CSC 212
Data Structures & Algorithms

Fall 2022 | Jonathan Schrader

Binary Search Trees

Housekeeping

Lab 8: Binary Search Trees

Election Day / Veteran's Day

- Nov 7-11
- Class only meets Thursday, Nov 10
- Assignment 4 Due
- Lab 9: Balancing Act Due
  - In-person labs are canceled

Term Project

K-ARY TREES

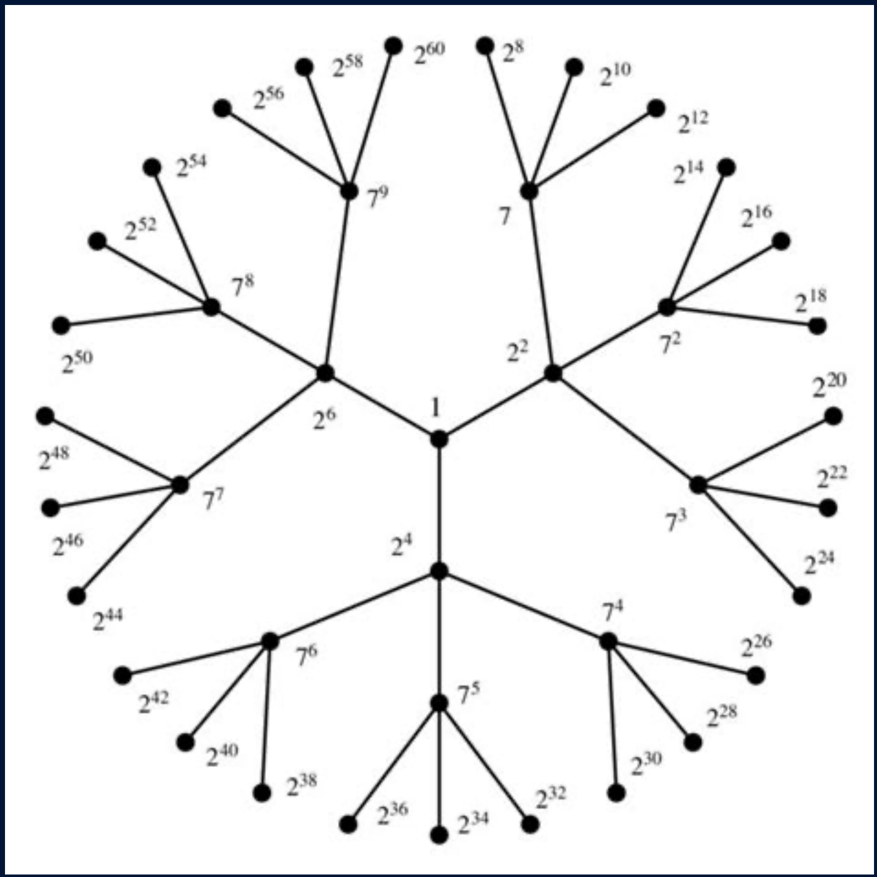
k-ary trees

In a **k-ary tree**, every node has between 0 and k children

In a **full (proper)** k-ary tree, every node has exactly 0 or k children

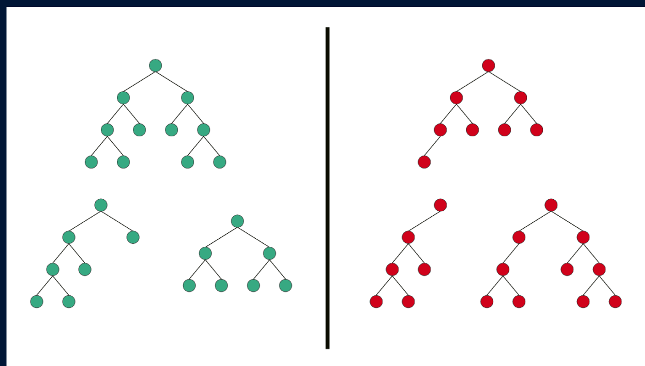
In a **complete** k-ary tree, every level is entirely filled, except possibly the deepest, where all nodes are as far left as possible

