Announcements

New Website Feature (Summary Page)

- Coursera viewing guide
- Most important points from each lecture
- Suggested problems (time permitting)

#	DATE	TOPIC	SLIDES	FLIPPED
1	2/4	Intro · Union Find	1up · 4up	
2	2/6	Analysis of Algorithms	1up · 4up	
3	2/11	Stacks and Queues	1up · 4up	
4	2/13	Elementary Sorts	1up · 4up	
5	2/18	Mergesort	1up · 4up	
6	2/20	Quicksort	1up · 4up	
7	2/25	Priority Queues	1up · 4up	1up · 4up
		Lectures and dates below are still tent		
8	2/27	Elementary Symbol Tables · BSTs	1up · 4up	
9	3/4	Balanced Search Trees	1up · 4up	
10	3/6	Hash Tables · Searching Applications	1up · 4up	

Very Short Survey after Class

- If you're surveyed out, wait until the one in two weeks (after midterm)

http://algs4.cs.princeton.edu

3.1 AND 3.2

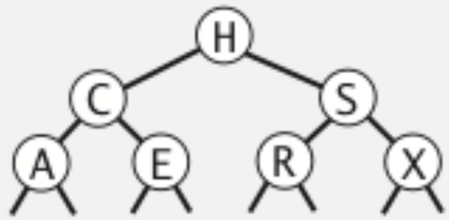
- *Basics and group work*
- *Mini lecture on symbol tables and BSTs*
- *Recursive and Iterative BST code*
- *Mini lecture on Hibbard delete*
- *Deeper thinking*

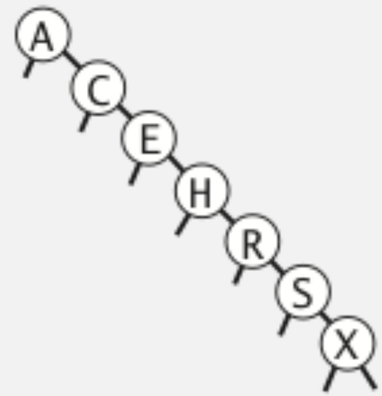
http://algs4.cs.princeton.edu

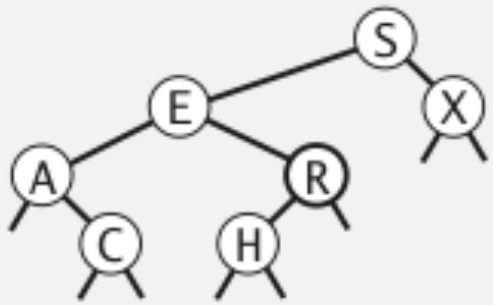
3.1 AND 3.2

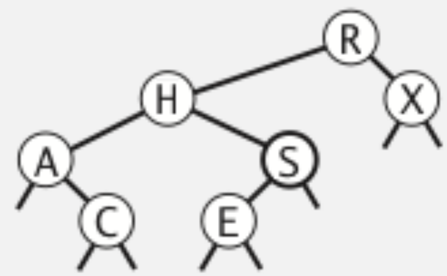
- Basics and group work
- Mini lecture on symbol tables and BSTs
- Recursive and Iterative BST code
- Mini lecture on Hibbard delete
- Deeper thinking

How many BSTs?







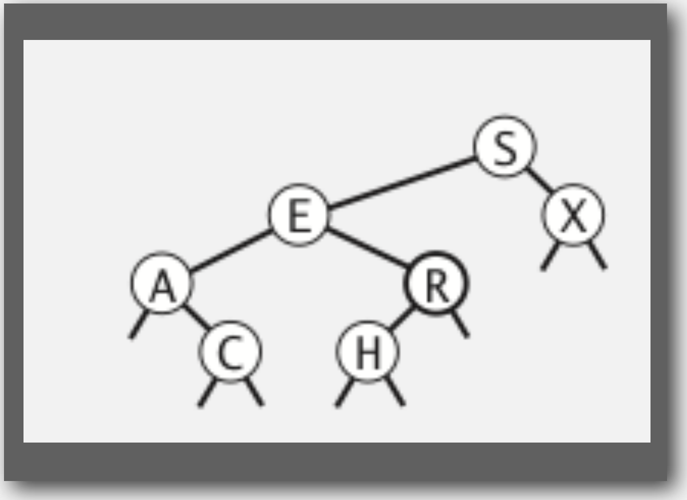


pollEv.com/jhug text to **37607**

How many of the figures above are BSTs?

A. 1 [117608]
B. 2 [118183]
C. 3 [118189]
D. 4 [118192]

Basics question #2



pollEv.com/jhug text to **37607**

Give an example of a single letter key which, when inserted, will increase the height of the tree.

```
Text:    702802 followed by a letter.
Example: 702802 Z
```

Symbol table group work

Groups of 3.

- Come up with one application of a symbol table and one application of an ordered symbol table.

application	purpose of search	key	value
dictionary	find definition	word	definition

- Given a symbol table, how do you implement: a) an array b) a set c) a symbol table that allows look-up by key **OR** value?
- Other than being an equivalence relation, what are the rules for implementing an `equals()` method? When should you use `equals()` to implement a symbol table? When should you use `compareTo()`?

pg 102

- What is the advantage of disallowing null keys?
- What is the advantage of disallowing null values?
- **Bonus**: How would you design an efficient data structure that keeps getting new words and can be asked for the M most popular words at any point? Can you find a solution that uses memory proportional to M?

Context

application	purpose of search	key	value
dictionary	find definition	word	definition
encyclopedia	find info		
username/pw	log in	username	password
histogram	count occurrrences	item	count
spell checker	find spell errors	misspelled words	correctly spelled words

Notes

- Ordered symbol table: can get minimum or maximum elements
- Ceiling: [first item bigger than or equal to x]
- Select: Get kth element (e.g. 3rd place)
- Rank: What place is element in (Green Bay Packers are ...computing... #3)

Symbol table basics

Given a symbol table, how do you implement: a) an array b) a set c) a symbol table that goes allows look-up by key or value?

- Array:
  - key: successive integers as keys
  - value: thing you're putting in array
  - put(0, 'first thing'), put (1, 'second thing')
- Set: (able to see if a thing is there or not, only allow one copy)
  - Keys with no values it seems...
  - Key: thing you're storing
  - Value: [dummy value]]
- Both way lookup symbol table
  - Have one symbol table for key -> value
  - Another for value -> key
  - Every time you   s1.put(key, value)   you also s2.put(value, key)
  - **One issue:** Would have to ensure that each value is mapped to by only one key

Symbol table basics

Other than being an equivalence relation, what are the rules for implementing an `equals()` method? When should you use `equals()` to implement a symbol table? When should you use `compareTo()`?

- Check that the two things being compared are not the same exact reference [just saves time]
- Check that the two objects are the same class
- Cast from Object ot the appropriate class
- Return false if the instance variables do not match

When do use each?

