



CS 311O

Balanced Trees

Prof. Clarkson
Fall 2019

Today's music: Get the Balance Right by Depeche Mode

CLICKER QUESTION 1

Review

Previously in 3110:

- Efficiency
- Hash tables: imperative constant-time maps

Today:

- Balanced trees: functional maps

Running example: Sets

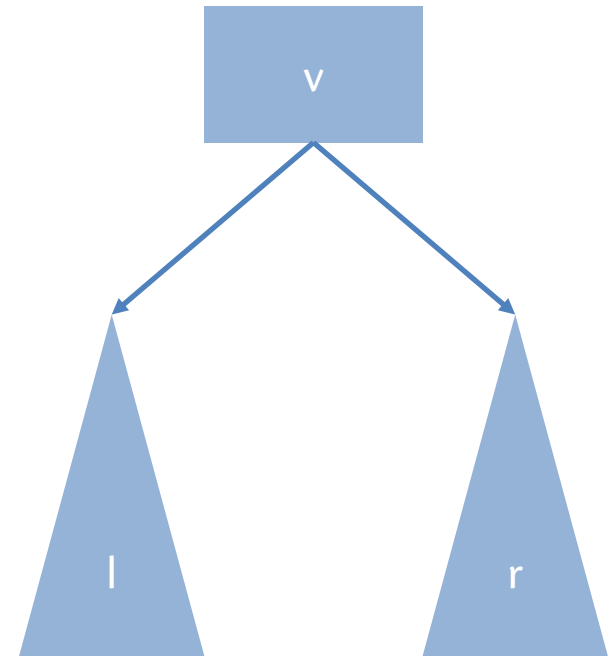
```
module type Set = sig
  type 'a t
  val empty : 'a t
  val insert : 'a -> 'a t -> 'a t
  val mem : 'a -> 'a t -> bool
  ...
end
```

LIST SET

BST SET

Binary search tree (BST)

- Binary tree: every node has two subtrees
- BST invariant:
 - all values in l are less than v
 - all values in r are greater than v



CLICKER QUESTION 2

Set implementations: performance

	Workload 1	
	insert	mem
ListSet	35s	106s

MacBook, 1.3 GHz Intel Core m7, 8 GB RAM, median of three runs

Set implementations: performance

	Workload 1	
	insert	mem
ListSet	35s	106s
BstSet	130s	149s

MacBook, 1.3 GHz Intel Core m7, 8 GB RAM, median of three runs

Set implementations: performance

	Workload 1		Workload 2	
	insert	mem	insert	mem
ListSet	35s	106s	35s	106s
BstSet	130s	149s	0.07s	0.07s

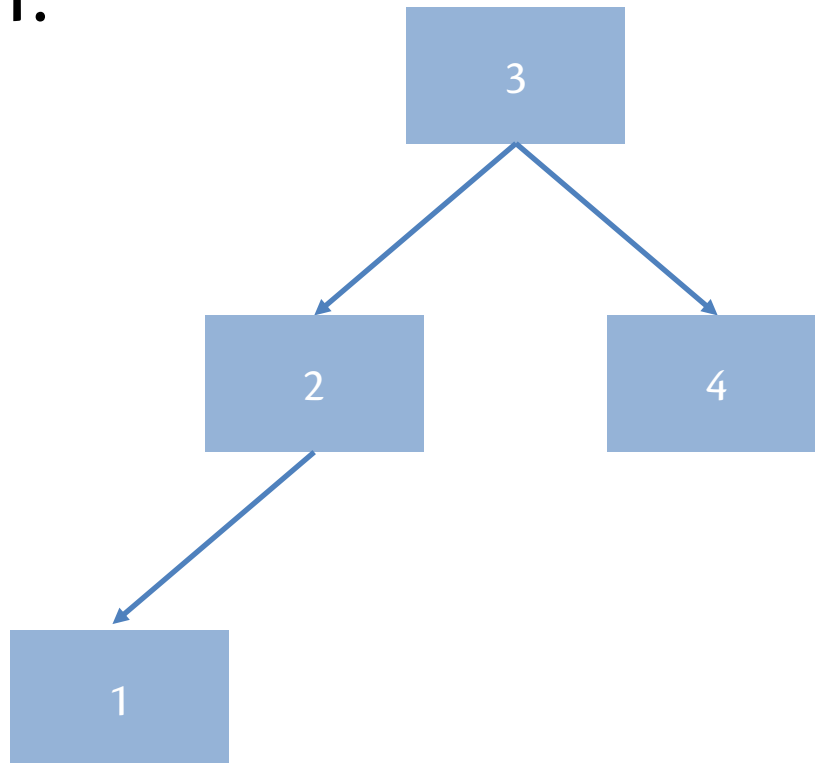
MacBook, 1.3 GHz Intel Core m7, 8 GB RAM, median of three runs

