Binary Search Trees

An *ordered* symbol table

Intuition for Binary Search Tree

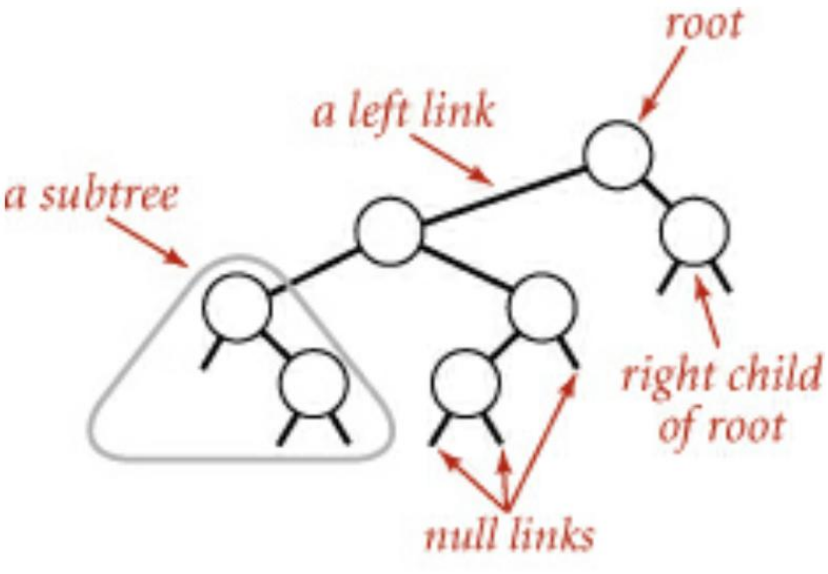
Combine the useful elements of linked lists with the useful elements of ordered arrays:

- Linked lists don't require resizing, making it easier (more flexibility) to add items
- Ordered arrays allow for binary search, making it easier (higher efficiency) to find items (also supports ordered operations)

Solution: Binary Search Tree!

Binary Search Tree

- Recursive definition: A Binary Search Tree is a **null** link or a **Node** with a **left link** and a **right link** that each point to Binary Search Trees
- A Node has one parent (except the root) and two children (left and right)

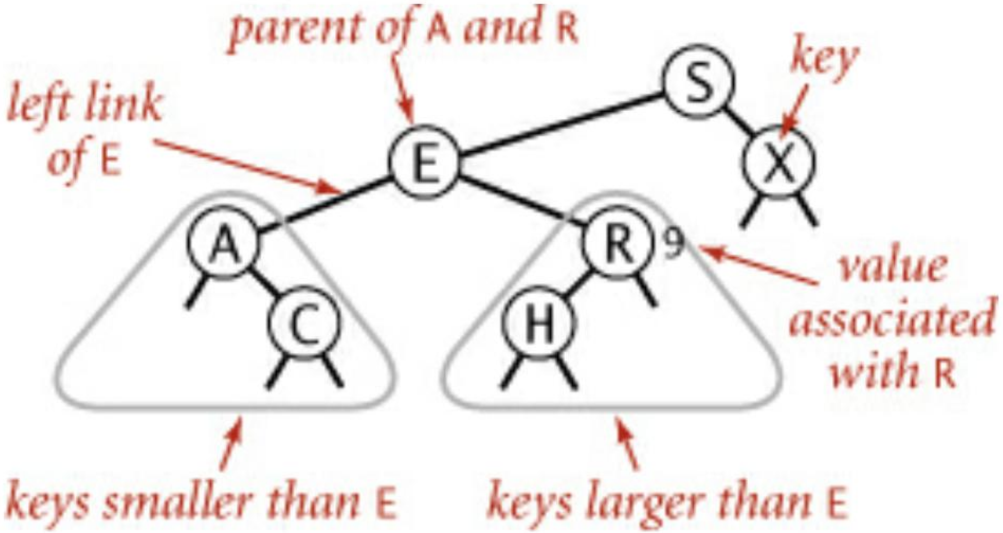


Anatomy of a binary tree

Picture from [1]