In a **perfect** k-ary tree, every leaf has the same depth and the tree is full

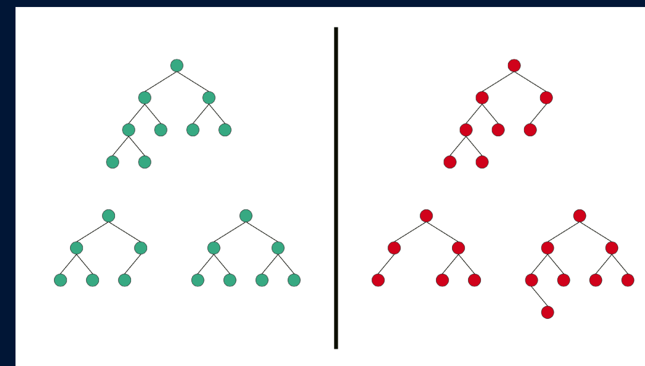


3-ary tree

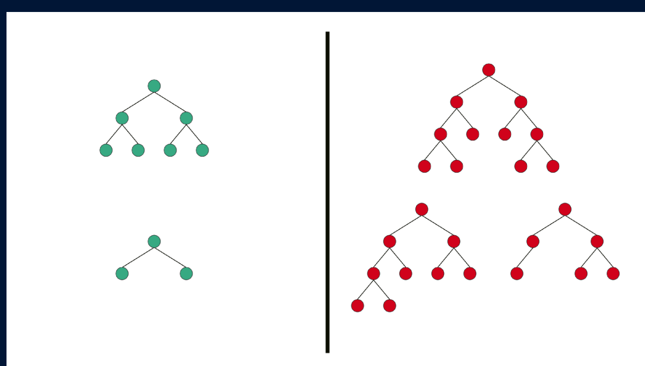
examples



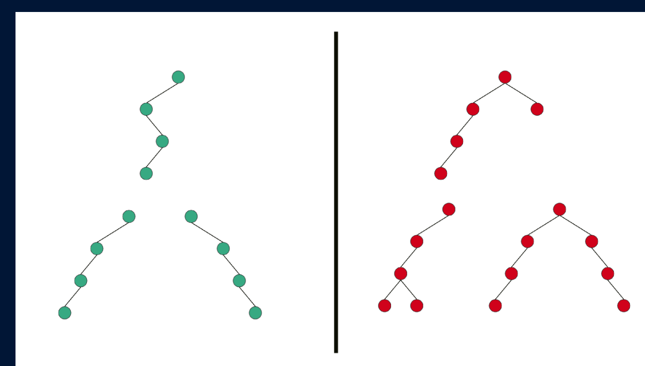
$full$



$complete$

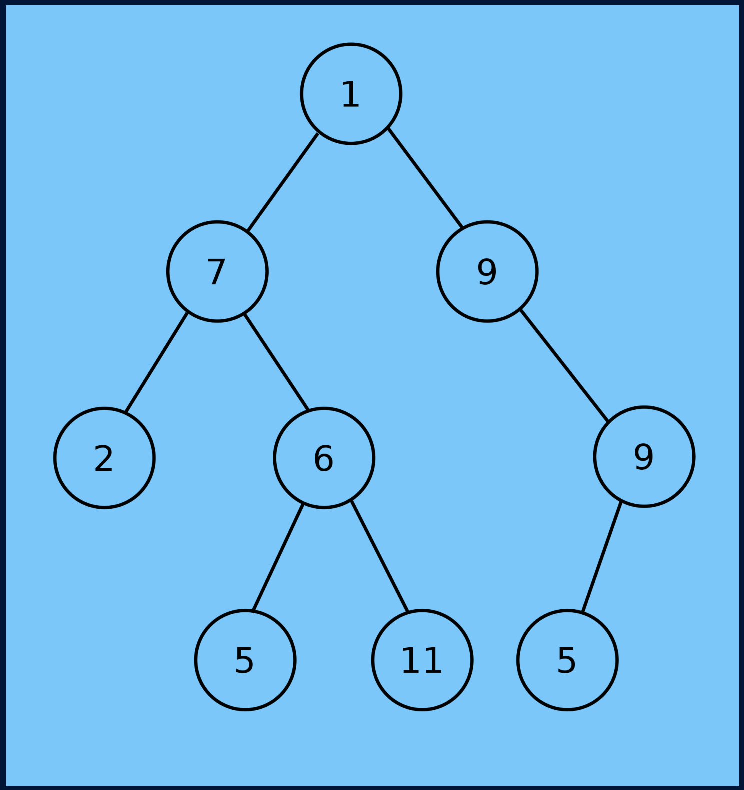


$perfect$



$degenerate$

Binary Tree



$$bin\_tree = \{1, 7, 9, 2, 6, 9, 5, 11, 5\}$$

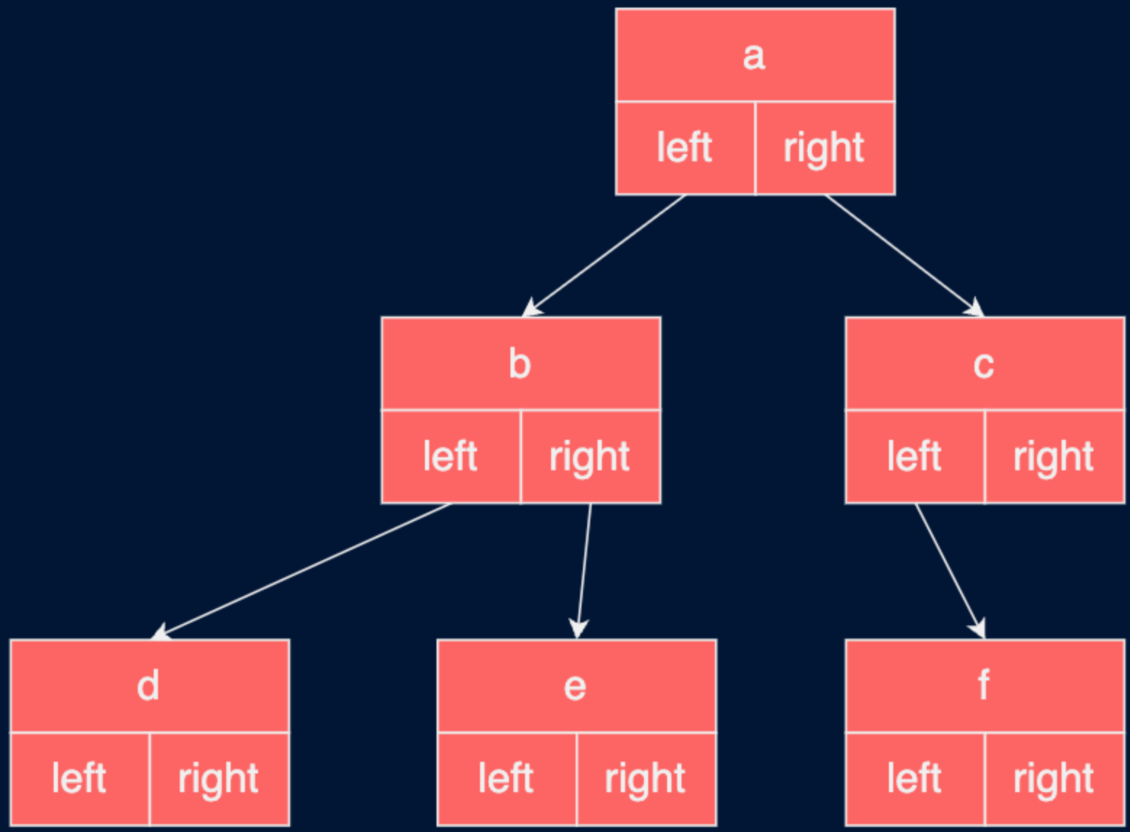
Implementing binary trees

Node

$data$
$left$ child
$right$ child

Tree

$bst = \{a, b, c, d, e, f\}$



BINARY SEARCH TREES

Binary Search Tree

A BST is a binary tree

A BST has symmetric order

- each node x in a BST has a key

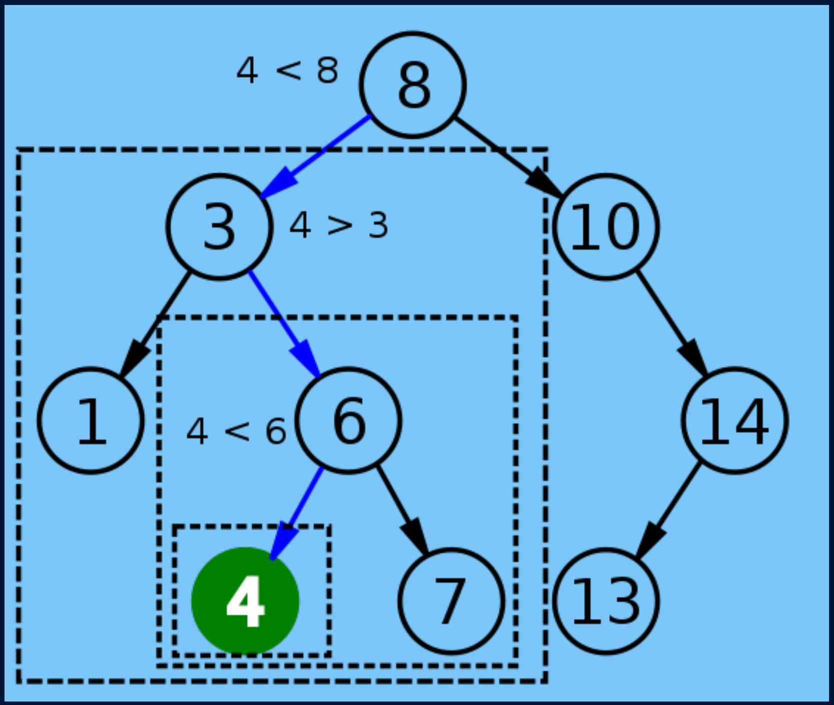
$$key(x)$$

- for all nodes y in the left subtree of x,

$$key(y) < key(x)^{**}$$

- for all nodes y in the right subtree of x,

