CS 61BL Midterm 2 Review Session

Chris Gioia, Patrick Lutz, Ralph Arroyo, Sarah Kim

Algorithmic Analysis

Recall the definitions of:

1. f(n) is in O(g(n))
2. f(n) is in big Omega of g(n)
3. f(n) is in big Theta of g(n)

Algorithmic Analysis

Recall the definitions of:

1. f(n) is in O(g(n))

There exist M, N > 0 such that

for all n > N, f(n) < M*g(n)

Algorithmic Analysis

Recall the definitions of:

2. f(n) is in big Omega of g(n)

There exist M, N > 0 such that

for all n > N, f(n) > M*g(n)

Algorithmic Analysis

Recall the definitions of:

3. f(n) is in big Theta of g(n)

f(n) is in O(g(n)) and f(n) is in big Omega of g (n)

Algorithmic Analysis

Big-Oh notation just denotes a mathematical relationship between functions.

Suppose you say that the worst case run-time of searching in a balanced binary search tree is $O(\log(n))$.

Algorithmic Analysis

Big-Oh notation just denotes a mathematical relationship between functions.

Suppose you say that the worst case run-time of searching in a balanced binary search tree is O(log(n)).

Make sure to specify what n is!

Algorithmic Analysis

Give a tight bound for the run-time of foo(x, y) in terms of x and y. Assume x and y are positive.

```
public static int foo(int n,
    int m) {
    if (m == 0) {
        return bar(n - 1, n);
    }
    return foo(n, m - 1);
}
```

```
public static int bar(int n,
int m) {
    if (n == 0) {
        return 1;
    }
    return foo(n, 3*m);
}
```

Algorithmic Analysis

$O(x^2 + y)$

First foo is called y times in a row

Then bar(x-1,x) is called, resulting in 3*x more calls to foo

and so on…

Algorithmic Analysis

$O(x^2 + y)$

y calls to foo + 1 call to bar

3*x calls to foo + 1 call to bar

3*(x-1) calls to foo + 1 call to bar

.

.

.

3*(1) calls to foo + 1 call to bar

Algorithmic Analysis

$O(x^2 + y)$

$y + 1 + 3x + 1 + 3(x-1) + 1 + \ldots + 3(1) + 1 =$

$y + x + 1 + 3(x + (x-1) + \ldots + 1) =$

$y + x + 1 + 3(x(x-1)/2) =$

$y + x + 1 + (3/2)x^2 - (3/2)x =$

$(3/2)x^2 - (3/2)x + 1 + y$

Linked Lists

```
public class List {
    private ListNode
        myHead;
    public List() {
        myHead = null;
    }
    public List(ListNode node){
        myHead = node;
    }
    // Rotate method goes here
    // Nested ListNode class
}
```

Linked Lists

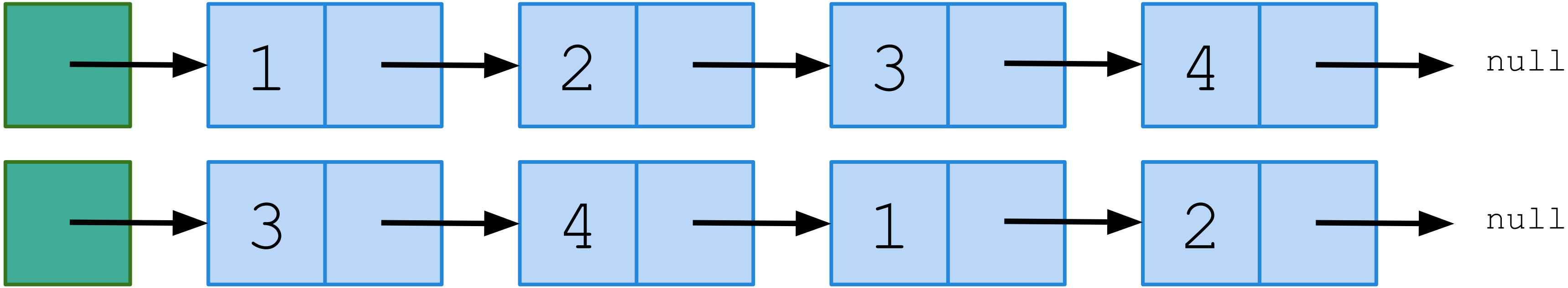
```
private static class ListNode {
    public Object myItem;
    public ListNode myNext;
    public ListNode(Object o) {
        myItem = o;
        myNext = null;
    }
    public ListNode(Object o, ListNode node) {
        this(o);
        myNext = node;
    }
}
```

Linked Lists - Rotate

Implement the `rotate` method destructively in the List class. Given an index, place all ListNodes before the index at the end of the list. Assume that index is not greater than the length of the list.

```
public void rotate(int index) {...}
```

Example below when rotate is called with index 2.