Workloads

- Workload 1:
 - insert: 50,000 elements in ascending order
 - mem: 100,000 elements, half of which not in set
- Workload 2:
 - insert: 50,000 elements in random order
 - mem: 100,000 elements, half of which not in set

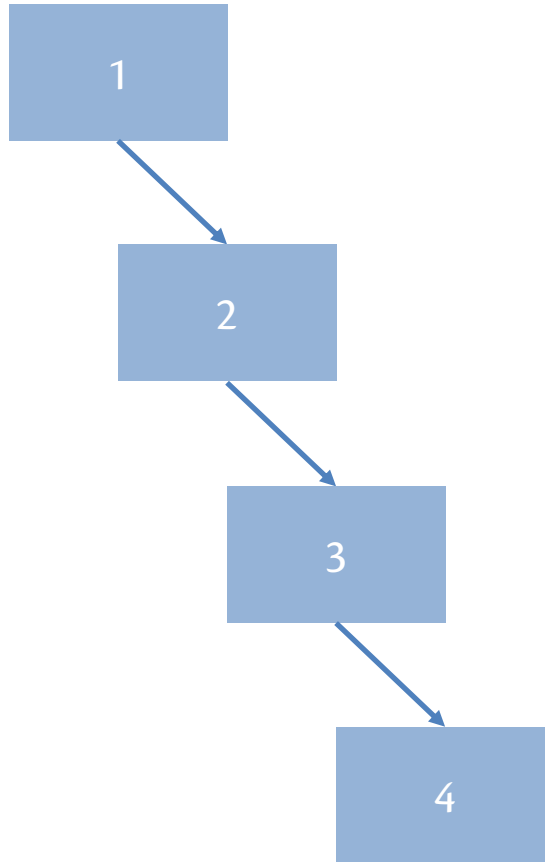
Insert in random order

- Resulting tree depends on exact order
- One possibility for inserting 1..4 in random order
3, 2, 4, 1:



Insert in linear order

Only one possibility for inserting 1..4 in linear order
1, 2, 3, 4:

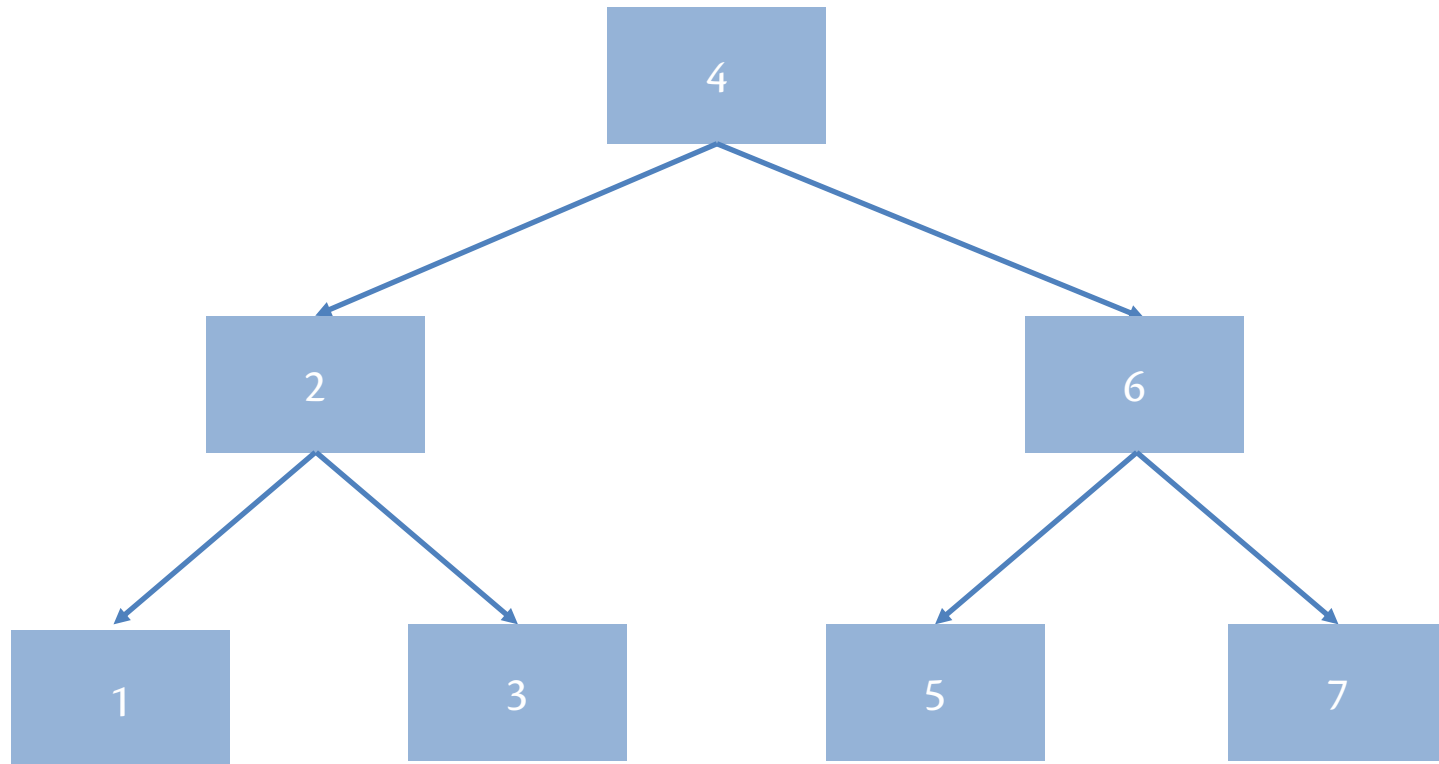


unbalanced: leaning toward the right

When trees get big

- Inserting next element in linear tree **always** takes n operations where n is number of elements in tree already
- Inserting next element in randomly-built tree **might** take far fewer...

Best case tree



all paths through *perfect binary tree* have same length: $\log_2 (n+1)$,
where n is the number of nodes,
recalling there are implicitly leafs below each node at bottom level

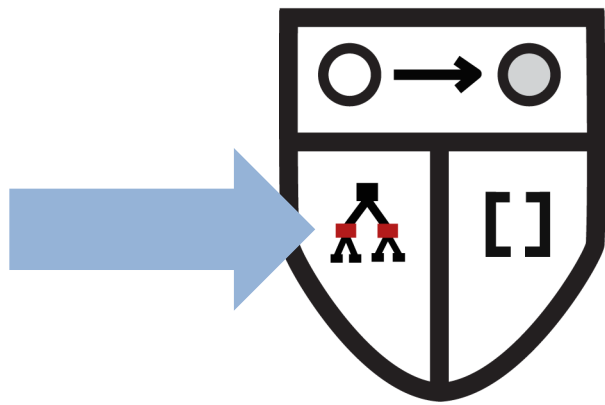
Performance of BST

- `insert` and `mem` are both $O(n)$
- But if trees always had short paths instead of long paths, could be better: $O(\log n)$
- How could we ensure short paths?
i.e., *balance* trees so they don't lean



Strategies for achieving balance

- In general:
 - Strengthen the RI to require balance
 - And modify insert to guarantee it
- Well-known data structures:
 - 2-3 trees: all paths have **same length**
 - AVL trees: length of shortest and longest path from any node **differ at most by one**
 - Red-black trees: length of shortest and longest path from any node **differ at most by factor of two**
- All of these achieve $O(\log n)$ insert and mem



CS 3110

RED-BLACK TREES

Red-black trees

- [Guibas and Sedgewick 1978], [Okasaki 1998]
- Binary search tree with:
 - Each node colored red or black
 - Leaves and root colored black
- RI: BST +
 - **Local invariant:** No red node has a red child
 - **Global invariant:** Every path from the root to a leaf has the same number of black nodes

CLICKER QUESTION 3

Path length

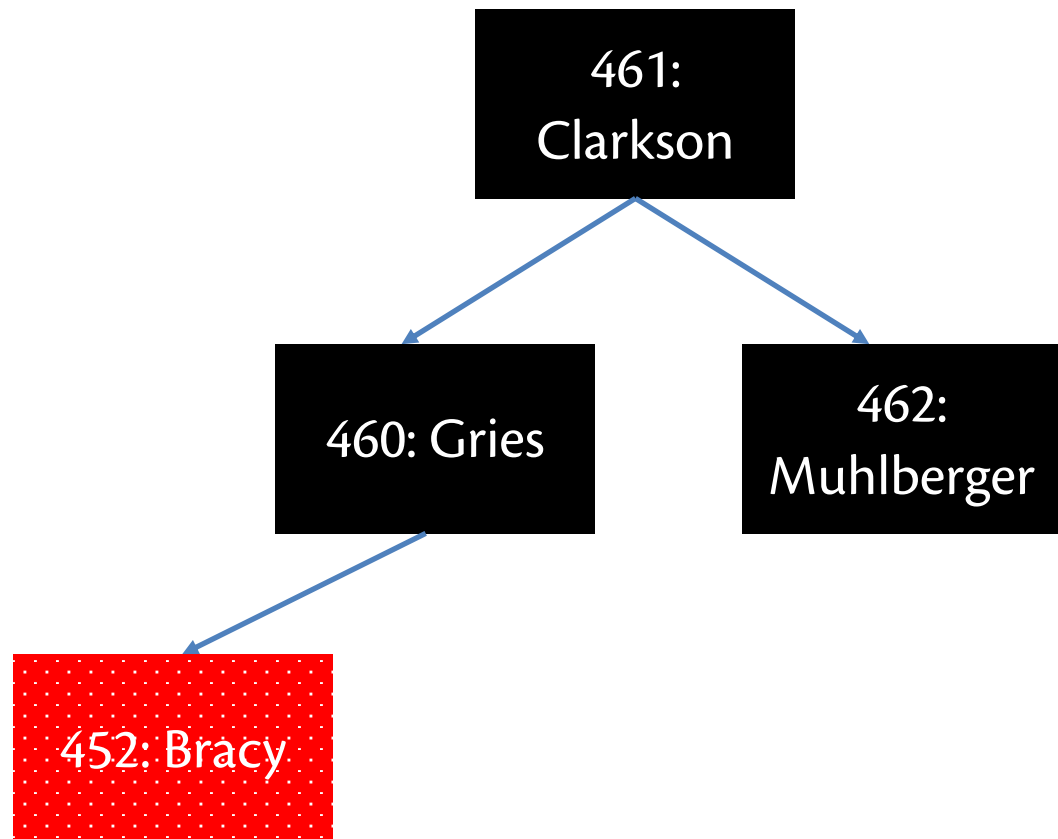
- Invariants:
 - No red node has a red child
 - Every path from the root to a leaf has the same number of black nodes
- Together imply: length of longest path is **at most twice** length of shortest path
 - e.g., B-**R**-B-**R**-B-**R**-B vs. B-B-B-B