$$key(y) > key(x)^{**}$$



(**) assume that the keys of a BST are pairwise distinct

BST Classes

```
class BSTNode {

  private:
    int data;
    BSTNode *left;
    BSTNode *right;

  public:
    BSTNode(int d);
    ~BSTNode();

  friend class BSTree;

};
```

```
class BSTree{

  private:
    BSTNode *root;
    void destroy(BSTNode *p);

  public:
    BSTree();
    ~BSTree();
    void insert(int d);
    void remove(int d);
    BSTNode *search(int d);

};
```

SEARCH INTO BSTs

search

1) Start at root node

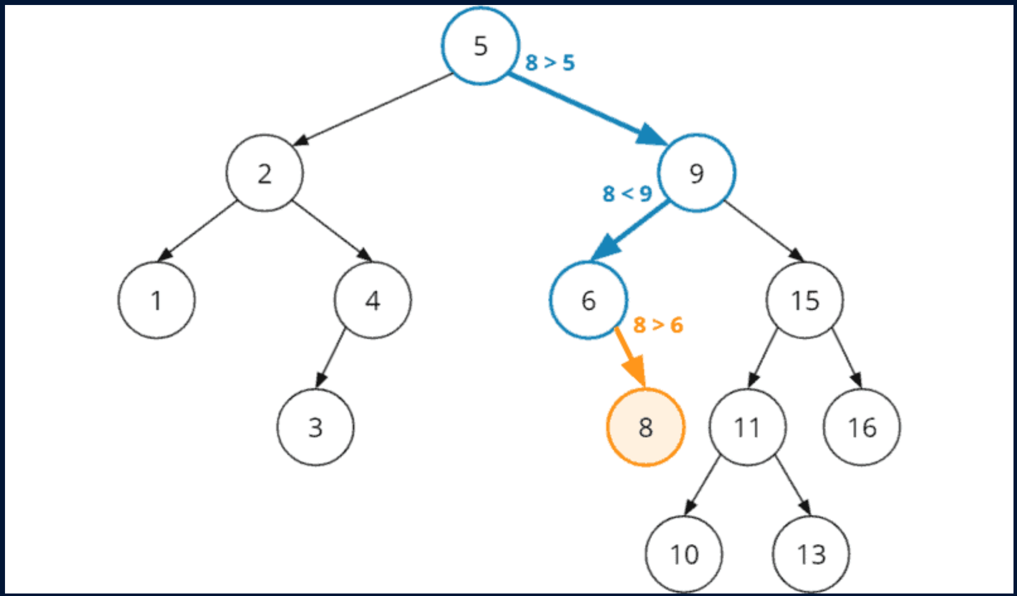
2) If the search key:

a) matches the current node's key
- then found

b) If search key $>$ current node's key
- search on $right$ child

c) If search key is less than current node's
- search on $left$ child

3) Stop when current node is NULL (not found)