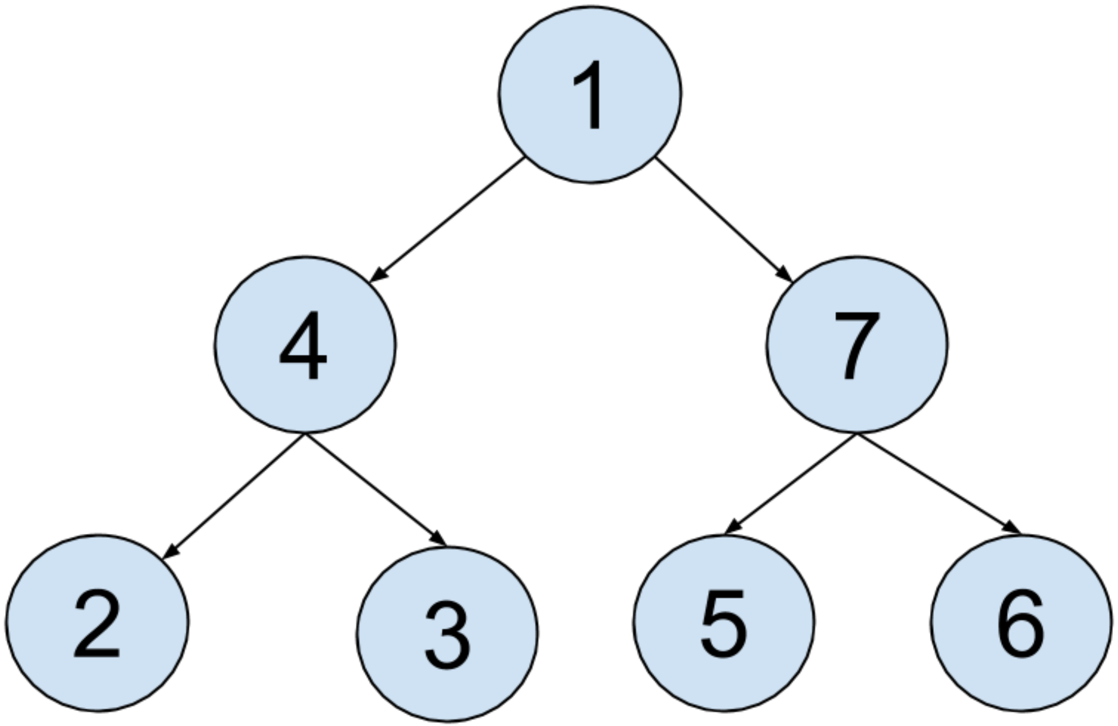


```java
// Flawed iterative solution; fix the errors!
public void rotate(int index) {
    if (index == 0) {
        return;
    }
    ListNode toBeRotated = this.myHead;
    ListNode toShiftUp = this.myHead;
    for (int i = 0; i < index; i++) {
        toShiftUp = toShiftUp.myNext;
    }
    this.myHead = toShiftUp;
    ListNode toAppend = this.myHead;
    while (toAppend.myNext != null) {
        toAppend = toAppend.myNext;
    }
    toAppend.myNext = toBeRotated;
}
```

```java
// Iterative
public void rotate(int index) {
    if (index == 0) {
        return;
    }
    ListNode toBeRotated = this.myHead;
    ListNode toShiftUp = this.myHead;
    for (int i = 0; i < index - 1; i++) {
        toShiftUp = toShiftUp.myNext;
    }
    this.myHead = toShiftUp.myNext;
    toShiftUp.myNext = null;
    ListNode toAppend = this.myHead;
    while (toAppend.myNext != null) {
        toAppend = toAppend.myNext;
    }
    toAppend.myNext = toBeRotated;
}
```

```java
// Recursive
public void rotate(int index) {
    if (index != 0) {
        return helper(this.myHead, this.myHead, index);
    }
}
public void helper(ListNode cur, ListNode head,
    int index) {
    if (index == 1) {
        helper(cur.myNext, head, index - 1);
        myHead = cur.myNext;
        cur.myNext = null;
    } else if (cur.myNext == null) {
        cur.myNext = head;
    } else {
        helper(cur.myNext, head, index - 1);
    }
}
```

Trees



Trees - Definitions

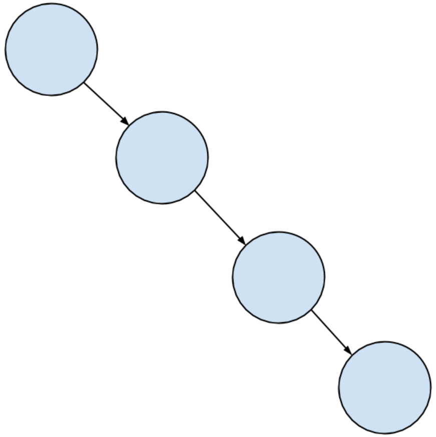
- Is a linked list with no cycles a type of tree?