- ~~equals(): primitive types~~
- ~~compareTo(): object --- but there's no natural ordering of say... pictures~~

- **equals: there is not a natural order**
- **compareTo: if there's a natural order**

Symbol table basics

What is the advantage of disallowing null keys?

- key.compareTo(x.key): //allows us to do this lazily

What is the advantage of disallowing null values?

- ~~Waste of space~~
- ~~Iterate through symbol table, would have nulls~~: Not inherently a problem
- Client safety (in 226 return 'null' as the value to indicate a thing is missing from the table, we'll see this soon)
- If we allowed null, we'd need to use exceptions to handle missing things

Bonus: How would you design an efficient data structure that keeps getting new words and can be asked for the M most popular words at any point?

- Approach: N space: Ordered symbol table, key: word, value: count. Use select to get top M words. When new word arrives, get the value, and increase by one (if not there, you'd make a new entry with value 1).
- Can you do it in M space? PQ ?? No! Have to track old word counts.

http://algs4.cs.princeton.edu

3.1 AND 3.2

- Basics and group work
- *Mini lecture on symbol tables and BSTs*
- Recursive and Iterative BST code
- Mini lecture on Hibbard delete
- Deeper thinking

Associative arrays are pretty darn fundamental

Almost as much as arrays

Key part of most programming languages.

Known by many names

- Dictionary, map, associative array, symbol table

Java: Interface Map

Implementations

- TreeMap
- ConcurrentSkipListMap
- EnumMap
- HashMap
- HashTable
- ConcurrentHashMap
- LinkedHashMap
- WeakHashMap

Python

(and Perl, and many other languages)

Totally baked into the language.

```
aa = {‘foo’: ‘bar’, 1: 2}

print aa[1]
```

Crazy stuff

There's a language that doesn't even have arrays, only associative arrays

PHP

There's a language where objects are merely associative arrays

obj.var gets translated to obj[var]

Javascript

Associative arrays and functions

Close connection: f(x, y, z) vs. aa.get([x, y, z])

Associative arrays (lookup tables) are one way to implement functions.

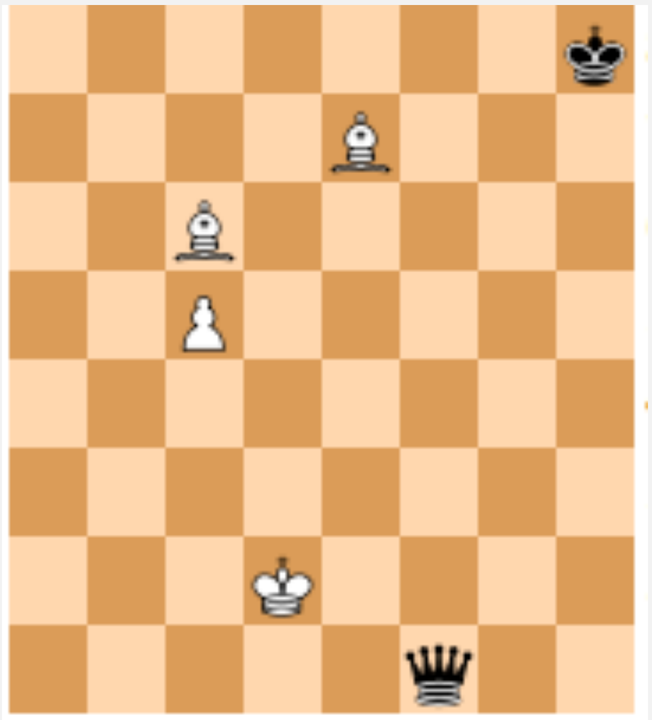
Simple lookup table (regular array): log tables

Oldest lookup table? [5th century AD] Sine function

मखि भखि फखि धखि णखि ञखि ङखि हस्झ स्ककि किष्ग श्घकि किघव ।
घलकि किग्र हक्य धकि किच स्ग झश ङ्व क्ल प्त फ छ कला-अर्ध-ज्यास् ॥

Associative arrays and chess

Minor revolution in chess playing

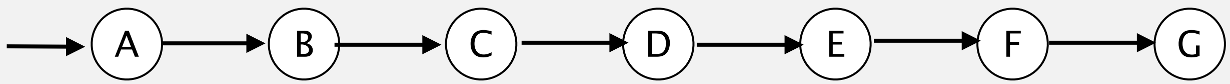


Black wins in 154 moves

Implementation of a symbol table (a.k.a. associative array)

Ordered Linked List

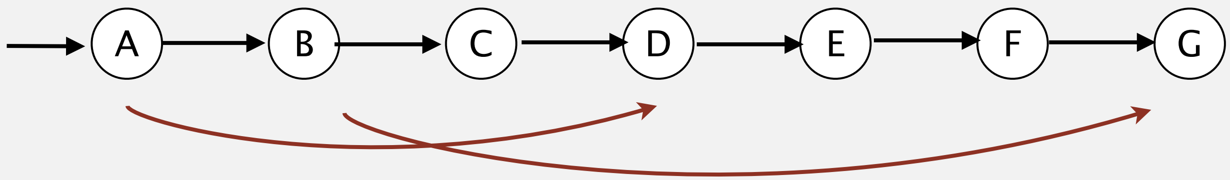
- Slow to find items we want (even though we're in order)



Implementation of a symbol table (a.k.a. associative array)

Ordered Linked List

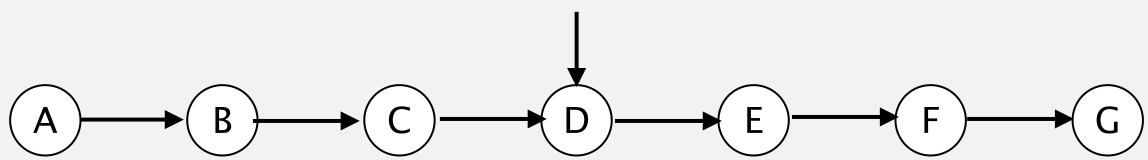
- Slow to find items we want (even though we're in order)
- Adding (random) express lanes: Skip list (won't discuss in 226)



Implementation of a symbol table (a.k.a. associative array)

Ordered Linked List to BST

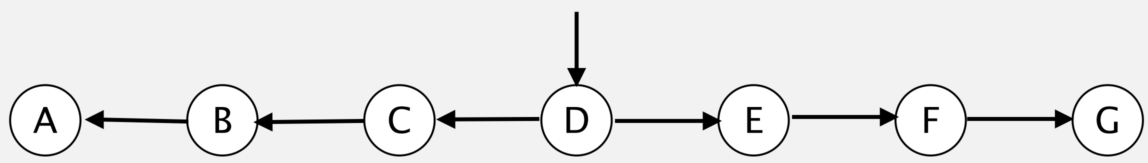
- Slow to find items we want (even though we're in order)
- Move pointer to middle: Can't see earlier elements



Implementation of a symbol table (a.k.a. associative array)

Ordered Linked List to BST

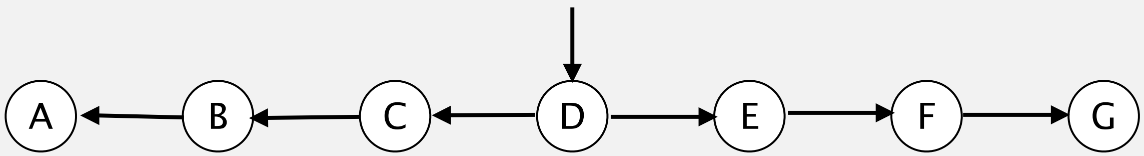
- Slow to find items we want (even though we're in order).
- Pointer in middle, flip left links: Search time is halved.



