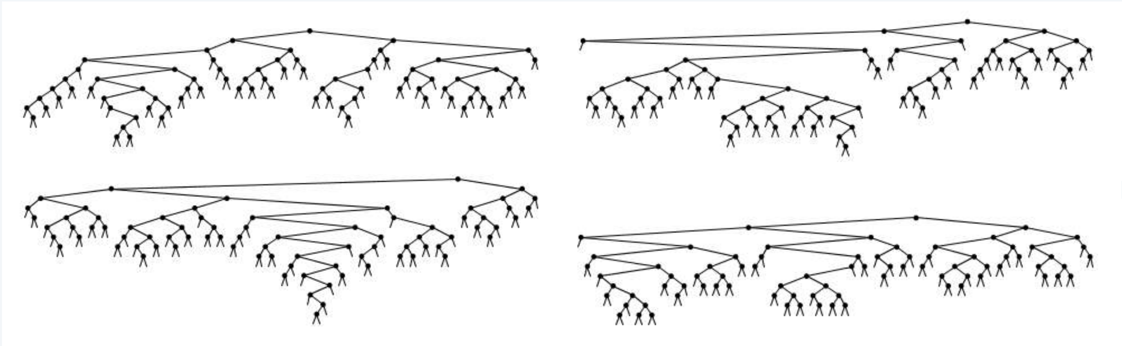
$$P_t = \binom{|t_L| + |t_R|}{|t_L|} \cdot P_{t_L} \cdot P_{t_R}$$

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



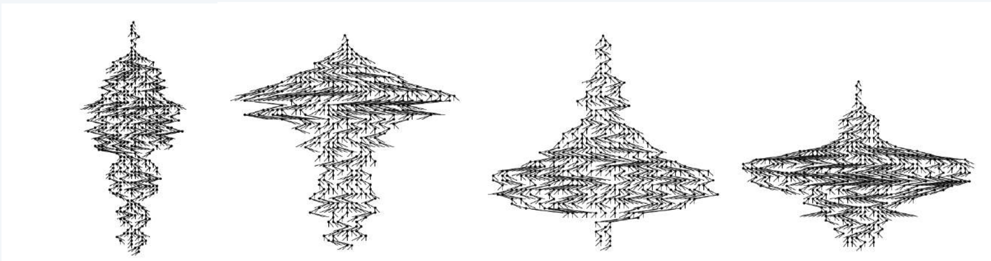
Two binary tree models

that are fundamental (and fundamentally different)



BST model

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



Catalan model

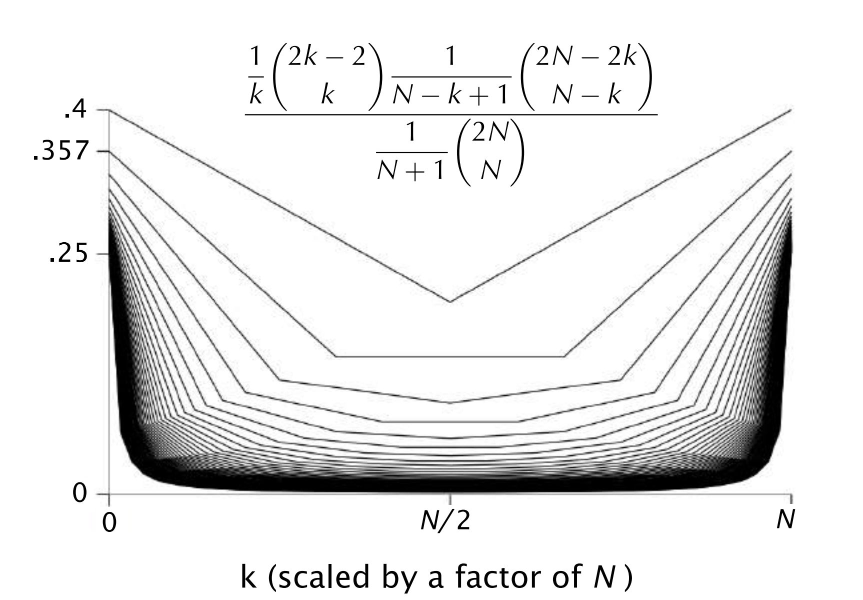
- Each tree shape equally likely.
- Probability root is of rank k:

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

k−1

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

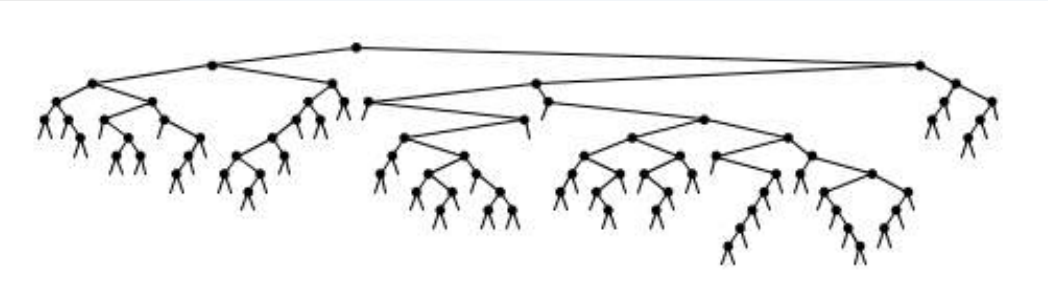
```
private Node generate(int N, int d)
{
   Node x = new Node();
   x.N = N; x.depth = d;
   if (h < d) h = d;
   if (N == 0) x.rank = w++;   else
   {
      int k = // internal rank of root
      x.left = generate(k-1, d+1);
      x.rank = w++;
      x.right = generate (N-k, d+1);
   }
   return x;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

random BST: `StdRandom.uniform(N)+1`

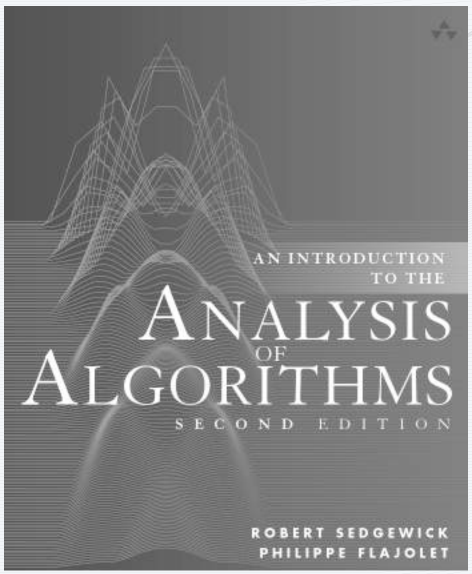
random binary tree: `StdRandom.discrete(cat[N]) + 1;`

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.