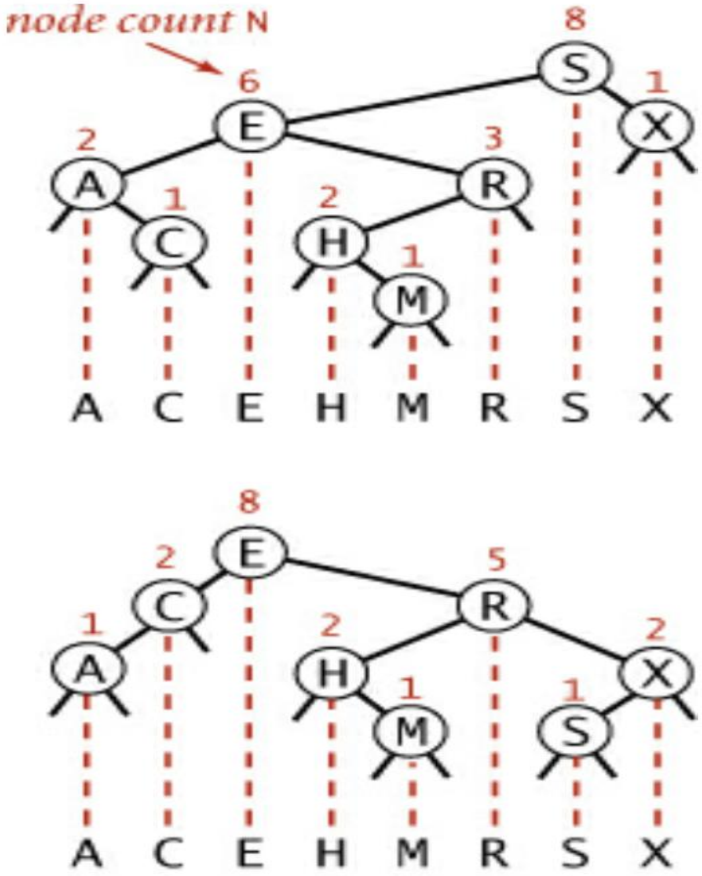
Binary Search Tree

- A Node can have a left child and a right child.
- Nodes are ordered such that any Node is greater than any element in its left subtree and less than any element in its right subtree.
- So an inorder traversal of a BST would yield the keys in sorted order.



Anatomy of a binary search tree

Picture from [1]

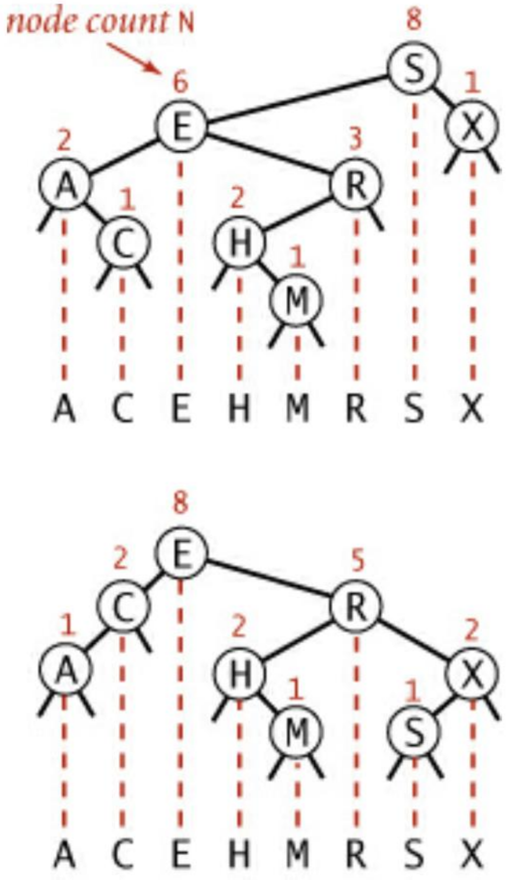


Two BSTs that represent the same set of keys

Picture from [1]

Same keys, different trees…

HOW DOES THIS HAPPEN?