Implementation of a symbol table (a.k.a. associative array)

Ordered Linked List to BST

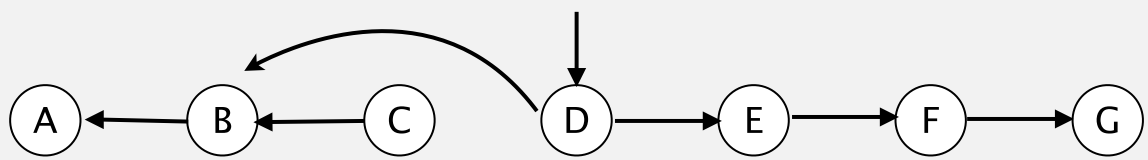
- Slow to find items we want (even though we're in order).
- Pointer in middle, flip left links: Search time is halved.
- Can do better: Dream big!



Implementation of a symbol table (a.k.a. associative array)

Ordered Linked List to BST

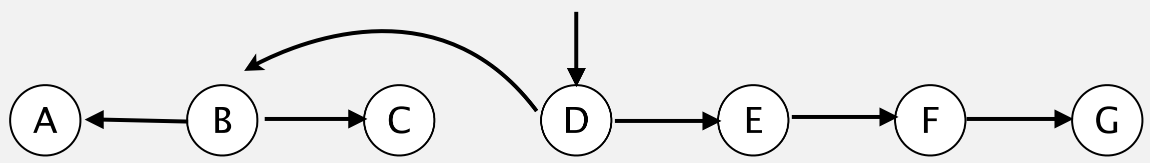
- Slow to find items we want (even though we're in order).
- Pointer in middle, flip left links: Search time is halved.
- Allow every node to make big jumps.



Implementation of a symbol table (a.k.a. associative array)

Ordered Linked List to BST

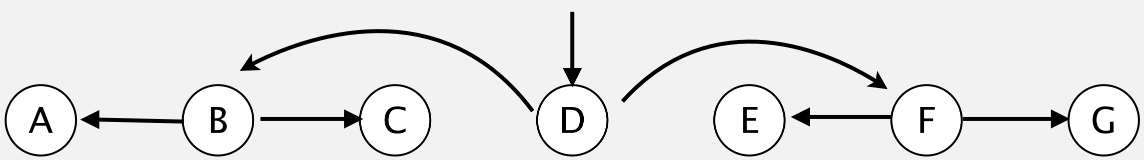
- Slow to find items we want (even though we're in order).
- Pointer in middle, flip left links: Search time is halved.
- Allow every node to make big jumps.

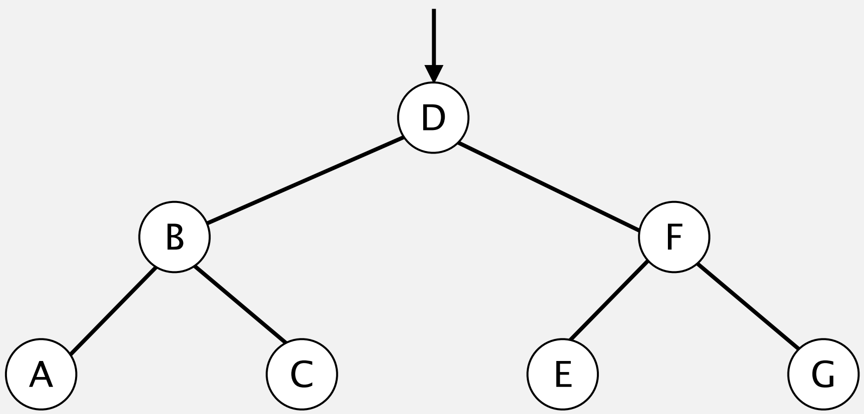


Implementation of a symbol table (a.k.a. associative array)

Ordered Linked List to BST

- Slow to find items we want (even though we're in order).
- Pointer in middle, flip left links: Search time is halved.
- Allow every node to make big jumps.

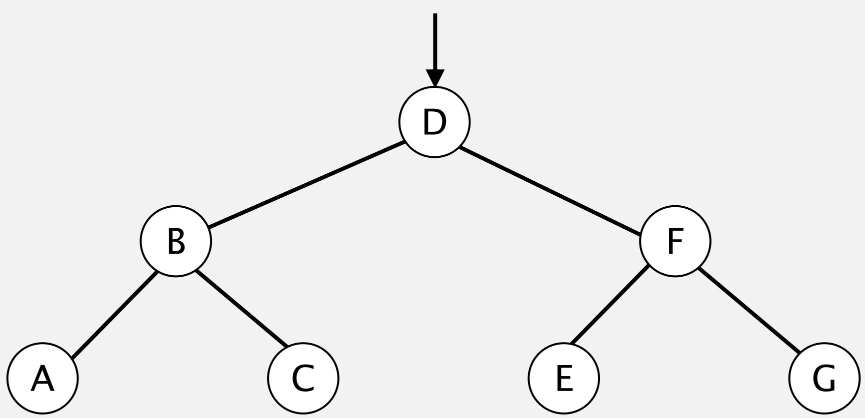




Implementation of a symbol table (a.k.a. associative array)