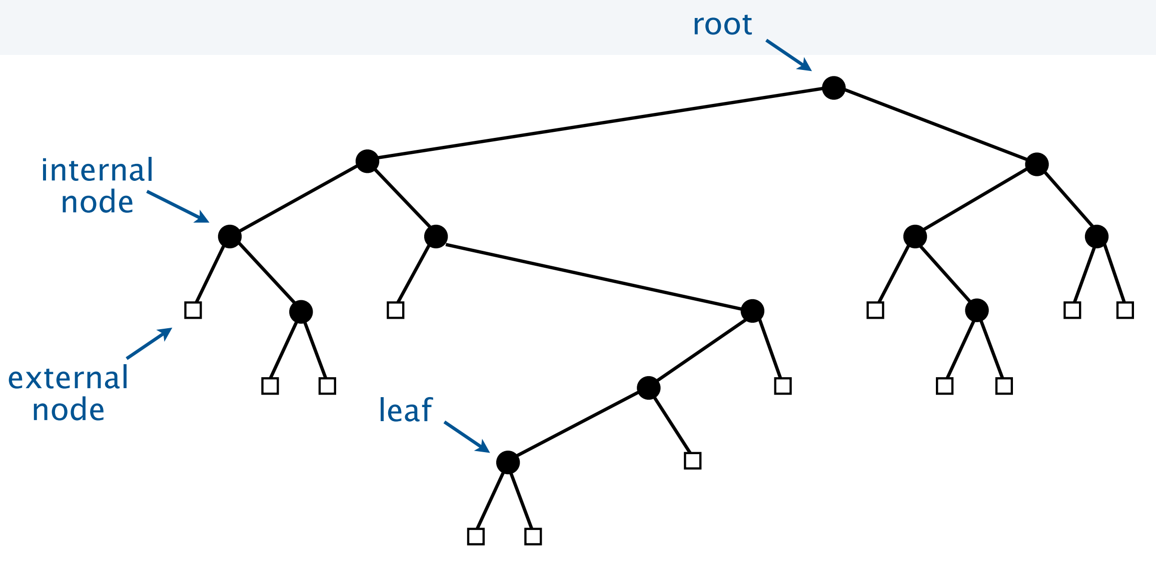
http://aofa.cs.princeton.edu

6. Trees

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

Path length in binary trees

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



level (depth): 0, 1, 2, 3, 4, 5, 6 ← height $h(t)$

internal path length $ipl(t)$: $0 \cdot 1 + 1 \cdot 2 + 2 \cdot 4 + 3 \cdot 3 + 4 \cdot 1 + 5 \cdot 1 = 28$

external path length $xpl(t)$: $0 \cdot 0 + 1 \cdot 0 + 2 \cdot 0 + 3 \cdot 5 + 4 \cdot 5 + 5 \cdot 1 + 6 \cdot 2 = 52$

internal path length: $$ipl(t) = \sum_{k \geq 0} k \cdot \{\#\text{ internal nodes at depth k}\}$$

external path length: $$xpl(t) = \sum_{k \geq 0} k \cdot \{\#\text{ external nodes at depth k}\}$$

Path length in binary trees

notation	*definition*
t	binary tree
$\lvert t \rvert$	# internal nodes in t
$\boxed{t}$	# external nodes in t
t_L and t_R	left and right subtrees of t
$ipl\,(t)$	internal path length of t
$xpl\,(t)$	external path length of t

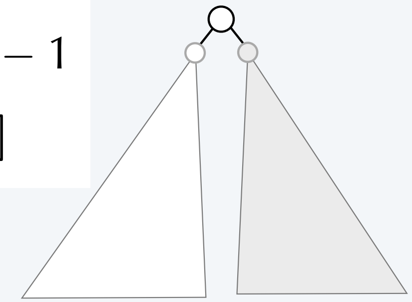
recursive relationships

$$|\,t\,| = |\,t_L\,| + |\,t_R\,| + 1$$

$$\boxed{t} = \boxed{t_L} + \boxed{t_R}$$

$$ipl(t) = ipl(t_L) + ipl(t_R) + |\,t\,| - 1$$

$$xpl(t) = xpl(t_L) + xpl(t_R) + \boxed{t}$$



Lemma 1. $\boxed{t} = |\,t\,| + 1$

Proof. Induction.

$$\boxed{t} = \boxed{t_L} + \boxed{t_R}$$

$$= |\,t_L\,| + 1 + |\,t_R\,| + 1$$

$$= |\,t\,| + 1$$

Lemma 2. $xpl(t) = ipl(t) + 2|\,t\,|$

Proof. Induction.

$$xpl(t) = xpl(t_L) + xpl(t_R) + \boxed{t}$$

$$= ipl(t_L) + 2|\,t_L\,| + ipl(t_R) + 2|\,t_R\,| + |\,t\,| + 1$$

$$= ipl(t) + 2|\,t\,|$$

Problem 1: What is the expected path length of a random binary tree?

Q_{Nk} = # trees with N nodes and ipl k

T_N = # trees

Q_N = cumulated cost (total ipl)

$Q_{10} = 1$

$T_1 = 1$

$Q_1 = 0$

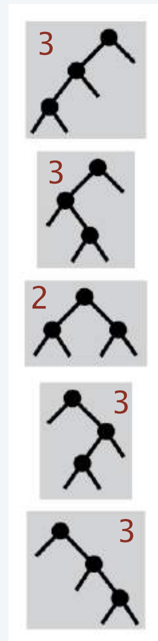
$Q_1 / T_1 = 0$

$Q_{21} = 2$

$T_2 = 2$

$Q_2 = 2$

$Q_2 / T_2 = 1$



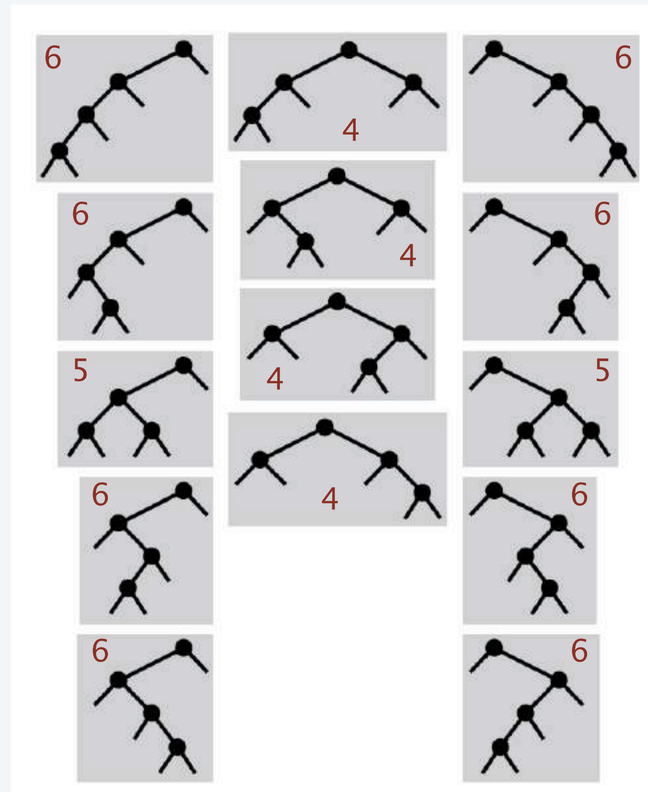
$Q_{32} = 1$

$Q_{33} = 4$

$T_3 = 2$

$Q_3 = 1 \cdot 2 + 4 \cdot 3 = 14$

$Q_3 / T_3 = 2.8$



$Q_{44} = 4$

$Q_{45} = 2$

$Q_{46} = 8$

$T_4 = 14$

$Q_4 = 4 \cdot 4 + 2 \cdot 5 + 8 \cdot 6 = 74$

$Q_4 / T_4 \doteq 5.286$

Average path length in a random binary tree

T is the set of all binary trees.