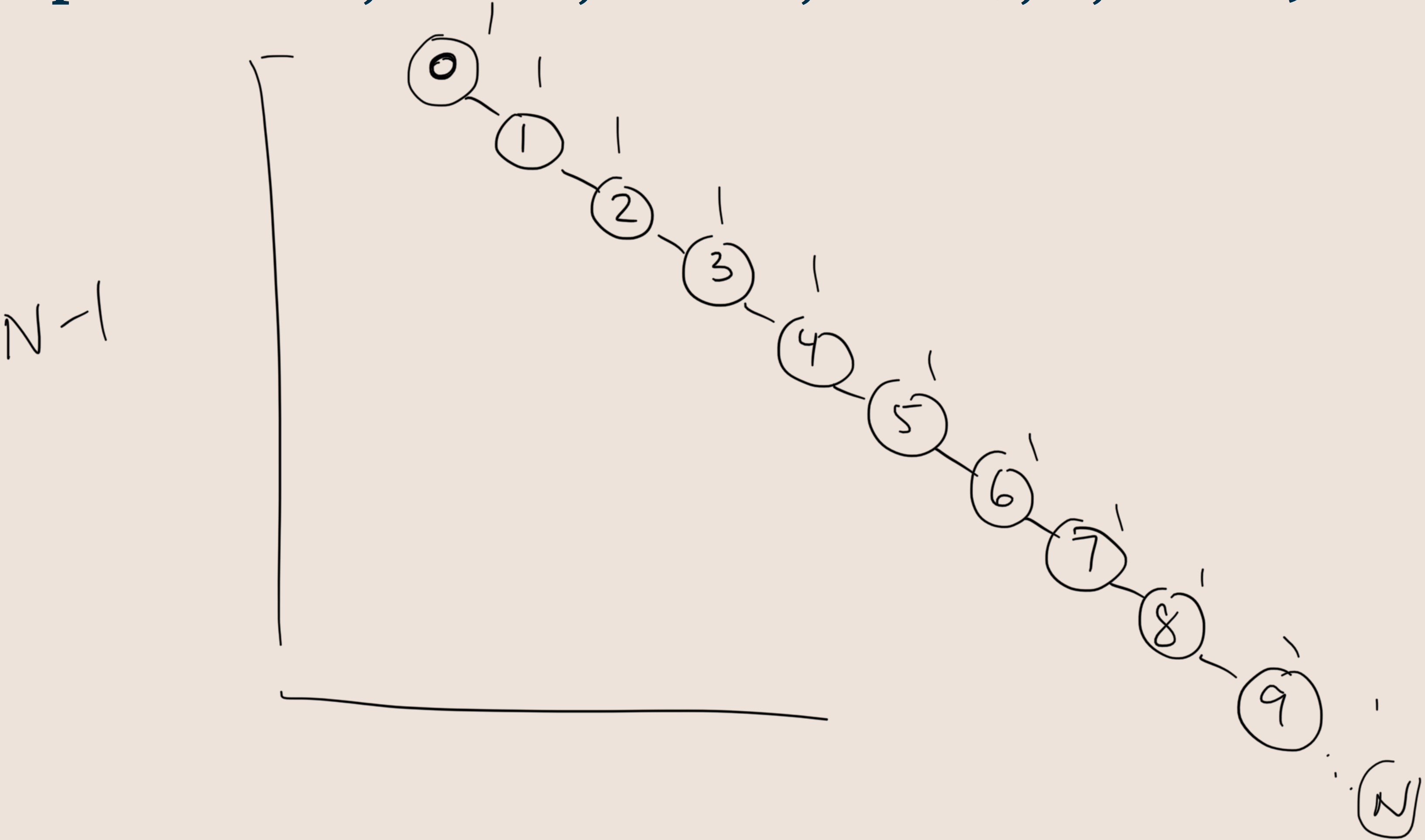


Two BSTs that represent the same set of keys

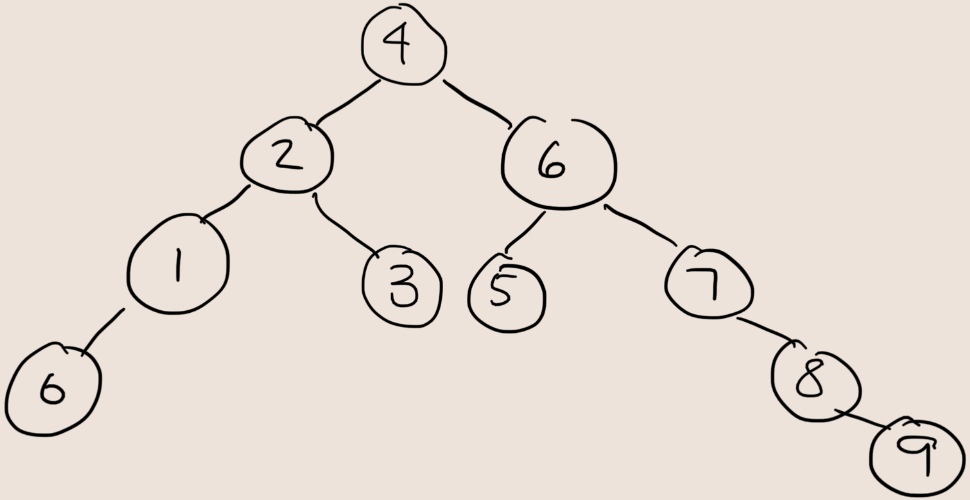
Resulting Binary Trees depend on the order in which elements are inserted, which means the same keys can be organized in very different trees.