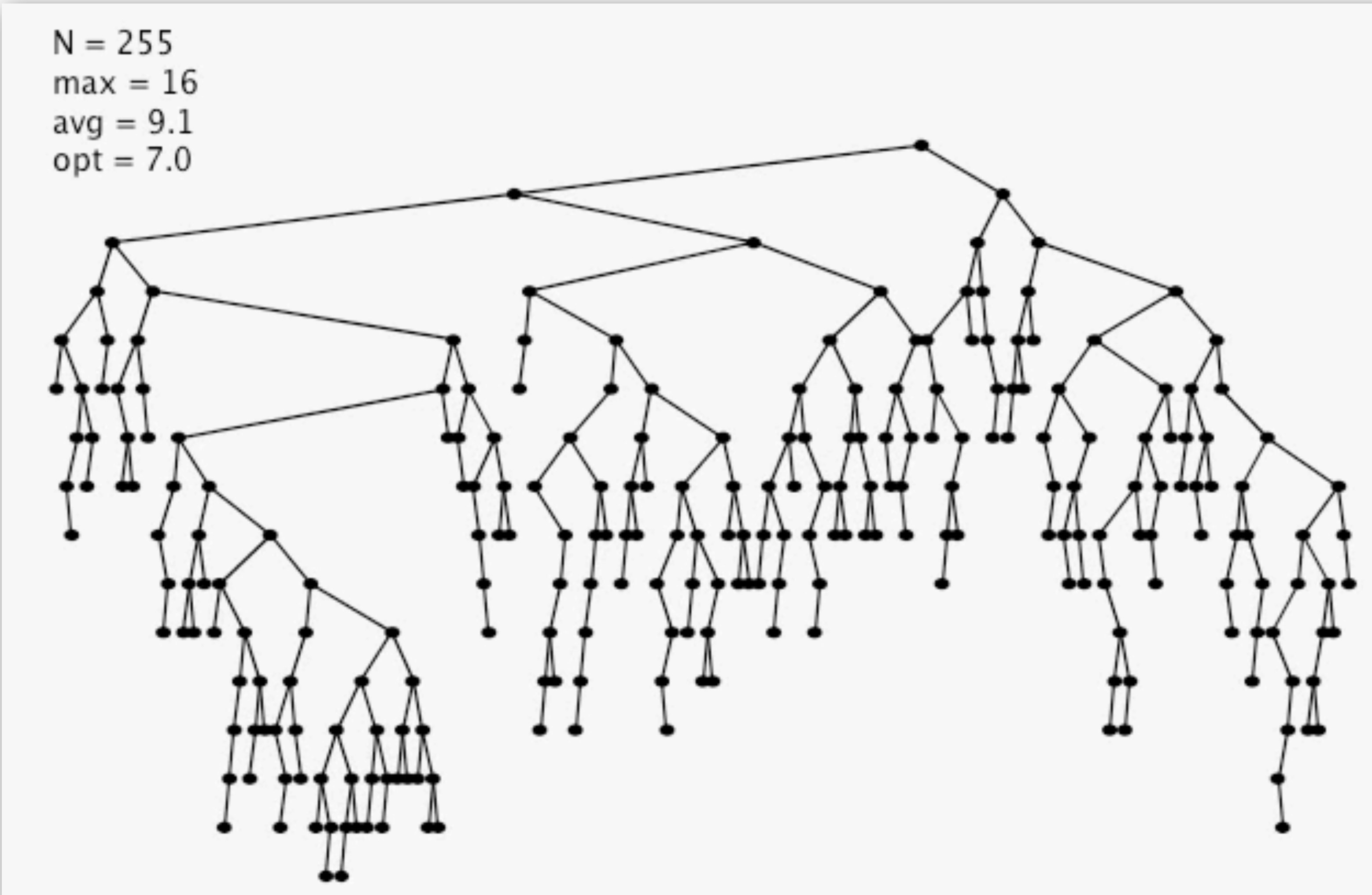
Binary Search Tree of height h

- Insert and search now take `h+1` calls to `compare()` in the worst case.
  - lg N if tree is balanced!
- Practical trees will be built from disordered data.
  - Goal is to maintain reasonable balance after `put()` and `delete()`.



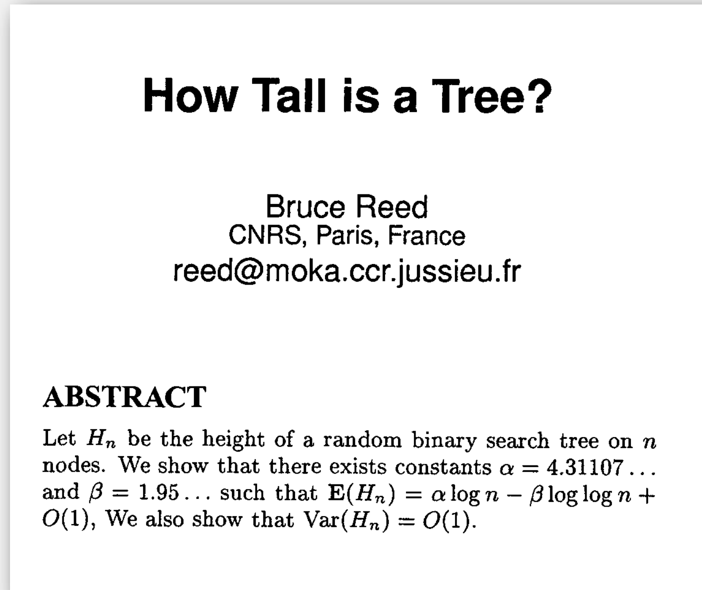
BST insertion: random order visualization

Ex. Insert keys in random order.



BSTs: mathematical analysis

Proposition. [Reed, 2003] If N distinct keys are inserted in random order, expected height of tree is $\sim 4.311 \ln N$.



How Tall is a Tree?

Bruce Reed
CNRS, Paris, France
reed@moka.ccr.jussieu.fr

ABSTRACT

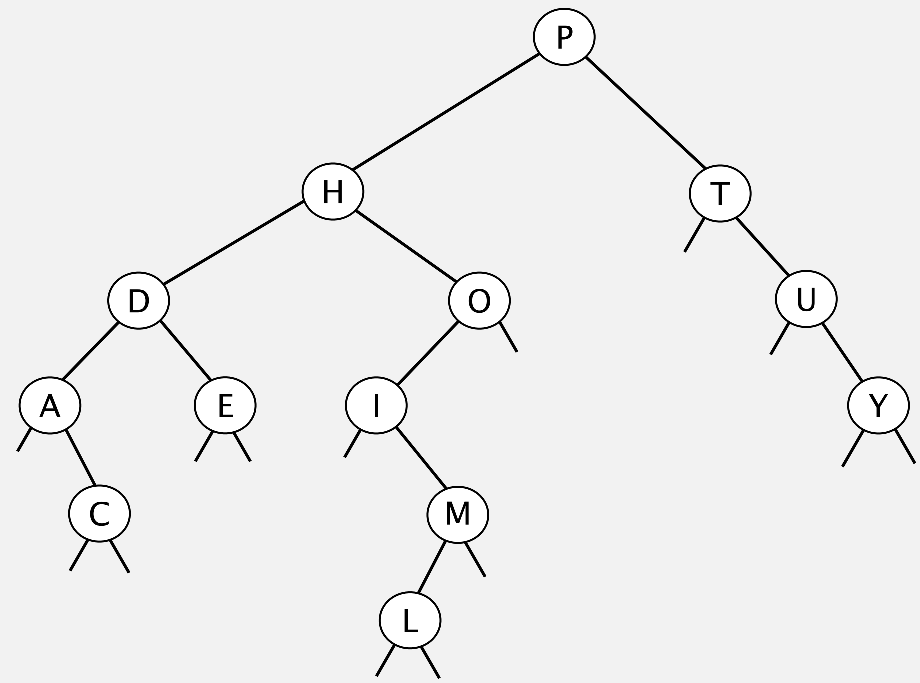
Let H_n be the height of a random binary search tree on n nodes. We show that there exists constants $\alpha = 4.31107\ldots$ and $\beta = 1.95\ldots$ such that $\mathbf{E}(H_n) = \alpha \log n - \beta \log\log n + O(1)$, We also show that $\mathrm{Var}(H_n) = O(1)$.