$|t|$ is the number of internal nodes in t.

$\mathrm{ipl}(t)$ is the internal path length of t.

T_N is the # of binary trees of size N (Catalan).

Q_N is the total ipl of all binary trees of size N.

Counting GF.

$$T(z) = \sum_{t \in T} z^{|t|} = \sum_{N \geq 0} T_N z^N = \sum_{N \geq 0} \frac{1}{N+1}\binom{2N}{N} z^N \sim \frac{4^N}{\sqrt{\pi N^3}}$$

Cumulative cost GF.

$$Q(z) = \sum_{t \in T} \mathrm{ipl}(t) z^{|t|}$$

Average ipl of a random N-node binary tree.

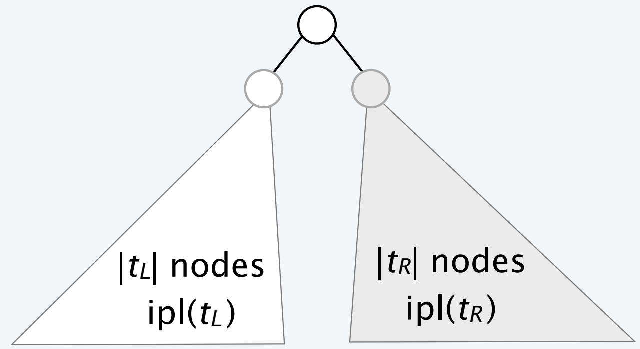
$$\frac{[z^N]Q(z)}{[z^N]T(z)} = \frac{[z^N]Q(z)}{T_N}$$

Next: Derive a functional equation for the CGF.

CGF functional equation for path length in binary trees

Counting GF. $$T(z) = \sum_{t \in T} z^{|t|}$$

CGF. $$Q(z) = \sum_{t \in T} ipl(t) z^{|t|}$$



$$ipl(t) = ipl(t_L) + ipl(t_R) + |\, t_L \,| + |\, t_R \,|$$

empty tree

root

Decompose from definition. $$Q(z) = 1 + \sum_{t_L \in T} \sum_{t_R \in T} \big(ipl(t_L) + ipl(t_R) + |t_L| + |t_R|\big) z^{|t_L| + |t_R| + 1}$$

0

$$\sum_{t_L \in T} ipl(t_L) z^{|t_L|} \sum_{t_R \in T} z^{|t_R|} = Q(z)T(z)$$

$$\sum_{t_L \in T} |t_L| z^{|t_L|} \sum_{t_R \in T} z^{|t_R|} = zT'(z)T(z)$$

$$= 1 + 2zQ(z)T(z) + 2z^2 T'(z)T(z)$$

0

Expected path length of a random binary tree: full derivation

CGF.

$$Q(z) = \sum_{t \in T} ipl(t) z^{|t|}$$

Decompose from definition.

$$Q(z) = 1 + \sum_{t_L \in T} \sum_{t_R \in T} \big(ipl(t_L) + ipl(t_R) + |t_L| + |t_R|\big) z^{|t_L|+|t_R|+1}$$

0

$$= 2zT(z)\big(Q(z) + zT'(z)\big)$$

Solve.

$$Q(z) = \frac{2z^2 T(z) T'(z)}{1 - 2zT(z)}$$

$$T(z) = \frac{1 - \sqrt{1-4z}}{2z} \qquad T_N \sim \frac{4^N}{N\sqrt{\pi N}}$$

$$T'(z) = -\frac{1 - \sqrt{1-4z}}{2z^2} + \frac{1}{z\sqrt{1-4z}}$$

$$1 - 2zT(z) = \sqrt{1-4z}$$

Do some algebra (omitted)

$$zQ(z) = \frac{z}{1-4z} - \frac{1-z}{\sqrt{1-4z}} + 1$$

Expand.

$$Q_N \equiv [z^N]Q(z) \sim 4^N$$

Compute *average internal path length.*

$$Q_N/T_N \sim N\sqrt{\pi N}$$

Problem 2: What is the expected path length of a random *BST*?

C_{Nk} = # *permutations* resulting in a BST with N nodes and ipl k

$N!$ = # permutations

C_N = cumulated cost (total ipl)

$C_{10} = 1$

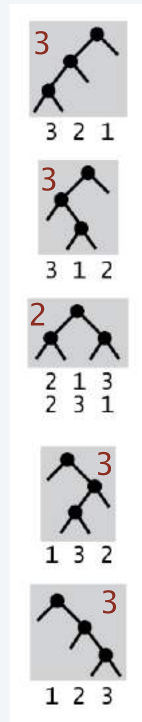
$C_1 = 0$

$C_1/1! = 0$

$C_{21} = 2$

$C_2 = 2$

$C_2/2! = 1$

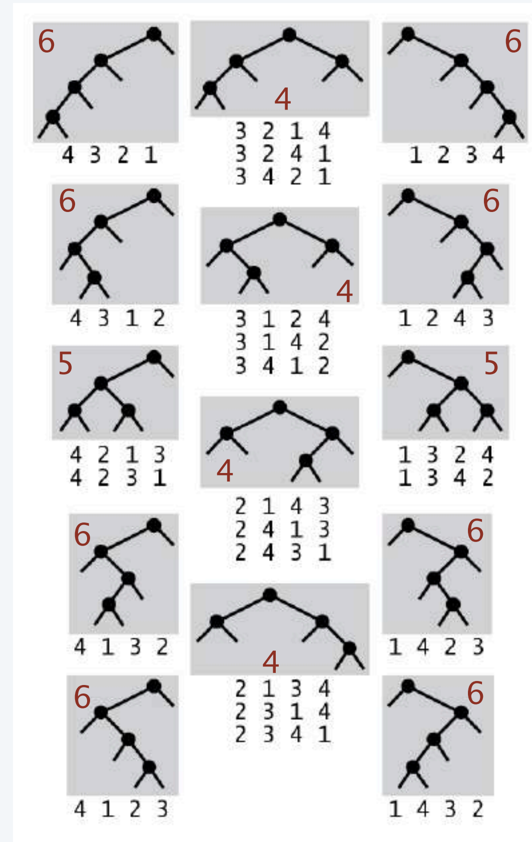


$C_{32} = 2$

$C_{33} = 4$

$C_3 = 2\cdot2 + 4\cdot3 = 16$

$C_3/3! \doteq 2.667$



$C_{44} = 12$

$C_{45} = 4$

$C_{46} = 8$

$C_4 = 12\cdot4 + 4\cdot5 + 8\cdot6 = 74$

$C_4/4! \doteq 4.833$

Recall: A property of permutations.