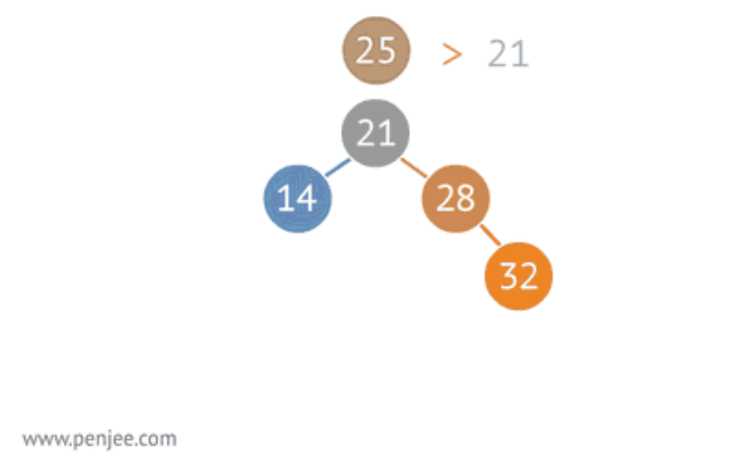
$Search\ for\ 8$

INSERT INTO BSTs

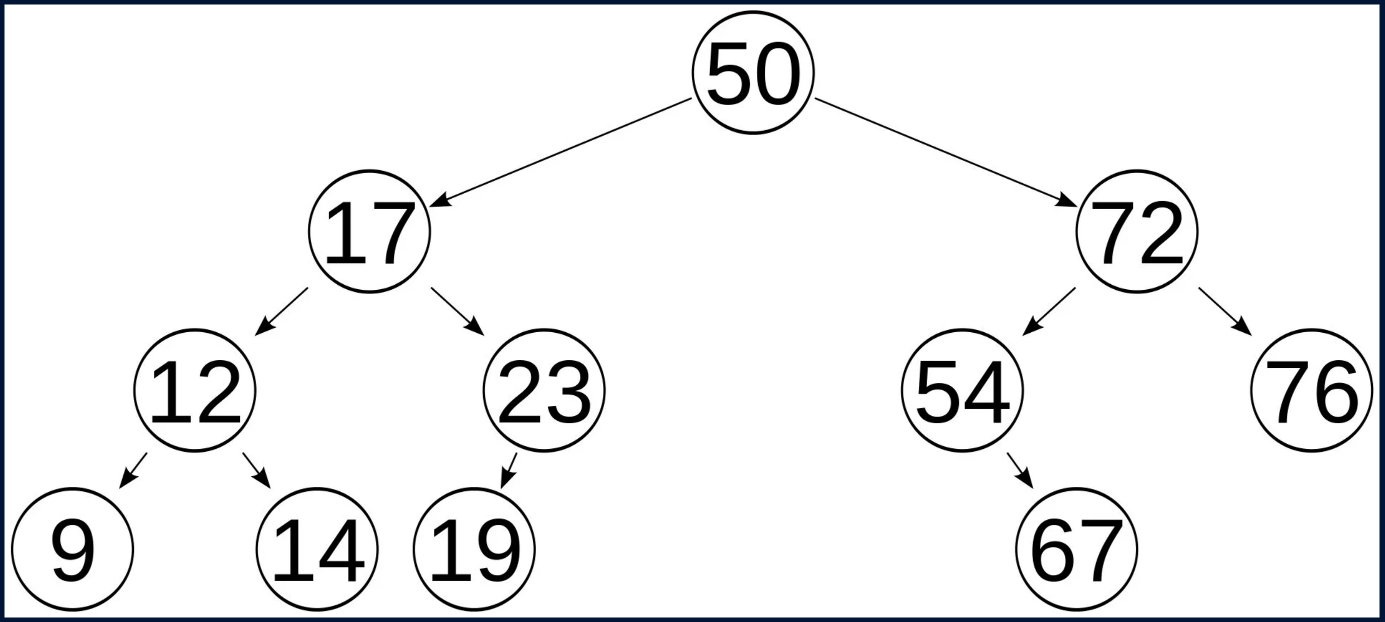
insert

Perform a Search operation

- If found, no need to insert (may increase counter)
- If not found, insert node where Search stopped