But… Worst-case height is N.
(exponentially small chance when keys are inserted in random order)

Proposition. If N distinct keys are inserted into a BST in random order, the expected number of compares for a search/insert is $\sim 2 \ln N$.
Pf. 1-1 correspondence with quicksort partitioning.

Correspondence between BSTs and quicksort partitioning

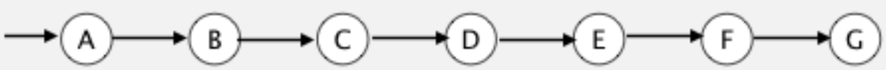
0	1	2	3	4	5	6	7	8	9	10	11	12	13
P	S	E	U	D	O	M	Y	T	H	I	C	A	L
P	S	E	U	D	O	M	Y	T	H	I	C	A	L
H	L	E	A	D	O	M	C	I	P	T	Y	U	S
D	C	E	A	H	O	M	L	I	P	T	Y	U	S
A	C	D	E	H	O	M	L	I	P	T	Y	U	S
A	C	D	E	H	O	M	L	I	P	T	Y	U	S
A	C	D	E	H	O	M	L	I	P	T	Y	U	S
A	C	D	E	H	O	M	L	I	P	T	Y	U	S
A	C	D	E	H	I	M	L	O	P	T	Y	U	S
A	C	D	E	H	I	M	L	O	P	T	Y	U	S
A	C	D	E	H	I	L	M	O	P	T	Y	U	S
A	C	D	E	H	I	L	M	O	P	T	Y	U	S
A	C	D	E	H	I	L	M	O	P	S	T	U	Y
A	C	D	E	H	I	L	M	O	P	S	T	U	Y
A	C	D	E	H	I	L	M	O	P	S	T	U	Y
A	C	D	E	H	I	L	M	O	P	S	T	U	Y
A	C	D	E	H	I	L	M	O	P	S	T	U	Y

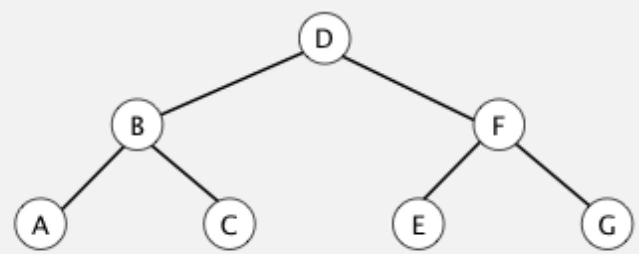


Remark. Correspondence is 1-1 if array has no duplicate keys.

Symbol table implementations

implementation	guarantee		average case		ordered ops?	operations on keys
	search	insert	search hit	insert		
sequential search (unordered list)	N	N	N/2	N	no	`equals()`
binary search (ordered array)	lg N	N	lg N	N/2	yes	`compareTo()`
BST	N	N	1.39 lg N	1.39 lg N	yes	`compareTo()`





3.1 AND 3.2

- Basics and group work
- Mini lecture on symbol tables and BSTs
- Recursive and Iterative BST code
- Mini lecture on Hibbard delete
- Deeper thinking

Algorithms

Robert Sedgewick | Kevin Wayne

http://algs4.cs.princeton.edu

Recursive BST code

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

   public Value get(Key key)
   public void put(Key key, Value val)
   public void deleteMin(Node x)
   public void delete(Node x)
}
```

```
private class Node {
   Key key;
   Value value;
   Node left, right;
}
```

Suggested Exercise

- Complete an implementation of the API above.
- We'll do `get()` and `put()` in lecture.
- Try the delete methods on your own.
- On assignment 5 (kdtree) you'll get plenty of practice with recursive search tree search and construction.

Recursive BST code

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

   public Value get(Key key)
   public void put(Key key, Value val)
}
```

```
private class Node {
   Key key;
   Value value;
   Node left, right;
}
```

```
 public Value get(Key key) {
   return get(root, key);
 }

 private Value get(Node x, key key) {
    if (x == null)  return null;

    int cmp = key.compareTo(x.key);
    if (cmp == 0)   return x.value;
    if (cmp < 0)    return get(x.left, key);
    if (cmp > 0)    return get(x.right, key);

 }
```

Recursive BST code

```
public void put(Key key, Value val) {

}
```

Recursive BST code

```
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.value = val;
   else if (cmp < 0)   x.left = put(x.left, key, val);
   else if (cmp > 0)   x.right = put(x.right, key, val);
   return x;
}
```

Recursive BST code

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

   public Value get(Key key)
   public void put(Key key, Value val)
   public void deleteMin(Node x)
   public void delete(Node x)
}
```

```
private class Node {
   Key key;
   Value value;
   Node left, right;
}
```

Deletion

- Conceptually not so bad (will discuss in a few slides).
- Clean implementation is rather tricky.

Recursive BST code (group problem)

```
public Key mystery(Key key) {
   Node best = mystery(root, key, null);
   if (best == null) return null;
   return best.key;
}

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

pollEv.com/jhug   text to **37607**

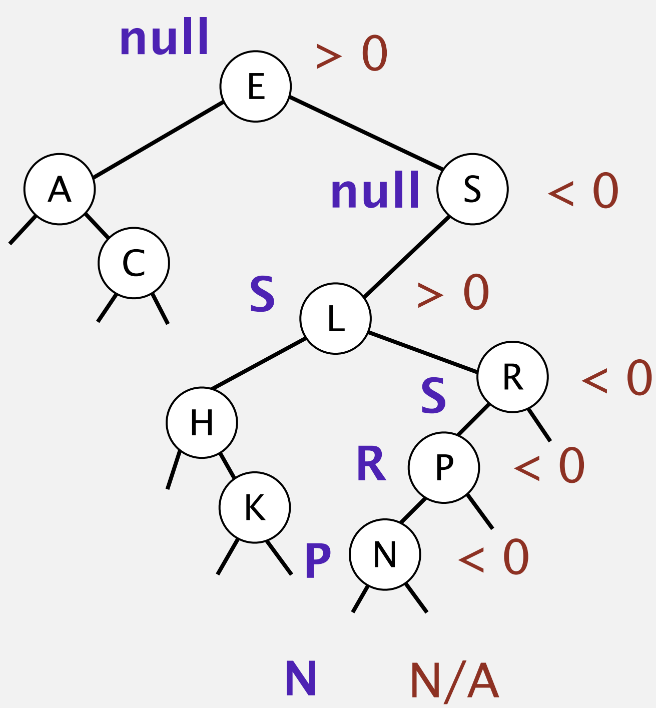
What operation does the code perform?