Average path length in a BST built from a random permutation

P is the set of all permutations.

$|p|$ is the length of p.

ipl(p) is the ipl of the BST built from p by inserting into an initially empty tree.

P_N is the # of permutations of size N (N!).

C_N is the total ipl of BSTs built from all permutations.

Counting EGF.

$$P(z) = \sum_{p \in P} \frac{z^{|p|}}{|p|!} = \sum_{N \geq 0} N! \frac{z^N}{N!} = \frac{1}{1-z}$$

Cumulative cost EGF.

$$C(z) = \sum_{p \in P} \mathrm{ipl}(p) \frac{z^{|p|}}{|p|!}$$

Expected ipl of a BST built from a random permutation.

$$\frac{N![z^N]C(z)}{[z^N]P(z)} = \frac{N![z^N]C(z)}{N!} = [z^N]C(z) \longleftarrow$$

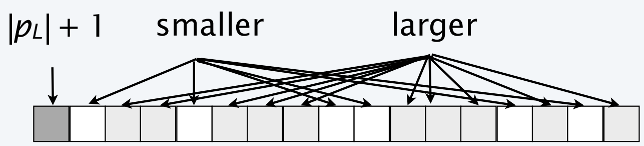
skip a step because counting sequence and EGF normalization are both N!

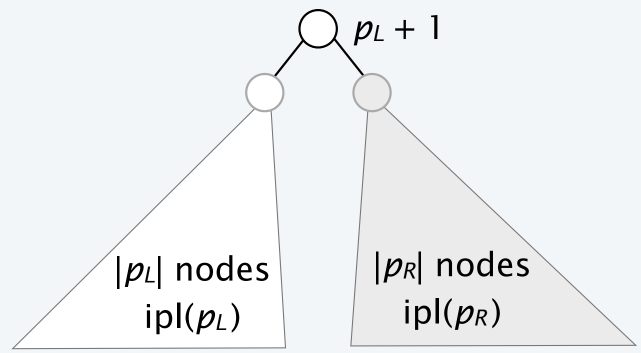
Next: Derive a functional equation for the cumulated cost EGF.

CGF functional equation for path length in BSTs

Cumulative cost EGF. $$C(z) = \sum_{p\in P} \mathrm{ipl}(p)\frac{z^{|p|}}{|p|!}$$

Counting GF. $$P(z) = \sum_{p\in P}\frac{z^{|p|}}{|p|!} = \frac{1}{1-z}$$





$\binom{|p_L|+|p_R|}{|p_L|}$ perms lead to the same tree with
- $|p_L|$ +1 at the root
- p_L nodes on the left
- p_R nodes on the right

Decompose. $$C(z) = \sum_{p_L\in\mathcal{P}}\sum_{p_R\in\mathcal{P}} \binom{|p_L|+|p_R|}{|p_L|}\frac{z^{|p_L|+|p_R|+1}}{(|p_L|+|p_R|+1)!}\big(ipl(p_L)+ipl(p_R)+|p_L|+|p_R|\big)$$

Differentiate. $$C'(z) = \sum_{p_L\in\mathcal{P}}\sum_{p_R\in\mathcal{P}} \frac{z^{|p_L|}}{|p_L|!}\frac{z^{|p_R|}}{|p_R|!}\big(ipl(p_L)+ipl(p_R)+|p_L|+|p_R|\big)$$

Tricky; often works with perms

$$= 2C(z)P(z) + 2zP'(z)P(z) = \frac{2C(z)}{1-z} + \frac{2z}{(1-z)^3}$$

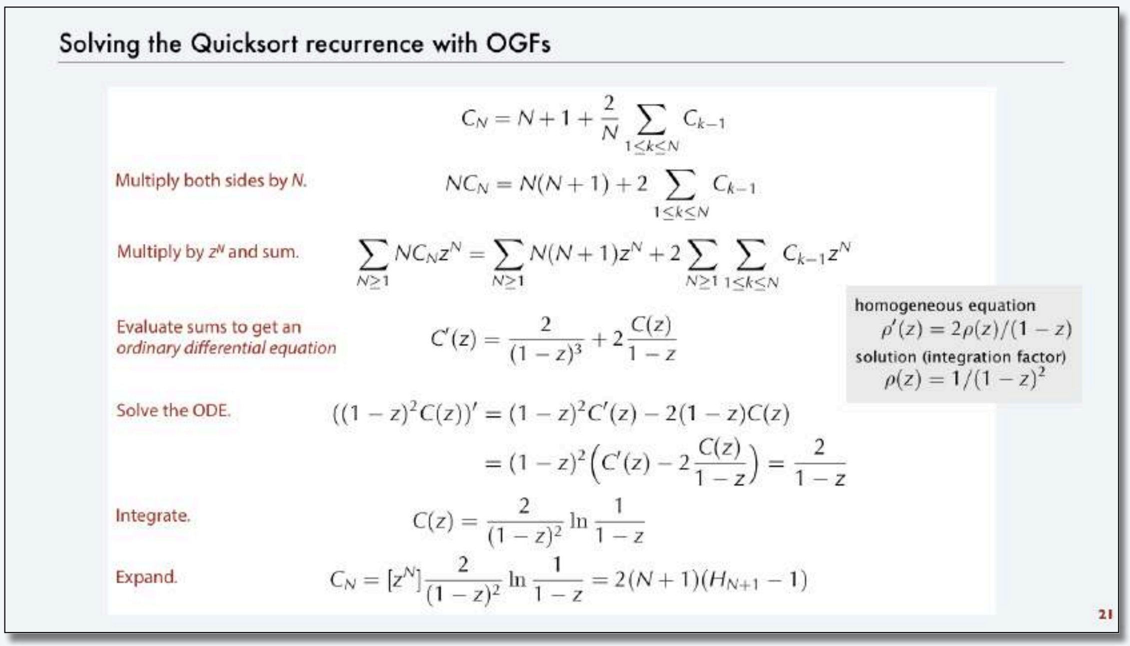
$$P(z) = \sum_{p\in P}\frac{z^{|p|}}{|p|!} = \frac{1}{1-z}$$

$$P'(z) = \sum_{p\in P}\frac{z^{|p|-1}}{(|p|-1)!} = \frac{1}{(1-z)^2}$$

CGF functional equation for path length in BSTs

$$C'(z) = \frac{2C(z)}{1-z} + \frac{2z}{(1-z)^3}$$

Look familiar?



Expected path length in BST built from a random permutation: full derivation

CGF.