Try it…

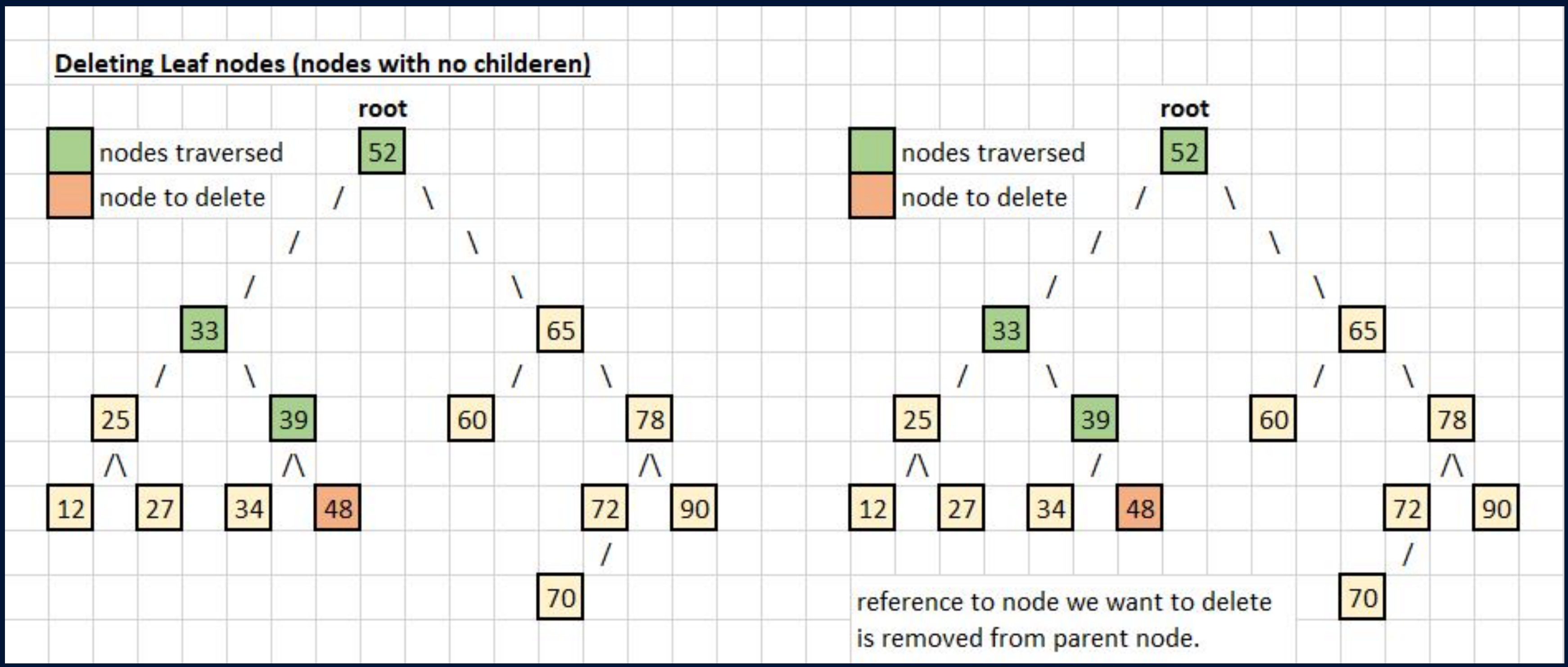


$$Serach...\ 23, 67, 18...$$

$$Insert...\ 65, 27, 90, 11, 51...$$

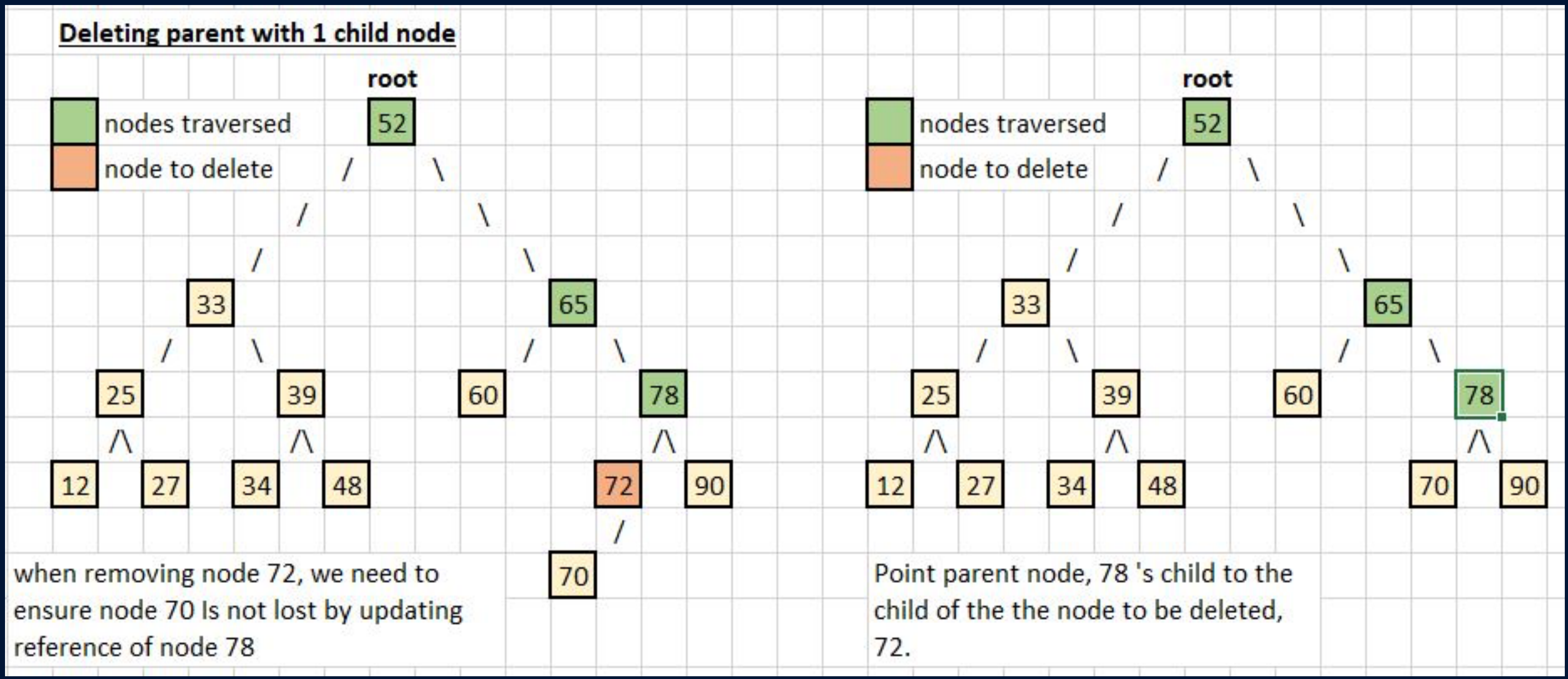
REMOVE FROM BSTs

$remove : \ leaf$



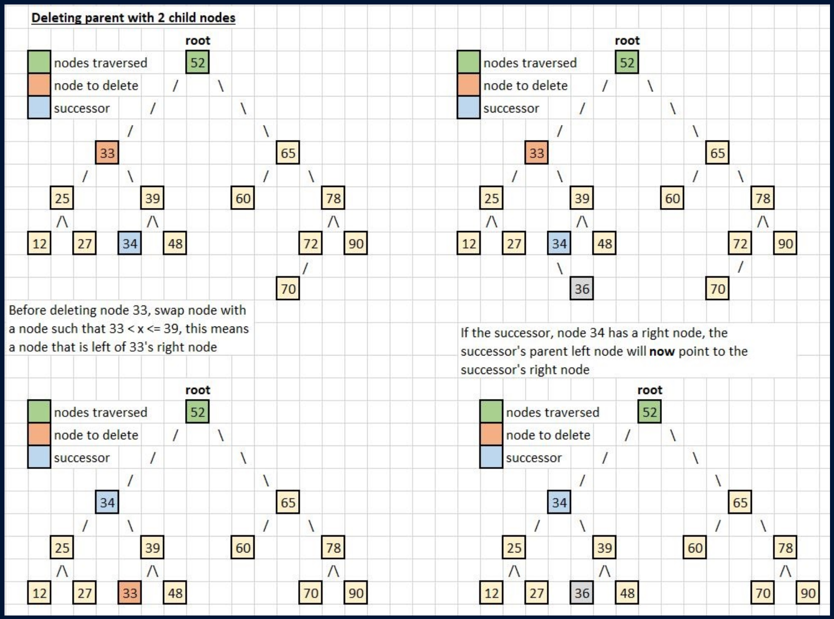
$trivial, \ delete \ node \ and \ set \ parent's \ pointer \ to \ NULL$

remove : with 1 child



trivial, set parent's pointer to the only child andelete node

$remove: \ with\ 2\ children$



$find\ successor$

$copy\ successor's\ data\ to\ node$