A. Floor(key)	[703632]	D. Maximum key	[703635]
B. Ceiling(key)	[703633]	E. Key passed as argument	[703636]
C. Minimum key	[703634]	F. Median key	[703637]

Note: `y = floor(x)` is smallest x in tree s.t. `y <= x`

mystery(root, M, null)

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

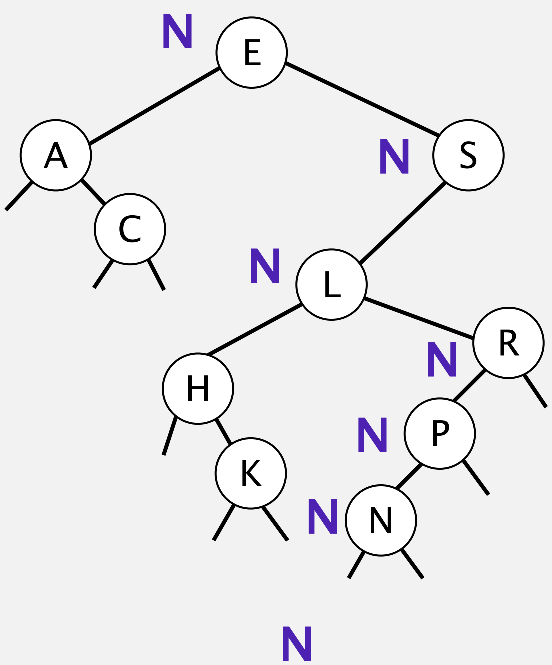


What operation does the code perform?

Ceiling

mystery(root, M, null)

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



What operation does the code perform?