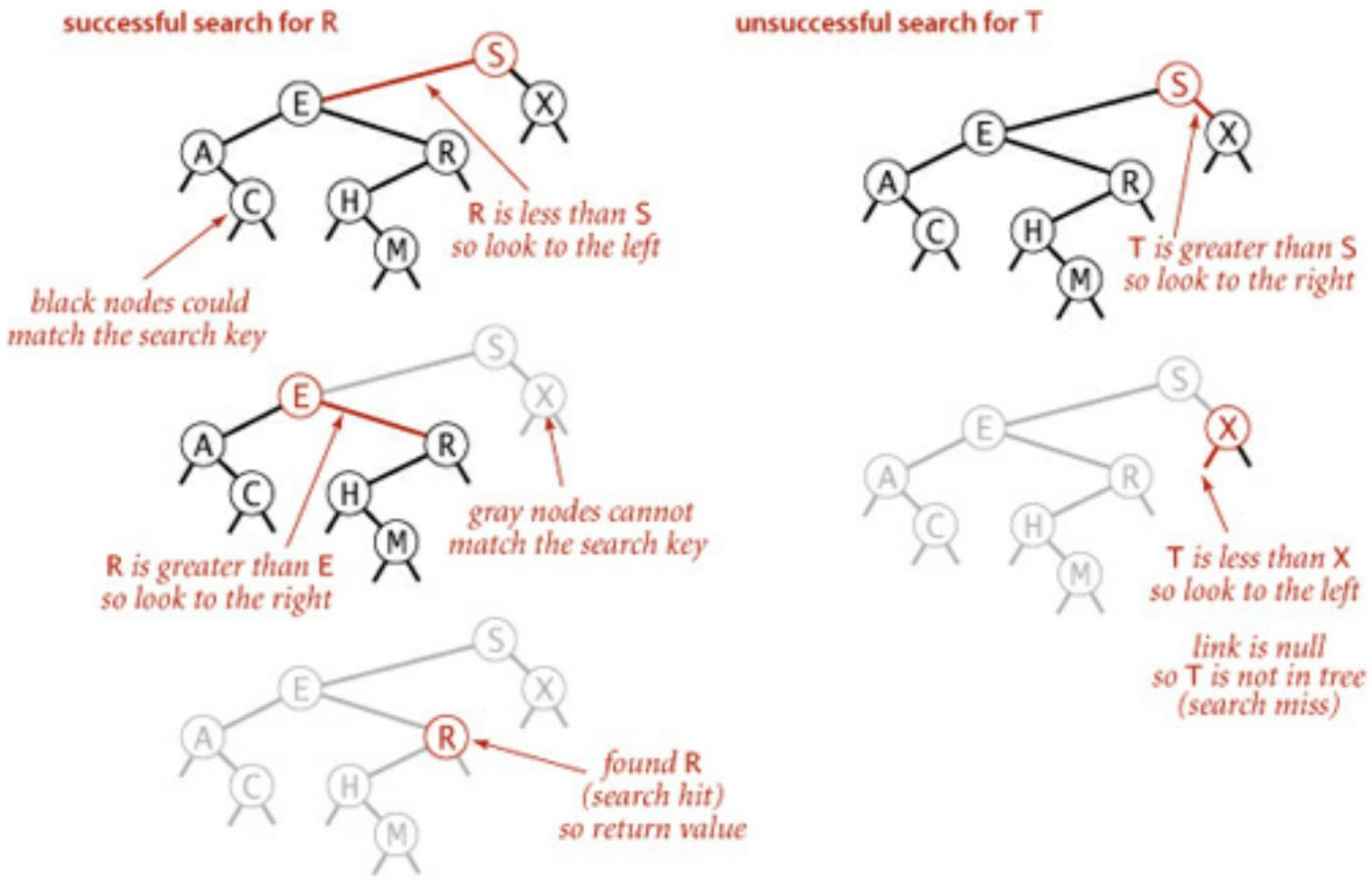
Picture from [1]