$delete\ successor$

BST TRAVERSALS

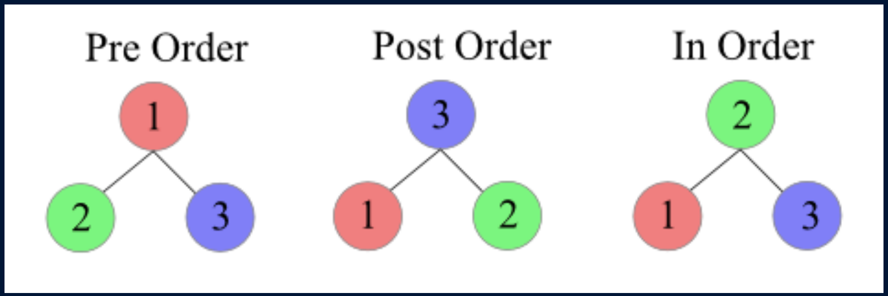
traversals

$O(n)$

```
algorithm preorder (p) {
  if (p) {
    visit(p)
    inorder(p -> left)
    inorder(p -> right)
  }
}
```

```
algorithm postorder (p) {
  if (p) {
    inorder(p -> left)
    inorder(p -> right)
    visit(p)
  }
}
```

```
algorithm inorder (p) {
  if (p) {
    inorder(p -> left)
    visit(p)
    inorder(p -> right)
  }
}
```

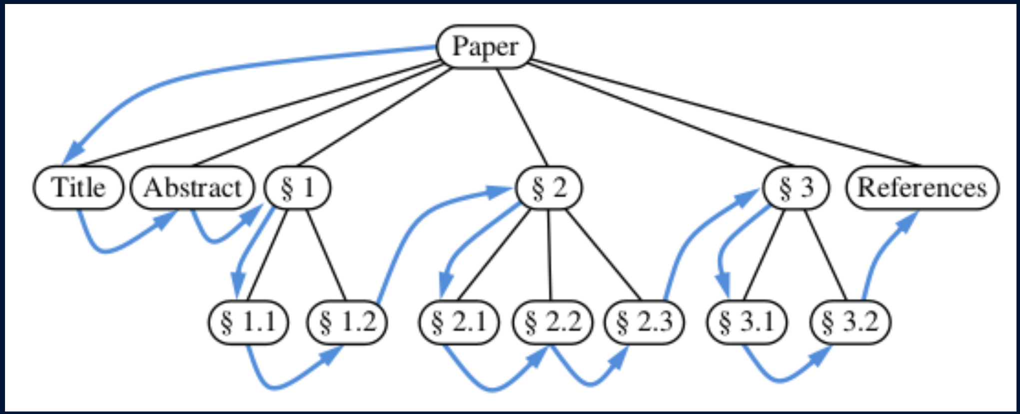


How would we:

- Destroy a binary tree
- Print all elements ascending order

$traversal:\ preorder$

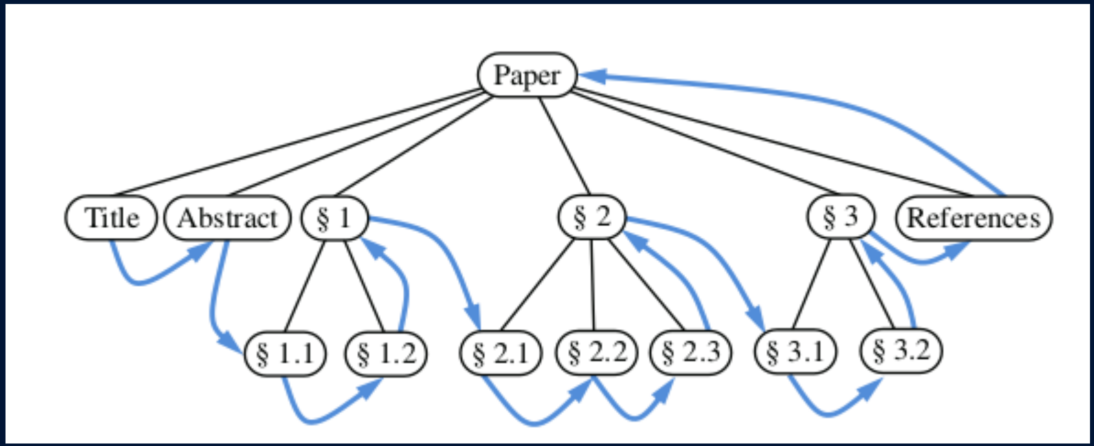
```
algorithm preorder (p) {
  if (p) {
    visit(p)
    inorder(p -> left)
    inorder(p -> right)
  }
}
```



https://sbme-tutorials.github.io/2020/data-structure-FALL/notes/week08.html#preorder-traversals

traversal : postorder

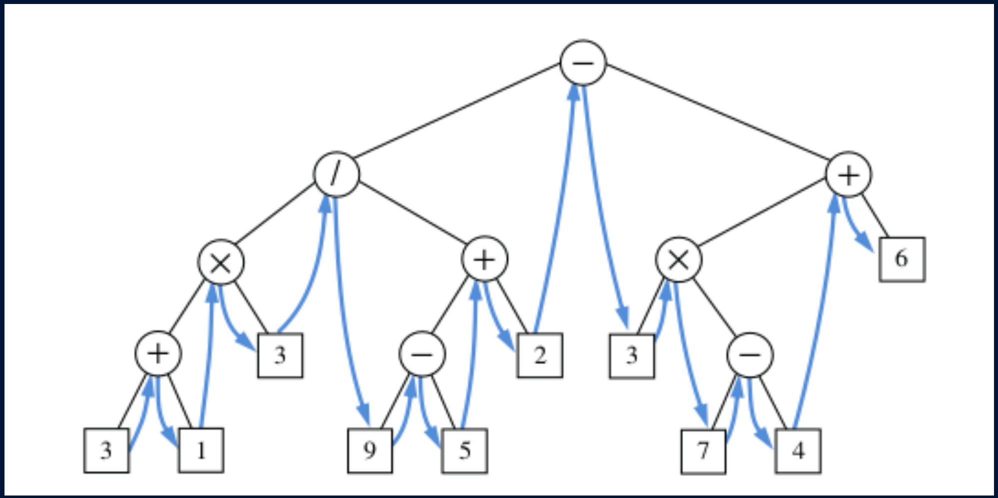
```
algorithm postorder (p) {
  if (p) {
    inorder(p -> left)
    inorder(p -> right)
    visit(p)
  }
}
```



https://sbme-tutorials.github.io/2020/data-structure-FALL/notes/week08.html#postorder-traversals

$traversal: \ inorder$

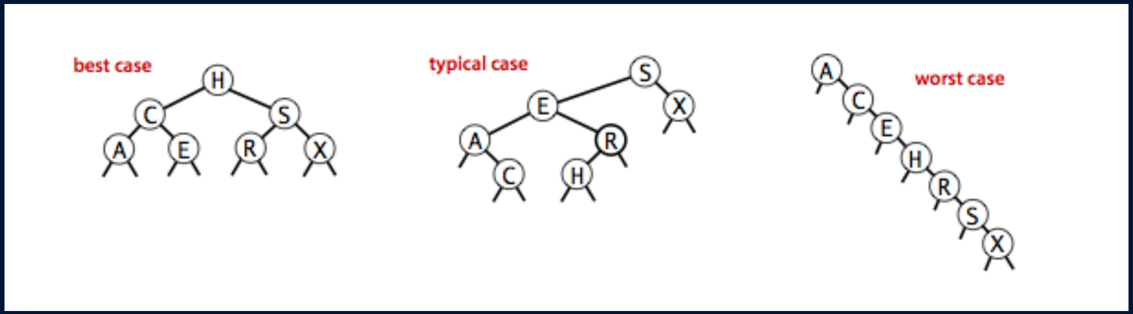
```
algorithm inorder (p) {
  if (p) {
    inorder(p -> left)
    visit(p)
    inorder(p -> right)
  }
}
```



https://sbme-tutorials.github.io/2020/data-structure-FALL/notes/week08.html#inorder-traversals

ANALYSIS

Tree Shape



$$best\ case = \{H, C, S, A, E, R, X\} \qquad typical = \{S, E, A, R, C, H, X\}$$

$$order\ of\ operations\ matter\ldots$$

Implications

$$Cost\ of\ basic\ operations??$$

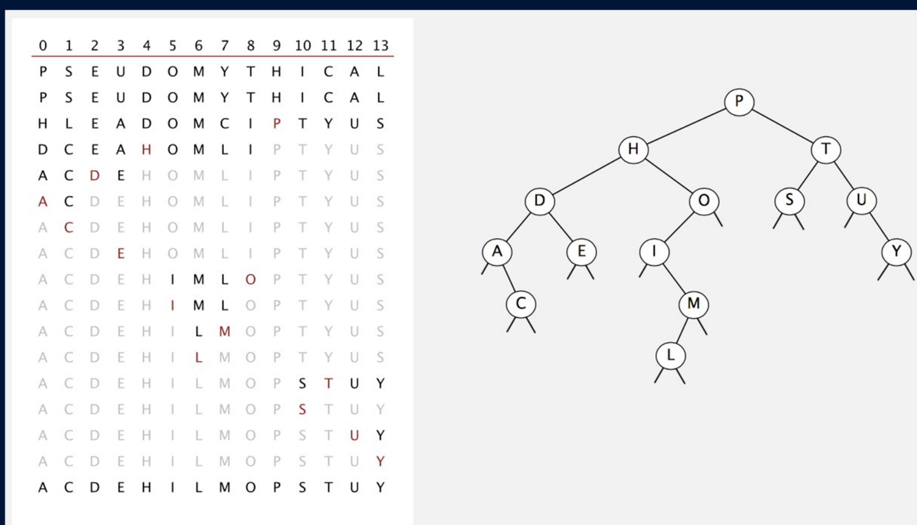
	best-case	worst-case	average-case
$search$			
$insert$			
$remove$			