Ceiling

Iterative code for BSTs

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

   public void put(Key key, Value val)
}
```

```
private class Node {
   Key key;
   Value value;
   Node left, right;
}
```

```
public void put(Key key, Value val) {
   Start with x = root;
   repeat until found or inserted:
      compare key to x.key:
         if less and nothing to the left, insert;
         if less and something to the left, x = x.left;
         if more and nothing to the right, insert;
         if more and something to the right, x = x.right;
         if equal, replace value;
}
```

See online slides for iterative source code example.

http://algs4.cs.princeton.edu

3.1 and 3.2

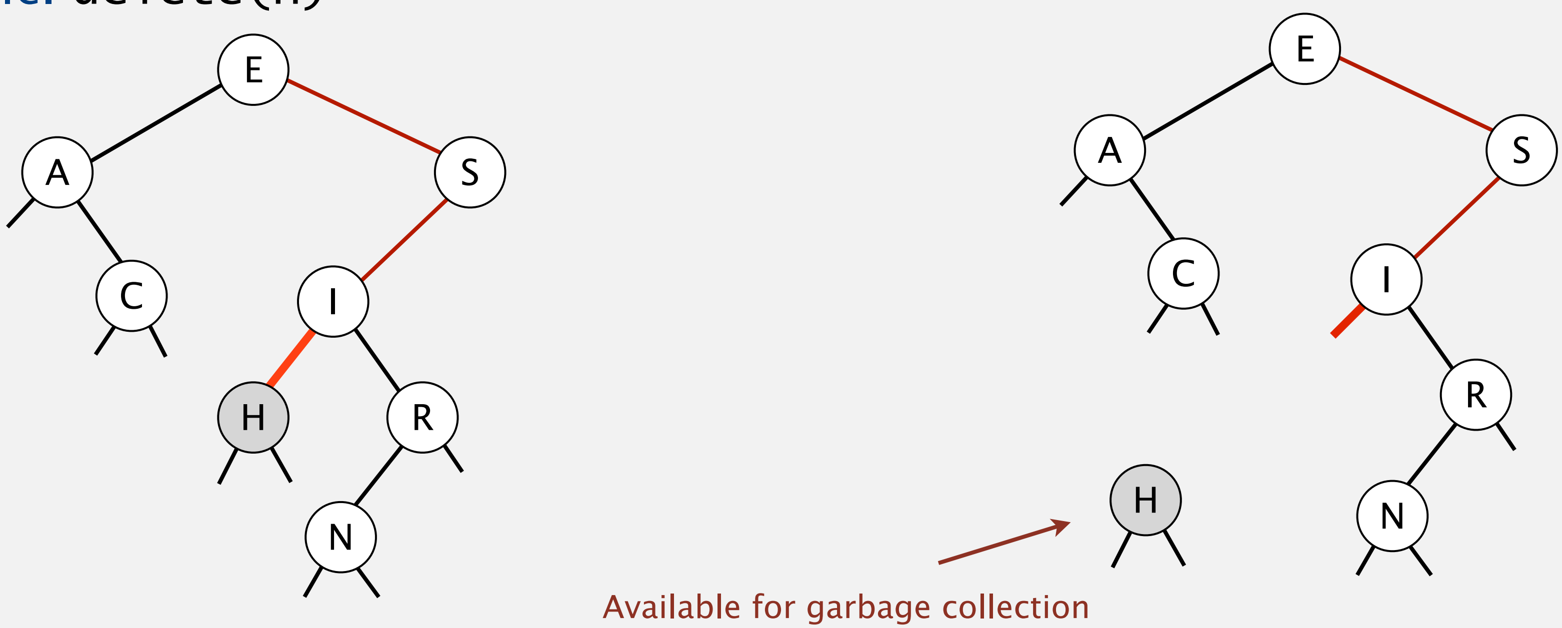
- *Basics and group work*
- *Mini lecture on symbol tables and BSTs*
- *Recursive and Iterative BST code*
- *Mini lecture on Hibbard delete*
- *Deeper thinking*

Hibbard deletion

To delete a node with key `k`: search for node `t` containing key `k`.

Case 0. [0 children] Delete `t` by setting parent link to null.

Example. `delete(H)`



Recursive Call. Much like `put()`, visited nodes return a new pointer to their parent. Example: When `x = I: x.left = delete(x.left(), H);`
When `x = H: return null;`

Hibbard deletion

To delete a node with key `k`: search for node `t` containing key `k`.

Case 1. [1 child] Delete `t` by replacing parent link.

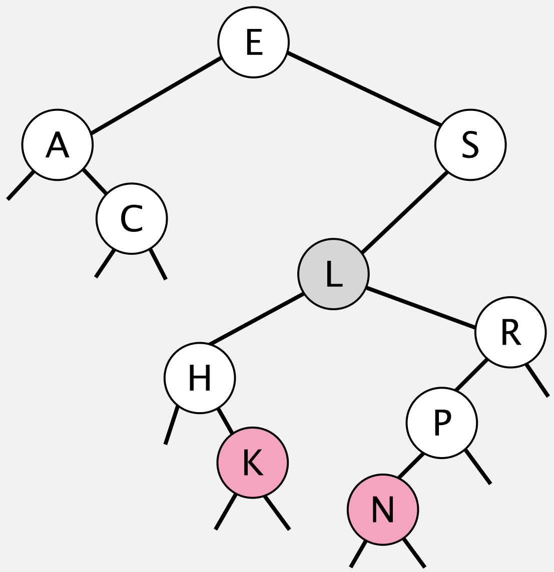
Example. `delete(R)`

Hibbard deletion

To delete a node with key `k`: search for node `t` containing key `k`.

Case 2. [2 children] Delete `t` by replacing parent link.

Example. `delete(L)`



Choosing a replacement.

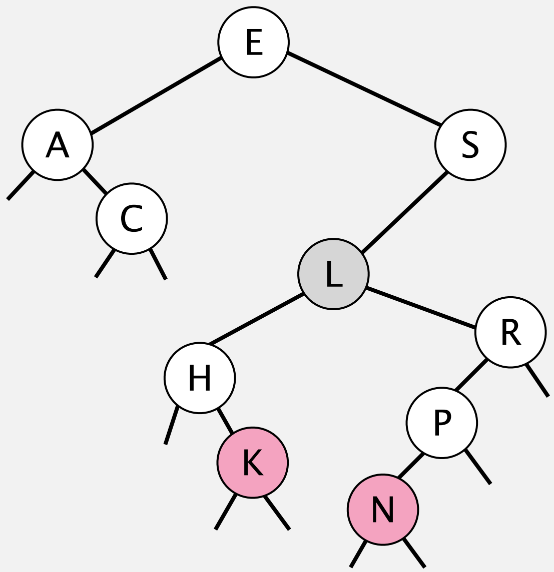
- Successor: N
- Predecessor: K

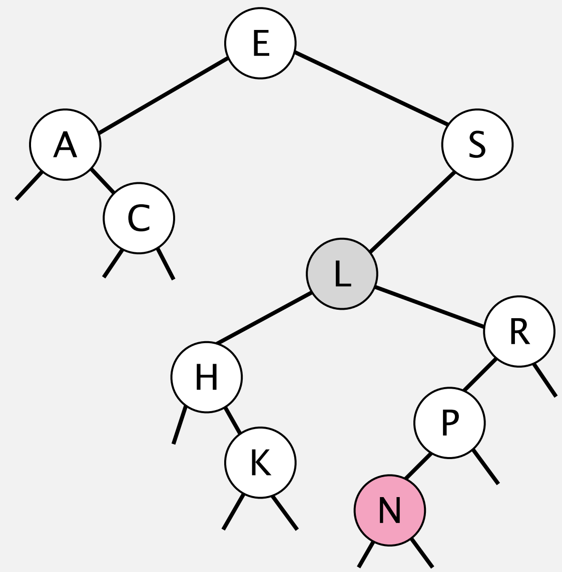
Hibbard deletion

To delete a node with key `k`: search for node `t` containing key `k`.

Case 2. [2 children] Delete `t` by replacing parent link.

Example. `delete(L)`





Choosing a replacement.

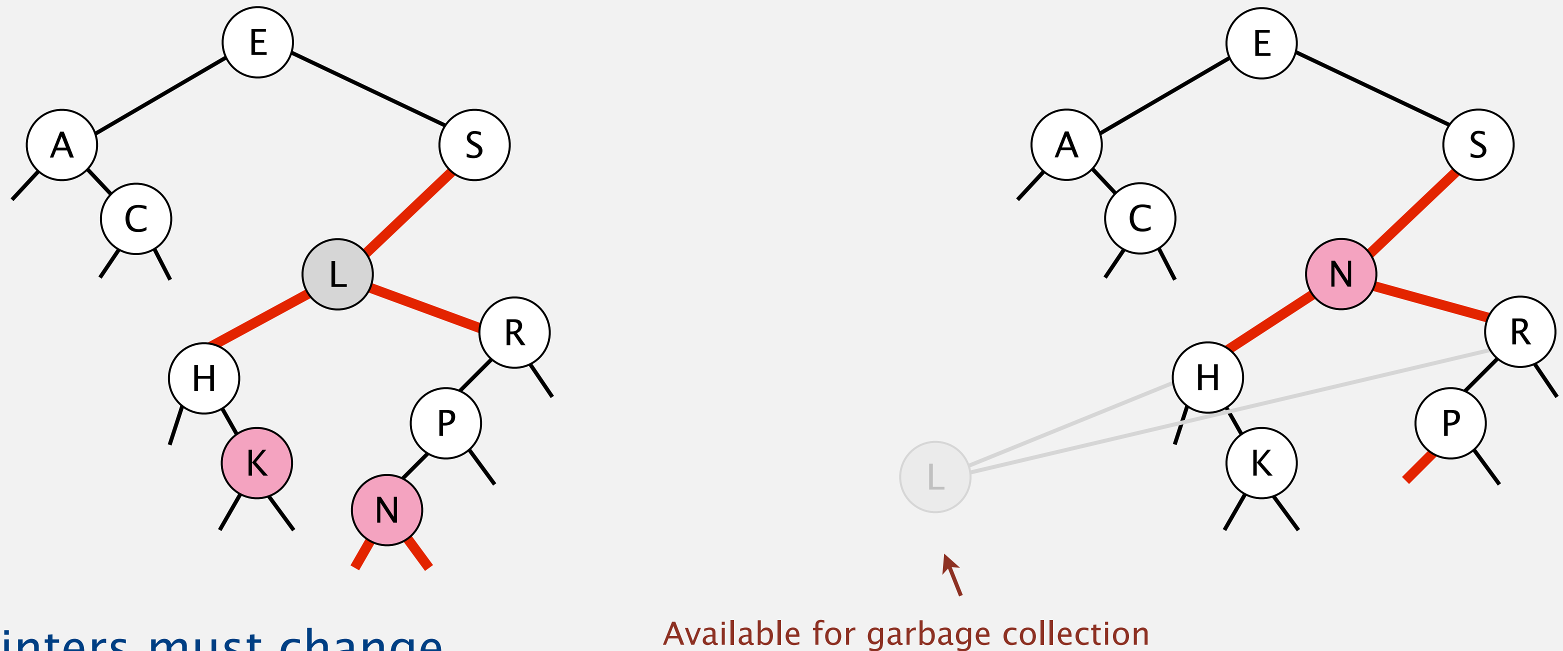
- Successor: N [by convention]
- ~~Predecessor: K~~

Hibbard deletion

To delete a node with key `k`: search for node `t` containing key `k`.

Case 2. [2 children] Delete `t` by replacing parent link.

Example. `delete(L)`

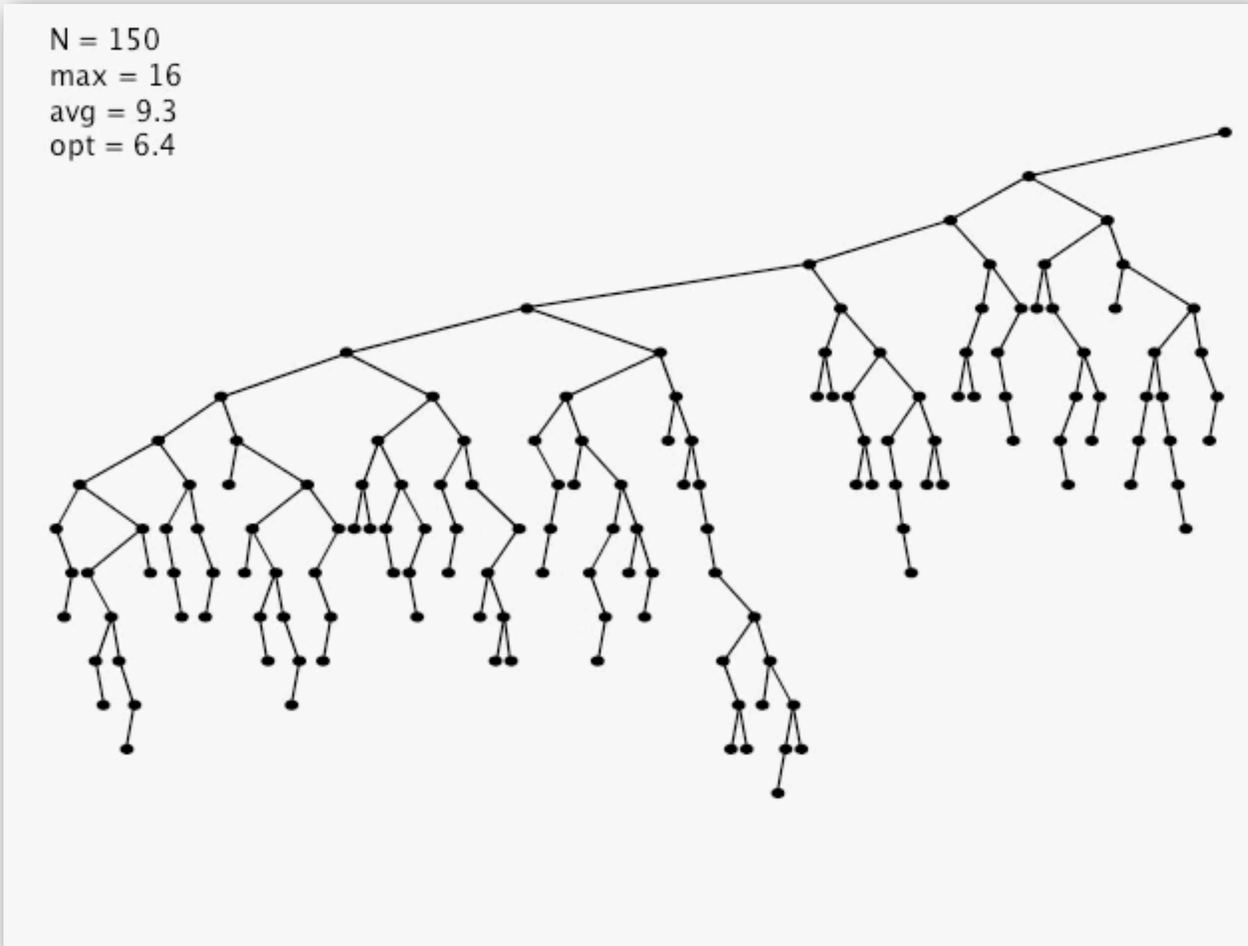


Four pointers must change.

- Parent of deleted node
- Parent of successor
- Left child of successor
- Right child of successor

Hibbard deletion: analysis

Unsatisfactory solution. Not symmetric.



Surprising consequence. Trees not random (!) $\Rightarrow$ $\text{sqrt}\,(N)$ per op.

Longstanding open problem. Simple and efficient delete for BSTs.

ST implementations: summary

implementation	guarantee			average case			ordered iteration?	operations on keys
	search	insert	delete	search hit	insert	delete		