Example: *insert 0, insert 1, insert 2, insert 3, ..., insert 9*



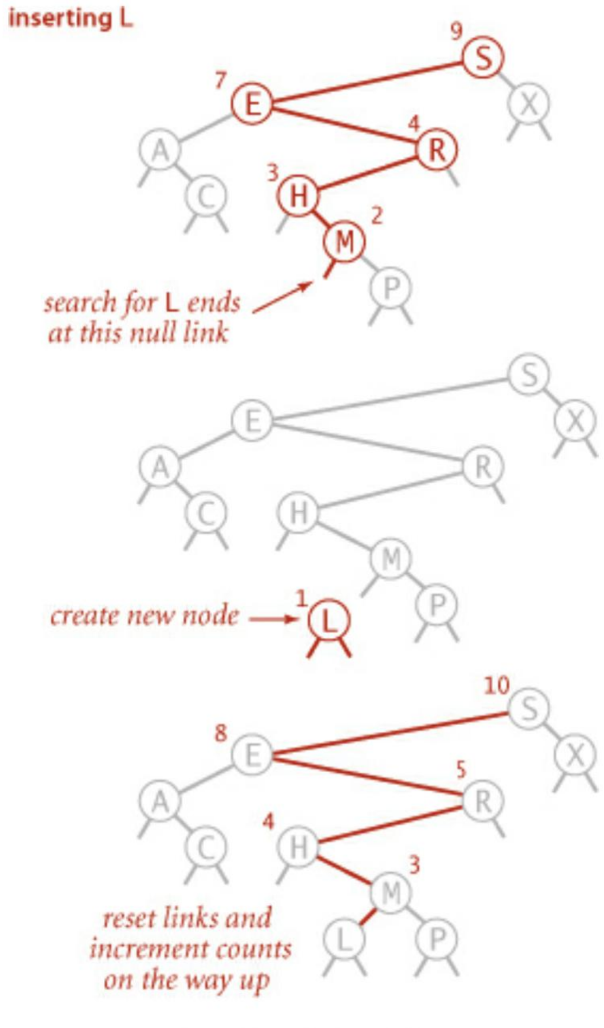
Example: *insert 4, insert 2, insert 6, insert 1, insert 3, insert 5, insert 7, insert 0, insert 8, insert 9*





Search hit (left) and search miss (right) in a BST

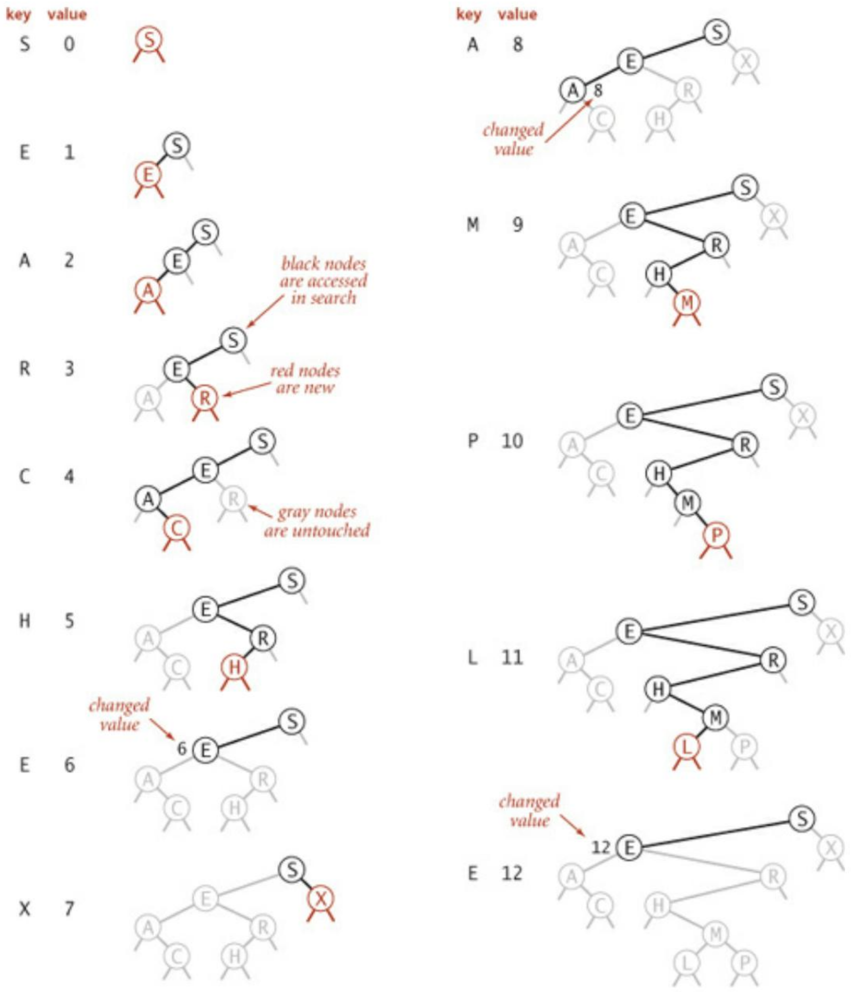
Picture from [1]



