CSE 326: Data Structures
Branching Out

Hannah Tang and Brian Tjaden
Summer Quarter 2002

Today's Outline

- Introduction to Project #2
- ADT Redux
- *New!* Dictionary and Search ADTs
- (Binary) Tree Review
  - Recursive Operations on Binary Trees
- Unbalanced Binary Search Trees
  - Operations: Insert, Remove

ADT Redux

What operations do these ADT's support? What interesting properties do they have?

- List ADT
- Stack ADT
- Queue ADT
- Priority Queue ADT

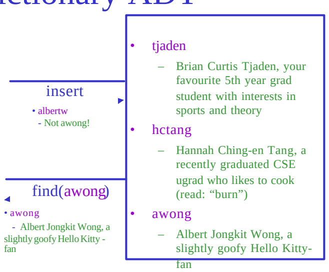
Comparing ADT's We Know

How are Lists/Stacks/Queues similar to PriorityQueues? How do they differ?

- Similarities:
- Differences:

New! The Dictionary ADT

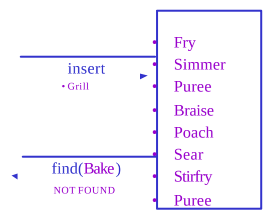
- Dictionary ADT:
  - Maps *values* to user-specified *keys*
  - Or: a set of (key, value) pairs
    - Keys may be any (homogeneous) type
    - Values may be any (homogeneous) type
- Operations:
  - Insert
  - Find
  - Remove



The Dictionary ADT is sometimes called the "Map ADT"

Also New! The Search ADT

- Search ADT:
  - Contains unique user-specified *keys*
  - Or: a set of keys
    - Keys may be any (homogeneous) type
- Operations:
  - Insert
  - Find
    - Checks for membership
  - Remove



The Search ADT is sometimes called the "Set ADT"

Dictionary ADT: A Modest Few Uses

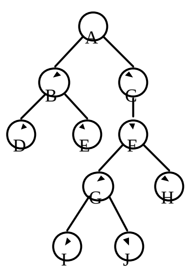
Naïve Implementations

	insert	find	delete
• Linked list			
• Unsorted array			
• Sorted array			

so close!

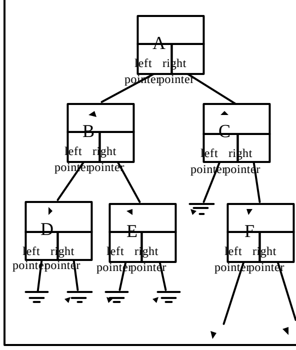
Binary Tree Data Structure

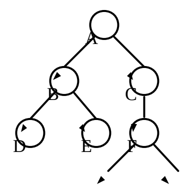
- A binary tree is:
  - a root
  - left subtree *(may be empty)*
  - right subtree *(may be empty)*
- Note the recursive definition!



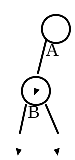
Is this tree complete?

Representation

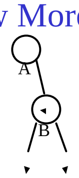




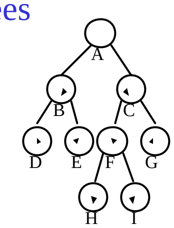
A Few More Trees



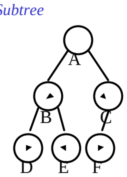
Left Subtree



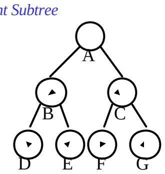
Right Subtree



Full Tree



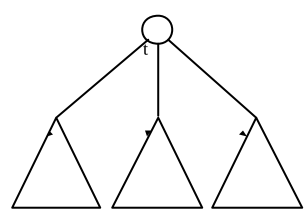
Complete Tree



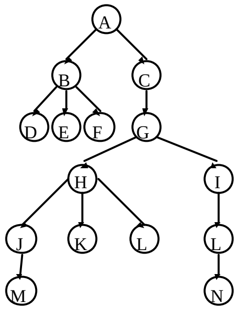
Perfect Tree

Recursive Tree Calculations

- Find the longest undirected path in a tree
- Think recursively!

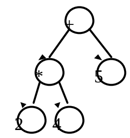


Recursive Tree Calculations Example



More Recursive Tree Calculations: Traversals

- A *traversal* is an order for visiting all the nodes of a tree
- Three types:
  - Pre-order
    - Root, left subtree, right subtree
  - In-order
    - Left subtree, root, right subtree
  - Post-order
    - Left subtree, right subtree, root



An expression tree

ADT Redux (last time today, I promise!)