Average-case analysis

If n **distinct keys** are inserted into a BST in random order, expected number of compares for basic operations is

$$\sim 2\ ln\ n \approx 1.39\ log\ n$$

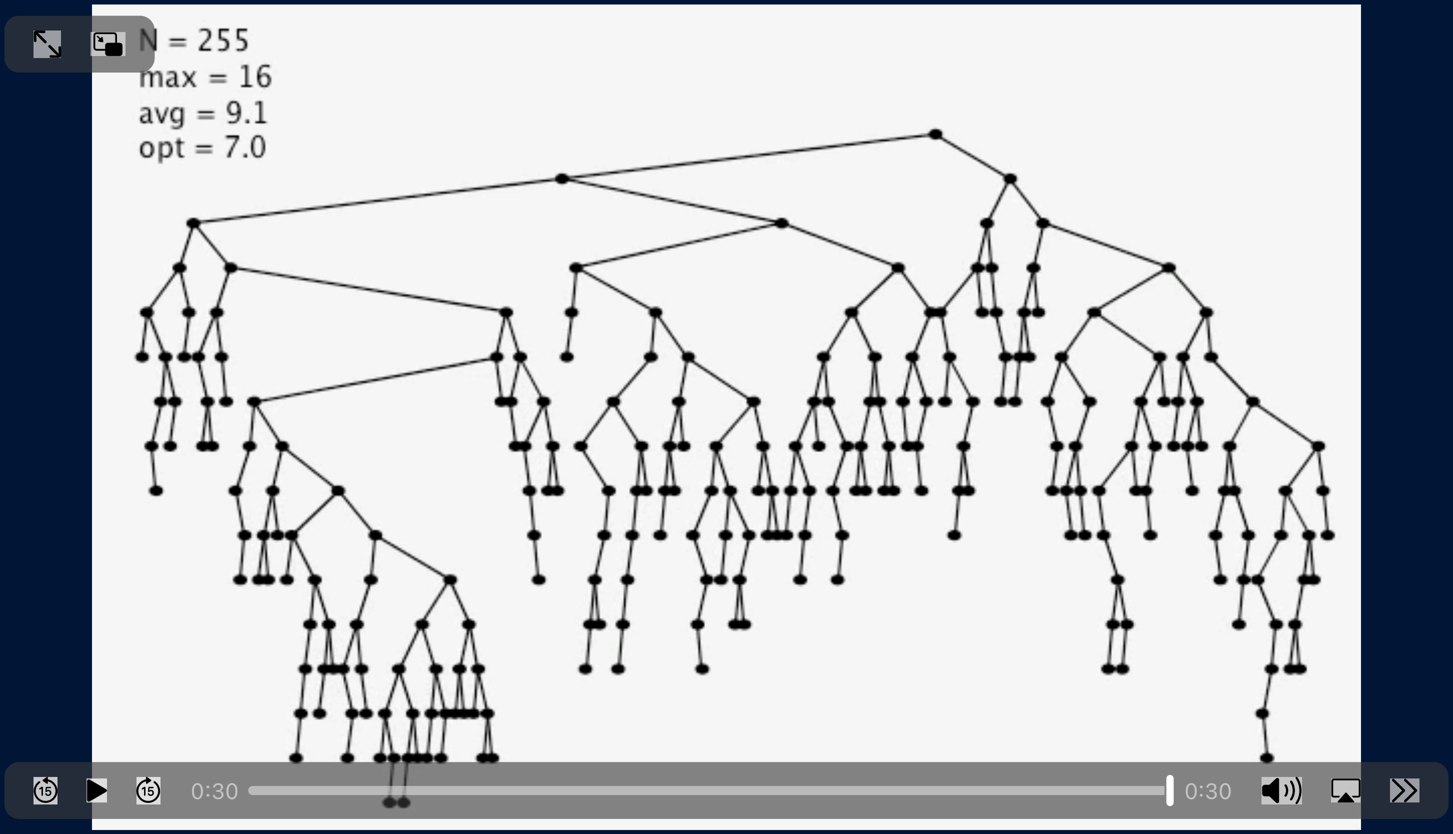
- proof: 1-1 correspondence with quick-so



$$h = O(log\ n)$$

inserting n keys in a BST in random order

$n = 255$



https://algs4.cs.princeton.edu/32bst/

Collections / Dictionaries

	What?	Sequential (unordered)	Sequential (ordered)	BST
search	search for a key	$O(n)$	$O(log\ n)$	$O(h)$
insert	insert a key	$O(n)$	$O(n)$	$O(h)$
delete	delete a key	$O(n)$	$O(n)$	$O(h)$
min/max	smallest/largest key	$O(n)$	$O(1)$	$O(h)$
floor/ceiling	predecessor / successor	$O(n)$	$O(log\ n)$	$O(h)$
rank	# of keys less than key	$O(n)$	$O(log\ n)$	$O(h)$**

(**) requires the use of ‘size’ at every node)