Insertion into a BST

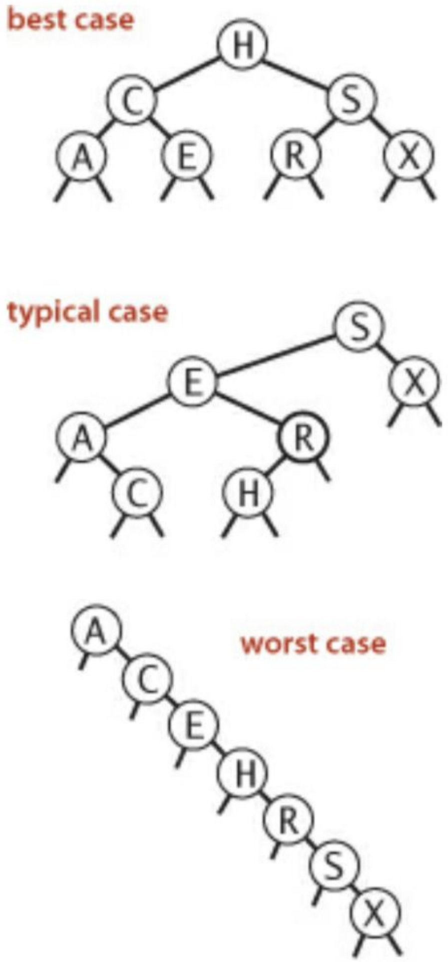
Picture from [1]

- Search for key first
- If it's found, reset the value
- If it isn't found, add a new Node at the null link



BST trace for standard indexing client

Picture from [1]



BST possibilities

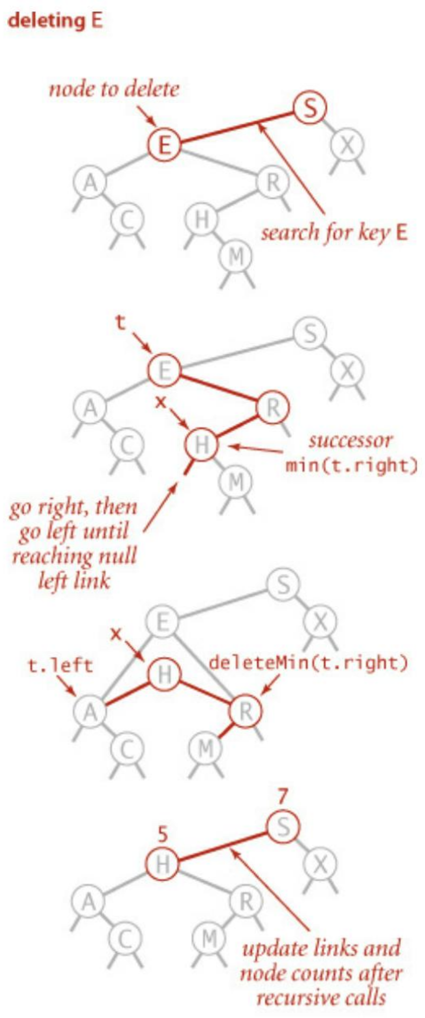
algorithm (data structure)	worst-case cost (after N inserts) search	worst-case cost (after N inserts) insert	average-case cost (after N random inserts) search hit	average-case cost (after N random inserts) insert	efficiently support ordered operations?
sequential search (unordered linked list)	N	N	$N/2$	N	no
binary search (ordered array)	$\lg N$	N	$\lg N$	$N/2$	yes
binary tree search (BST)	N	N	$1.39 \lg N$	$1.39 \lg N$	yes

Cost summary for basic symbol-table implementations (updated)

The good news is that the typical case is closer to the best case than the worst case!