$$C(z) = \sum_{p \in P} \mathrm{ipl}(p) \frac{z^{|p|}}{|p|!}$$

Decompose.

$$C(z) = \sum_{p_L \in \mathcal{P}} \sum_{p_R \in \mathcal{P}} \binom{|p_L| + |p_R|}{|p_L|} \frac{z^{|p_L|+|p_R|+1}}{(|p_L| + |p_R| + 1)!} (ipl(p_L) + ipl(p_R) + |p_L| + |p_R|)$$

Differentiate.

$$C'(z) = \sum_{p_L \in \mathcal{P}} \sum_{p_R \in \mathcal{P}} \frac{z^{|p_L|}}{|p_L|!} \frac{z^{|p_R|}}{|p_R|!} (ipl(p_L) + ipl(p_R) + |p_L| + |p_R|)$$

$$P(z) = \sum_{p \in P} \frac{z^{|p|}}{|p|!} = \frac{1}{1-z}$$

$$P'(z) = \sum_{p \in P} \frac{z^{|p|-1}}{(|p|-1)!} = \frac{1}{(1-z)^2}$$

Simplify.

$$= 2C(z)P(z) + 2zP'(z)P(z)$$

$$= \frac{2C(z)}{1-z} + \frac{2z}{(1-z)^3}$$

Solve the ODE (see GF lecture).

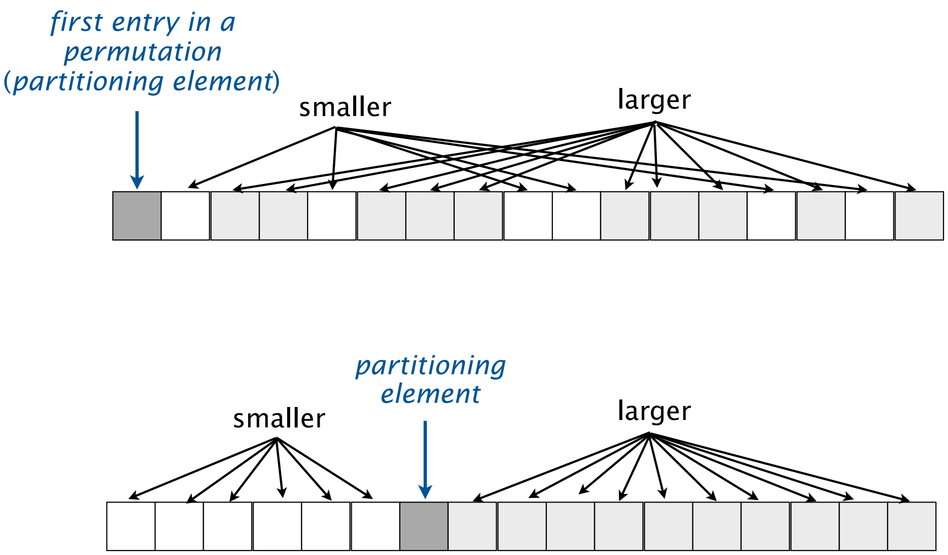
$$C(z) = \frac{2}{(1-z)^2} \ln \frac{1}{1-z} - \frac{2z}{(1-z)^2}$$

Expand.

$$C_N = 2(N+1)(H_{N+1} - 1) - 2N \sim 2N \ln N$$

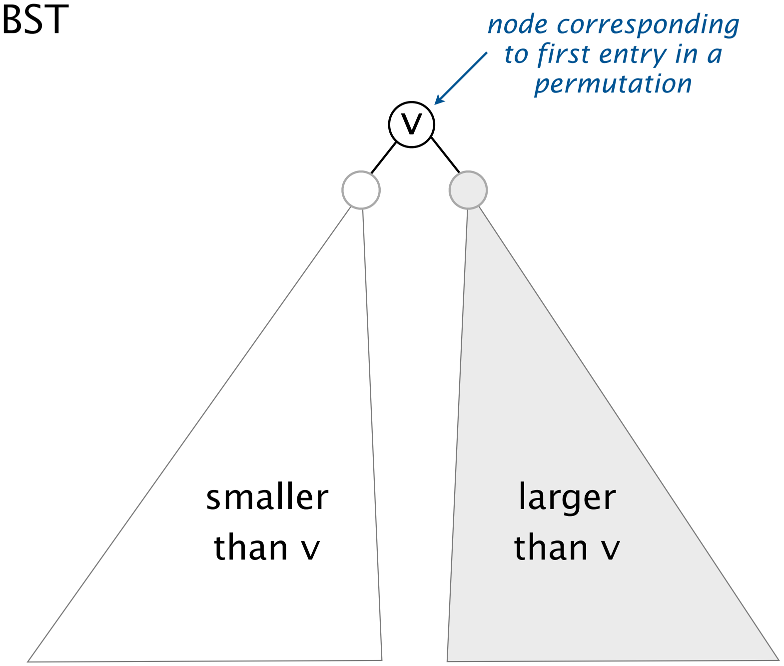
BST – quicksort bijection

Quicksort



model: random permutation
 # compares: $N+1$ + # compares for subfiles

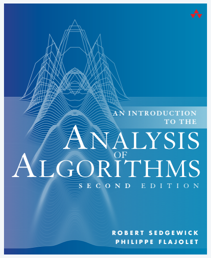
BST



model: random permutation
xpl: $N+1$ + xpl of subtrees

Average # compares for quicksort
= average external path length of BST *built from a random permutation*
= average internal path length + $2N$

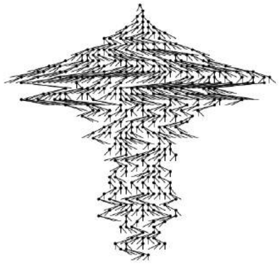
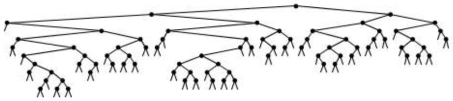
Height and other parameters



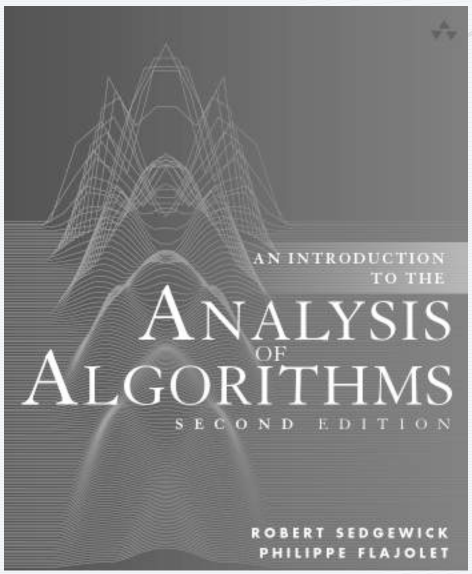
Approach works for any "additive parameter" (see text).

Height requires a different (much more intricate) approach (see text).