Efficiency of red-black set

- mem: $O(\log n)$
 - Worst case: walk down one entire path
 - Length of every path is the same as a perfect tree, or at most twice that long
- insert: $O(\log n)$
 - You will see algorithm in recitation (and textbook)
 - Worst case: walk down then up one entire path

Red-black dictionary

- Store (key, value) pair at each node
- Order as a BST by keys



Map implementations

	insert	find	remove
Arrays	$O(1)$	$O(1)$	$O(1)$
Association lists	$O(1)$	$O(n)$	$O(n)$
Hash tables	$O(1)$	$O(1)$	$O(1)$
Red-black trees	$O(\log n)$	$O(\log n)$	$O(\log n)$

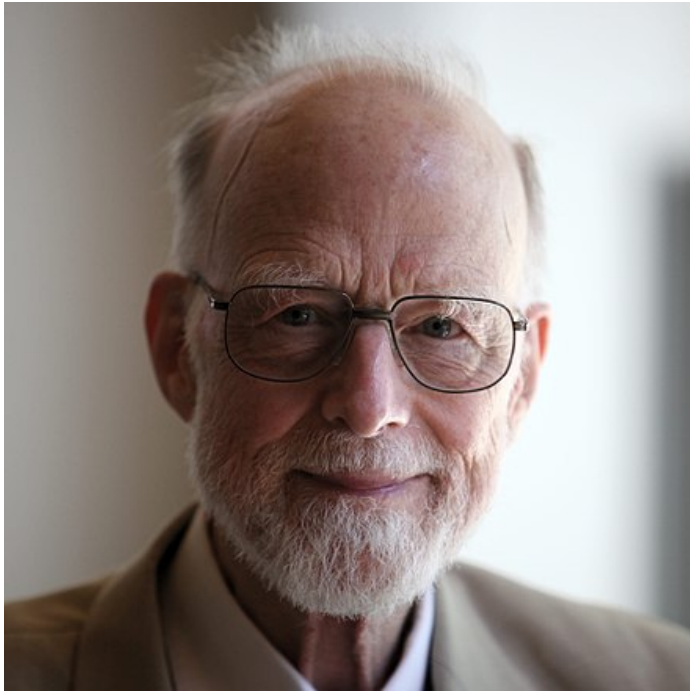
- **Arrays:** fast, but keys must be integers
- **Association lists:** allow any keys, but slower
- **Hash tables:** fast, but requires good hash function; and worst-case performance relaxed to expected & amortized performance
- **Red-black trees:** almost as fast, and immutable

Set implementations: performance

	Workload 1		Workload 2	
	insert	mem	insert	mem
ListSet	35s	106s	35s	106s
BstSet	130s	149s	0.07s	0.07s
RbSet	0.12s	0.07s	0.15s	0.08s

MacBook, 1.3 GHz Intel Core m7, 8 GB RAM, median of three runs

Sir Tony Hoare



b. 1934

Turing Award Winner 1980

For his fundamental contributions to the definition and design of programming languages.

"We should forget about small efficiencies, say about 97% of the time: **premature** optimization is the root of all evil."

Upcoming events

- [later today] A5 released
- [Fri] MS0 due – no late submissions

This is blissfully balanced.

THIS IS 3110