Pictures from [1]

BST Deletion



Deletion in a BST

Picture from [1]

- Option 1: key doesn't exist in tree

  do nothing

- Option 2: key exists and the node has no children

  delete it

- Option 3: key exists and the node has one child

  make parent pointer point to node's child

- Option 4: key exists and the node has two children

  replace node w/ successor

  delete successor

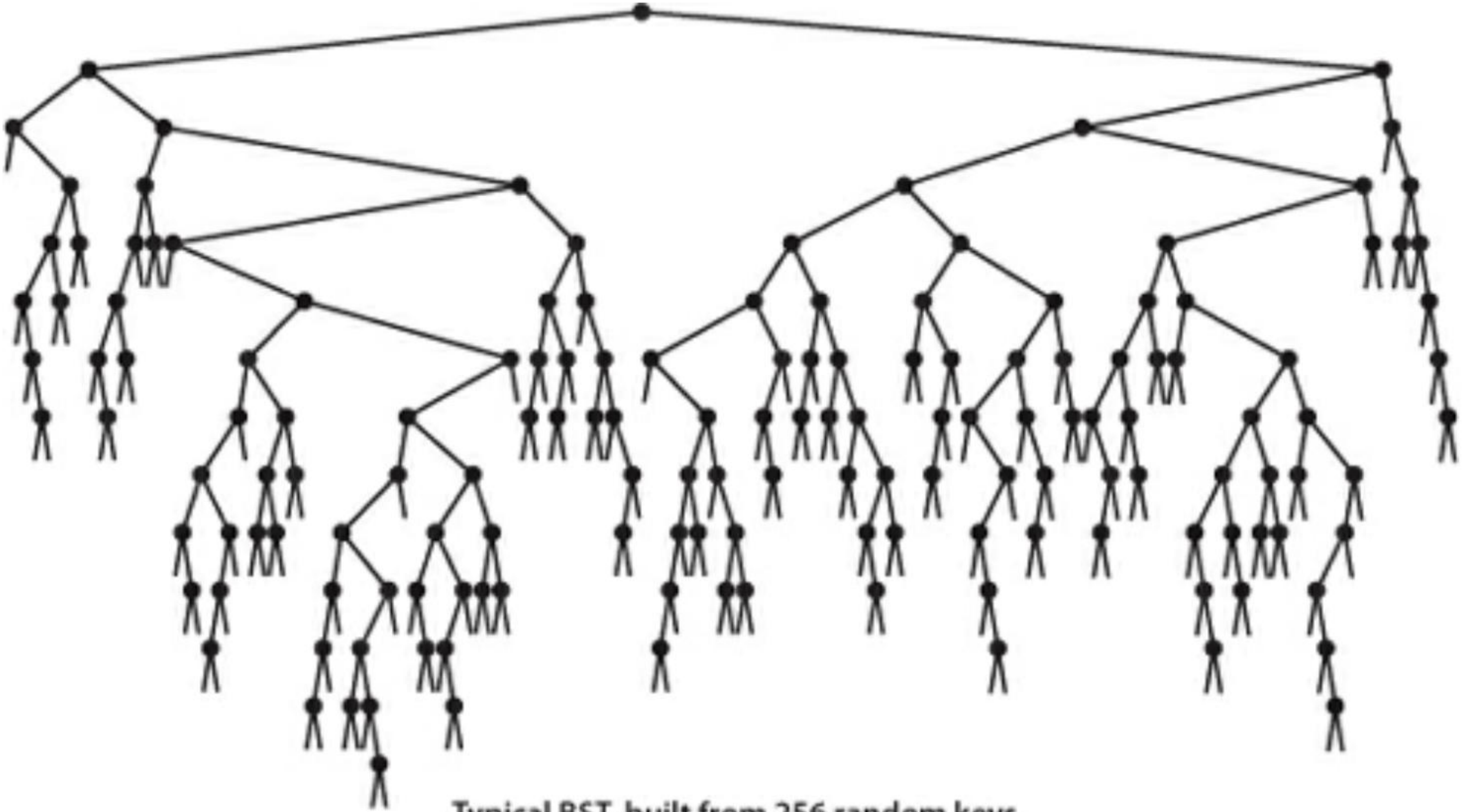
Example: *insert 4, insert 2, insert 6, insert 1, insert 3, insert 5, insert 7, insert 0, insert 8, insert 9, ~~delete 9~~, ~~delete 7~~, ~~delete 4~~, ~~delete 2~~, ~~delete 3~~, ~~delete 6~~, ~~delete 0~~, delete 1, delete 5, delete 8*

Analysis Summary

algorithm (data structure)	worst-case cost (after N inserts)		average-case cost (after N random inserts)		efficiently support ordered operations?
	search	insert	search hit	insert	
sequential search (unordered linked list)	N	N	$N/2$	N	no
binary search (ordered array)	$\lg N$	N	$\lg N$	$N/2$	yes
binary tree search (BST)	N	N	$1.39 \lg N$	$1.39 \lg N$	yes