Summary:

	typical shape	*average path length*	*height*
random binary tree		$\sim \sqrt{\pi N}$	$\sim 2\sqrt{\pi N}$
BST built from random permutation		$\sim 2 \ln N$	$\sim c \ln N$

$c \doteq 4.311$

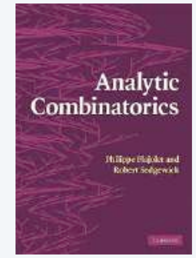


http://aofa.cs.princeton.edu

6. Trees

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

Other types of trees in combinatorics



Classic tree structures:

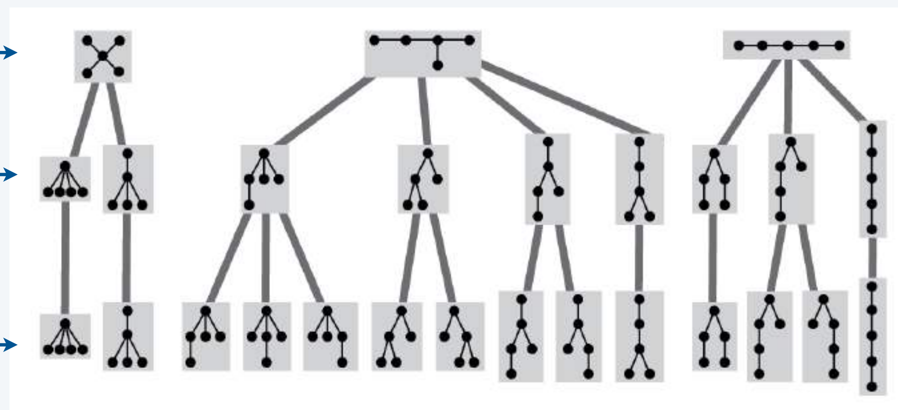
- The free tree, an acyclic connected graph.
- The rooted tree, a free tree with a distinguished root node.
- The ordered tree, a rooted tree where the order of the subtrees is significant.

Ex. 5-node trees:

3 free trees →

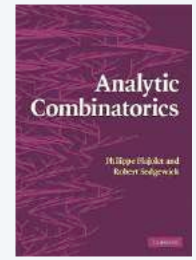
9 rooted trees →

14 ordered trees →



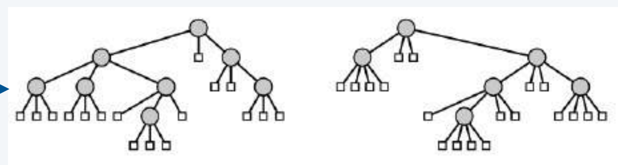
Enumeration? Path length? Stay tuned for *Analytic Combinatorics*

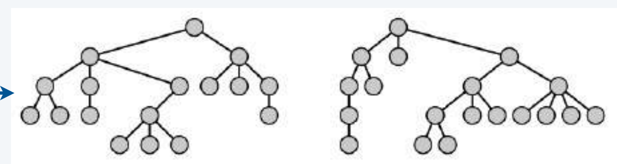
Other types of trees in algorithmics

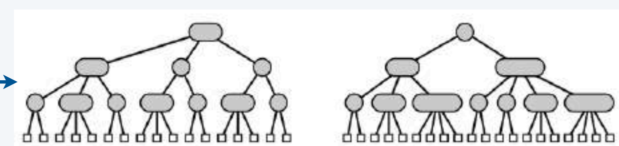


Variations on binary trees:

- The *t*-ary tree, where each node has *exactly t* children.
- The *t*-restricted tree, where each node has *at most t* children.
- The 2-3 tree, the method of choice in symbol-table implementations.

3-ary tree →

← 4-ary tree

3-restricted tree →

← 4-restricted tree

2-3 tree →

← 2-3-4 tree

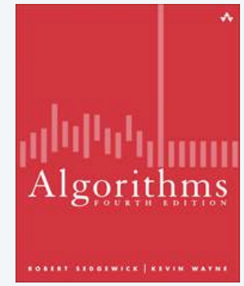
Enumeration? Path length? Stay tuned for *Analytic Combinatorics*

An unsolved problem

Balanced trees are the method of choice for symbol tables

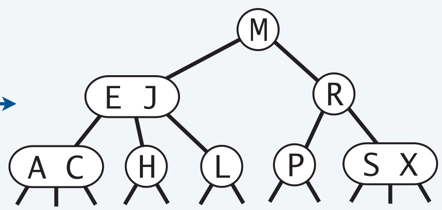
- Same search code as BSTs.
- Slight overhead for insertion.
- Guaranteed height < 2lg$N$.
- Most algorithms use 2-3 or 2-3-4 tree representations.

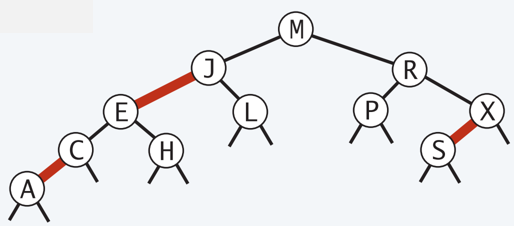
Ex. LLRB (left-leaning red-black) trees.