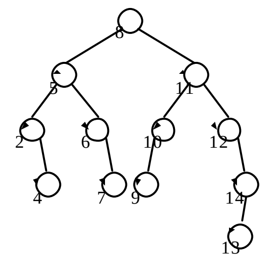
- Stack
  - Push
  - Pop
- Queue
  - Enqueue
  - Dequeue
- List
  - Insert
  - Remove
  - Find
- Priority Queue
  - Insert
  - DeleteMin

What's wrong with Lists?

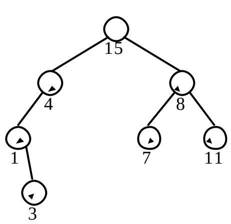
Remember decreaseKey?

Binary Search Tree Data Structure

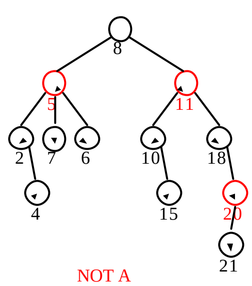
- Structure property
  - Each node has ≤ 2 children
- Order property
  - All keys in left subtree smaller than root's key
  - All keys in right subtree larger than root's key
  - Result:
    - Easy to find any given key



Example and Counter-Example

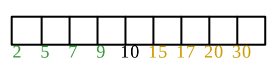


BINARY SEARCH TREE

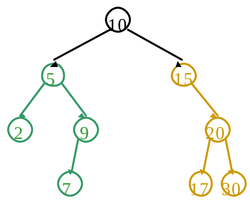


NOT A
BINARY SEARCH TREE

Why It's Called a "*Binary Search* Tree"



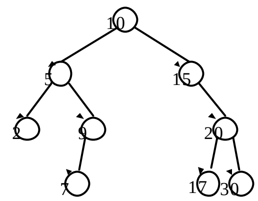
The Smooshed-Tree Principle



2	5	7	9	10	15	17	20	30

In order listing:

Finding a Node

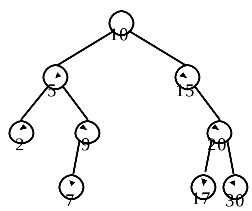



```
Node Find(Object key,
          Node root) {
  if (root == NULL)
    return NULL;

  if (key < root.key)
    return Find(key,
                root.left);
  else if (key > root.key)
    return Find(key,
                root.right);
  else
    return root;
}
```

Runtime:

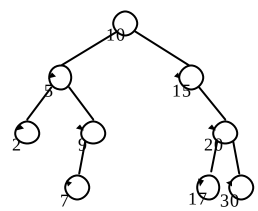
Iterative Find

```
Node Find(Object key,
          Node root) {

  while (root != NULL &&
         root.key != key) {
    if (key < root.key)
      root = root.left;
    else
      root = root.right;
  }

  return root;
}
```



Look familiar?

Insert

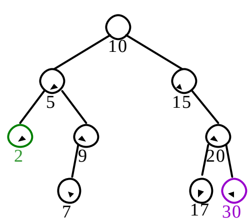



```
void Insert(Object key,
            Node root) {
  Node target = find(key,
                     root);
  // What should we do?
  if(target != NULL)


  target = new Node(key);
}
```

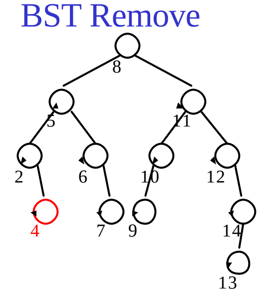
Runtime:

Bonus: FindMin/FindMax

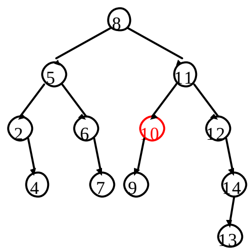
- Find minimum
- Find maximum



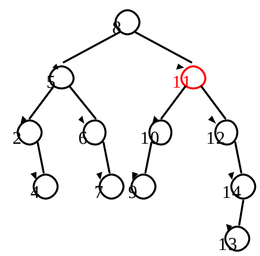
BST Remove



BST Remove (part deux)



BST Remove (part trois)



To Do

- Start Project 2
- Read chapter 4 in the book