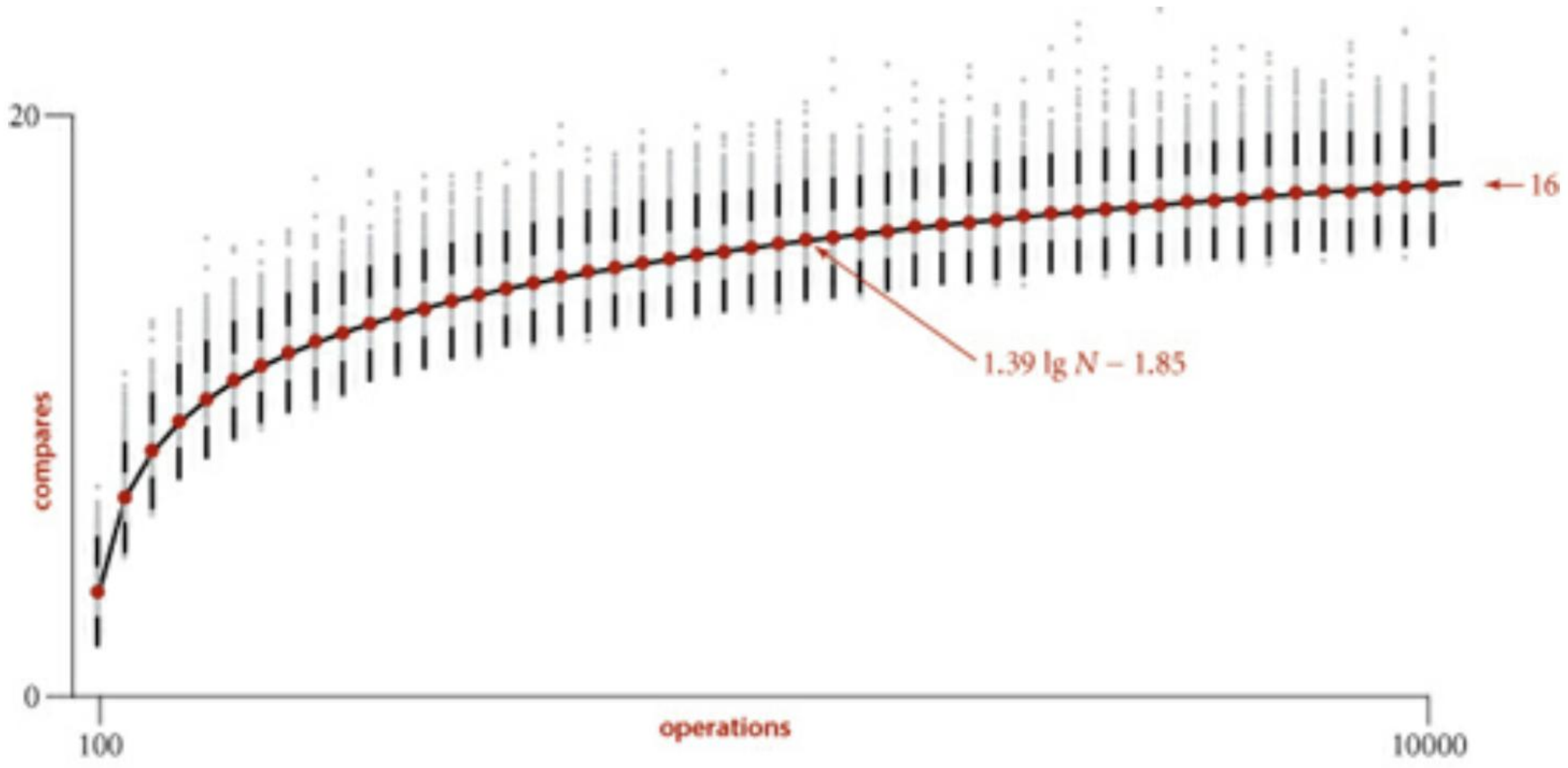
Cost summary for basic symbol-table implementations (updated)

Picture from [1]



Typical BST, built from 256 random keys

Picture from [1]



Average path length to a random node in a BST built from random keys

Picture from [1]

Binary Search Trees

Pros

- Easy to implement
- Good average performance for basic operations

Cons

- Bad worst case performance for basic operations
- WHY?

How can we improve upon this?

How can we improve upon this?

ENFORCE BALANCE

References

[1] *Algorithms, Fourth Edition;* Robert Sedgewick and Kevin Wayne (and associated slides)
[2] Slides from https://www.cs.princeton.edu/~rs/talks/LLRB/RedBlack.pdf