Section 3.3

2-3 tree



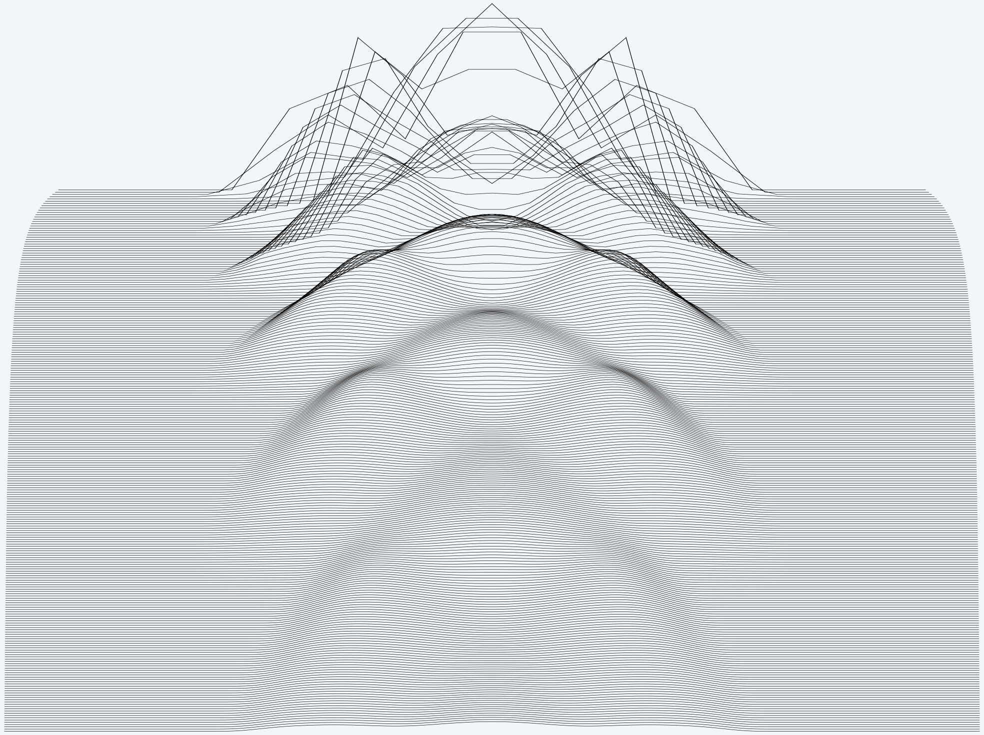


LLRB tree

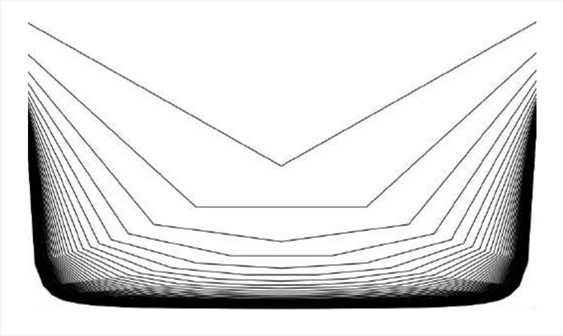
Q. Path length of balanced tree built from a random permutation? ← a property of permutations, not trees

Balanced tree distribution

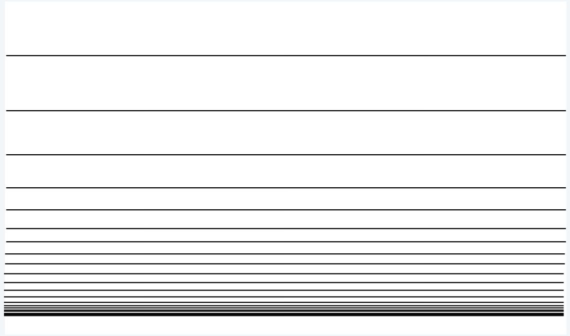
Probability that the root is of rank *k* in a randomly-chosen AVL tree.



Random binary tree

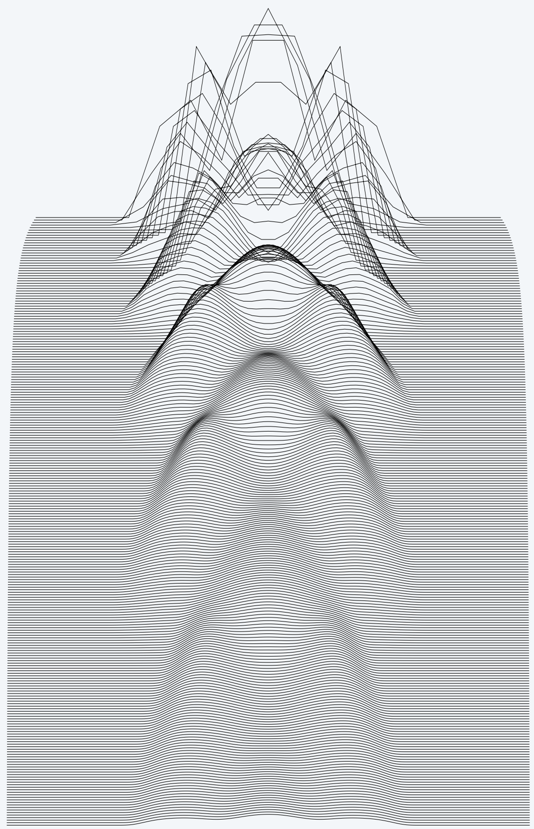


BST built from a random permutation

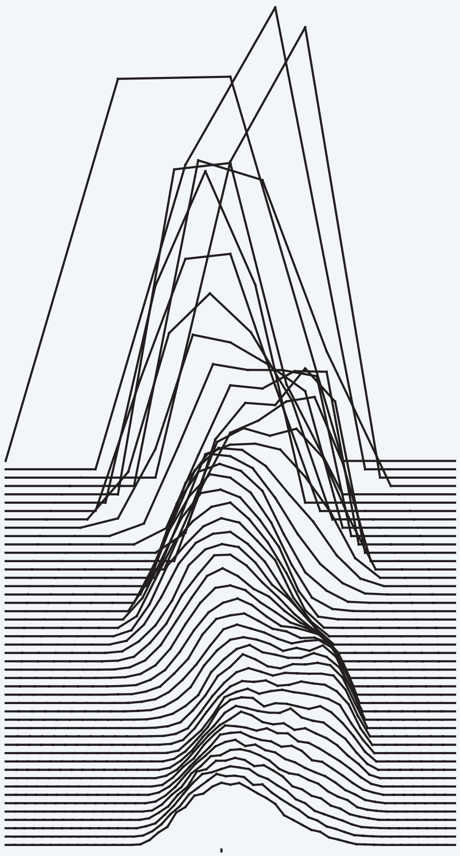


An unsolved problem