sequential search (linked list)	N	N	N	N/2	N	N/2	no	`equals()`
binary search (ordered array)	lg N	N	N	lg N	N/2	N/2	yes	`compareTo()`
BST	N	N	N	1.39 lg N	1.39 lg N	√N	yes	`compareTo()`

other operations also become √N
if deletions allowed

Next lecture. Guarantee logarithmic performance for all operations.

http://algs4.cs.princeton.edu

3.1 and 3.2

- Basics and group work
- Mini lecture on symbol tables and BSTs
- Recursive and Iterative BST code
- Mini lecture on Hibbard delete
- Deeper thinking

Design Problem

Solo

- **Erweiterten Netzwerk** is a new German minimalist social networking site that provides only two operations for its logged-in users.
  - Neu : Enter another user's username and click the **Neu** button. This marks the two users as friends.
  - Erweiterten Netzwerk : Type in another user's username and determine whether the two users are in the same extended network (i.e. there exists some chain of friends between the two users).

pollEv.com/jhug text to **37607**

Identify at least one API that **Erweiterten Netzwerk** should use:

A. Queue	[77170]	D. Priority Queue	[78510]
B. Union-find	[77173]	E. Symbol Table	[78580]
C. Stack	[77654]	F. Randomized Queue	[78635]

Note: There may be more than one 'good' answer.

Design Problem

Groups of 3 (design problem)

- **Erweiterten Netzwerk** is a new German minimalist social networking site that provides only two operations for its logged-in users.
  - Neu : Enter another user's username and click the **Neu** button. This marks the two users as friends.
  - Erweiterten Netzwerk : Type in another user's username and determine whether the two users are in the same extended network (i.e. there exists some chain of friends between the two users).

```
B. Union-find  [77173]    E. Symbol Table          [78580]
```

- In a group: What is the worst case order of growth of the running time that **Erweiterten Netzwerk** can guarantee for M operations and N users?

Amortized Time

Groups of 3

- Your symbol table implementation supports the *insert* and *search* operations in *amortized* 4 lg N compares. Which of the following are true?
  - I. Starting from an empty data structure, and sequence of *N insert* and *search* operations uses at most *4 N lg N* compares.
  - II. Any sequence of N *insert* and *search* operations uses at most *4 N lg N* compares.
  - III. Starting from an empty data structure, the expected number of compares for *N insert* and *search* operations is *4 N lg N*, but there is a (small) probability that it will take more.

pollEv.com/jhug text to **37607**

Which of the following are true?

```
A. I only             [704403]     D. I, II, and III   [704406]
B. I and II only      [704404]     E. None             [704407]
C. I and III only     [704405]
```