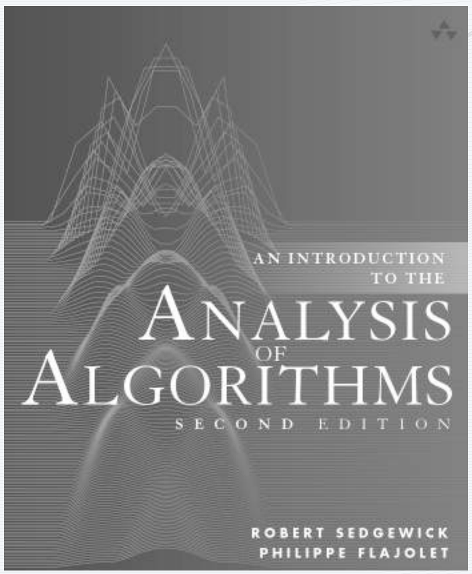
Q. Path length of balanced tree built from a random permutation?



random AVL tree



LLRB tree built from random perm (empirical)



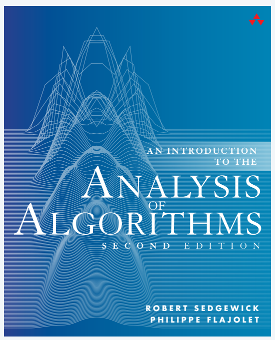
http://aofa.cs.princeton.edu

6. Trees

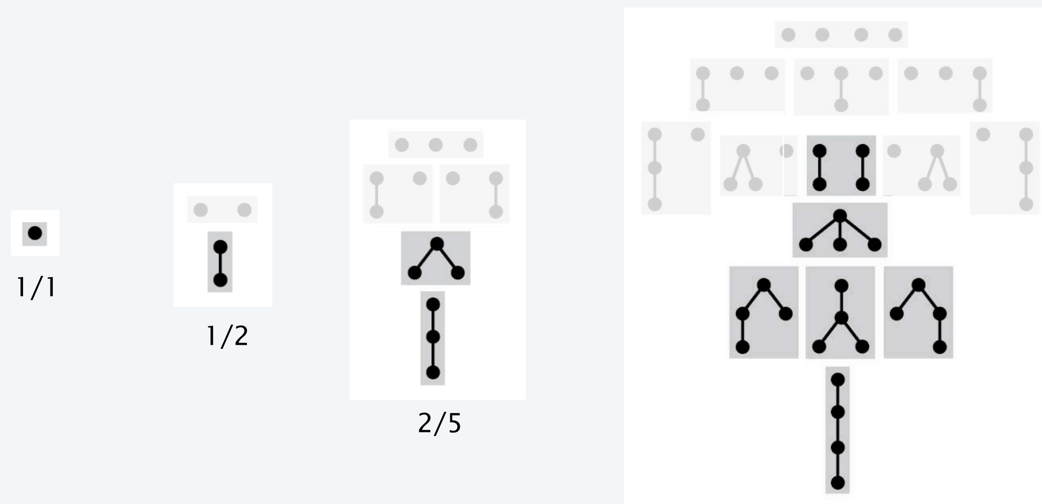
- Trees and forests
- Binary search trees
- Path length
- Other types of trees
- **Exercises**

Exercise 6.6

Tree enumeration via the symbolic method.



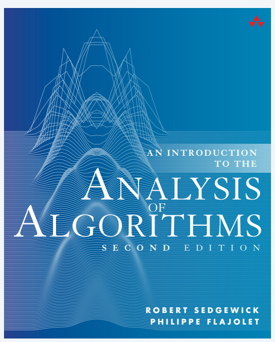
Exercise 6.6 What proportion of the forests with N nodes have no trees consisting of a single node? For $N = 1, 2, 3,$ and 4, the answer is $0, 1/2, 2/5,$ and $3/7$, respectively.



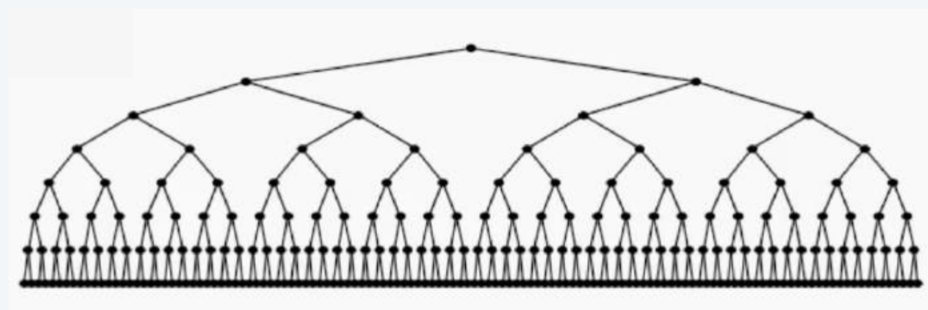
1/1 1/2 2/5 6/14

Exercise 6.27

Compute the probability that a BST is perfectly balanced.

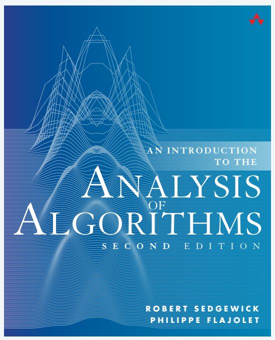


Exercise 6.27 For $N = 2^n - 1$, what is the probability that a perfectly balanced tree structure (all 2^n external nodes on level n) will be built, if all $N!$ key insertion sequences are equally likely?



Exercises 6.43

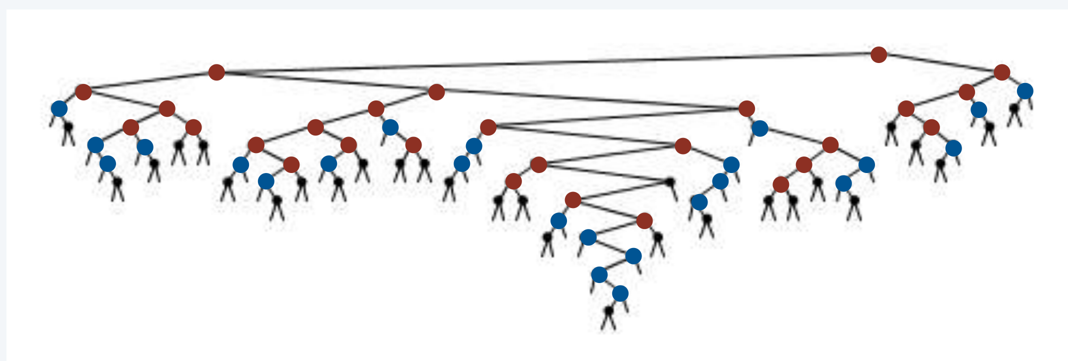
Parameters for BSTs built from a random permutation.



Answer these questions for BSTs built from a random permutation.

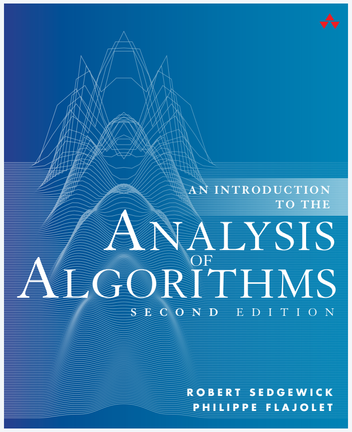
Exercise 5.15 Find the average number of internal nodes in a binary tree of size n with both children internal.

Exercise 5.16 Find the average number of internal nodes in a binary tree of size n with one child internal and one child external.



Assignments for next lecture

1. Read pages 257-344 in text.



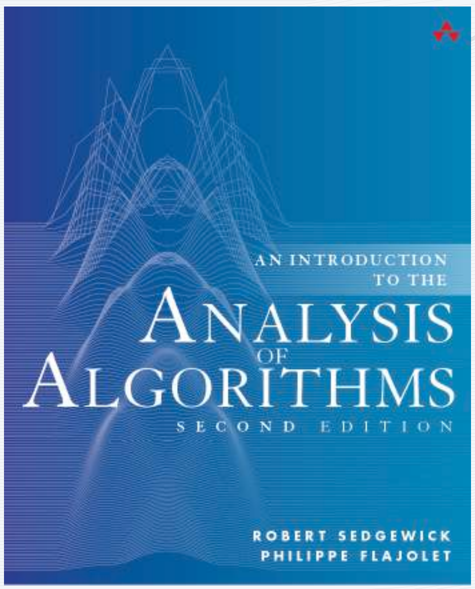
2. Run experiments to validate mathematical results.

Experiment 1. Generate 1000 random permutations for $N = 100$, 1000, and 10,000 and compare the average path length and height of the generated trees with the values predicted by analysis.

Experiment 2. *Extra credit.* Do the same for random binary trees.

3. Write up solutions to Exercises 6.6, 6.27, and 6.43.

ANALYTIC COMBINATORICS
PART ONE



http://aofa.cs.princeton.edu

6. Trees