Trees - Definitions

- Is a linked list with no cycles a type of tree? **Yes**

LinkedList's *next* node corresponds to a binary tree's "*myRight*"

(Provided LinkedList has no cycles)



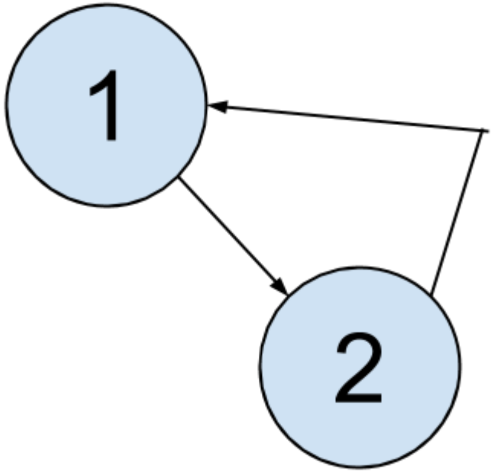
Trees - Definitions

What are the two constraints of a tree that this data structure does not ensure?

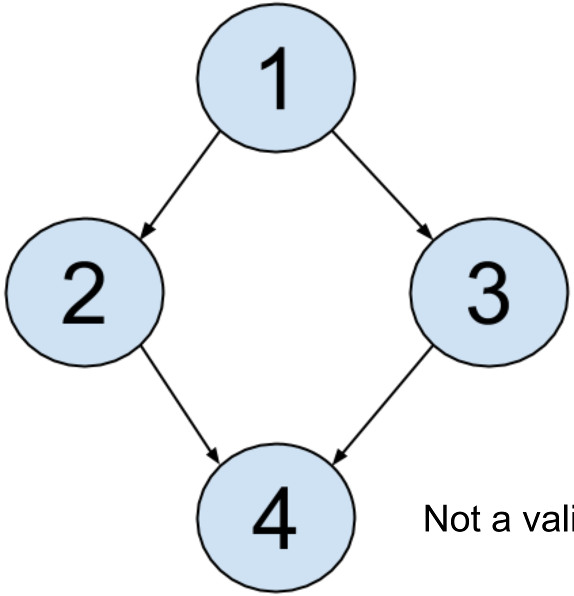
```
public class BinaryTree  {
    TreeNode root;

    public static class TreeNode {
        int item;
        TreeNode left;
        TreeNode right;

        public TreeNode(int item) {
            this.item = item;
        }
    }
}
```

Trees - Definitions

What are the two constraints of a tree that this data structure does not ensure?



Not a valid tree



Not a valid tree

Trees - Definitions

What are the two constraints of a tree that this data structure does not ensure?

1. No Cycles
2. Every node has at most one parent

```
public class BinaryTree  {
	TreeNode root;

	public static class TreeNode {
		int item;
		TreeNode left;
		TreeNode right;

		public TreeNode(int item) {
			this.item = item;
		}
	}
}
```

Trees - Definitions

`boolean isValidTree();`

<u>Description:</u>

Returns true if a tree satisfies the following:

1. no cycles
2. every node has at most one parent.

May add one helper method.

```
public class BinaryTree {
    TreeNode root;

    public static class TreeNode {
        int item;
        TreeNode left;
        TreeNode right;

        public TreeNode(int item) {
            this.item = item;
        }
    }
}
```

Trees - Definitions

```
boolean isValidTree();
```

<u>Description:</u>

Returns true if a tree satisfies the following:

1. no cycles
2. every node has at most one parent.

Hint: Use the helper method:

```
boolean static isValidTree
  (TreeNode node,
   HashSet<TreeNode> nodesSeen);
```

```
public class BinaryTree {
    TreeNode root;

    public static class TreeNode {
        int item;
        TreeNode left;
        TreeNode right;

        public TreeNode(int item) {
            this.item = item;
        }
    }
}
```

Trees - Definitions

DFS with HashSet

```
public boolean isValidTree() {
    HashSet<TreeNode> nodesSeen = new
HashSet<TreeNode>();
    if (root != null) {
        return isValidTree(root, nodesSeen);
    }
    return true;
}
```

```
public static boolean isValidTree
(TreeNode node, HashSet<TreeNode>
nodesSeen) {
    if (node != null) {
        if (nodesSeen.contains(node)) {
          return false;
        }
          nodesSeen.add(node);
          return isValidTree(node.left,
nodesSeen) && isValidTree(node.right,
nodesSeen);
    }
          return true;
}
```

Trees - Traversals

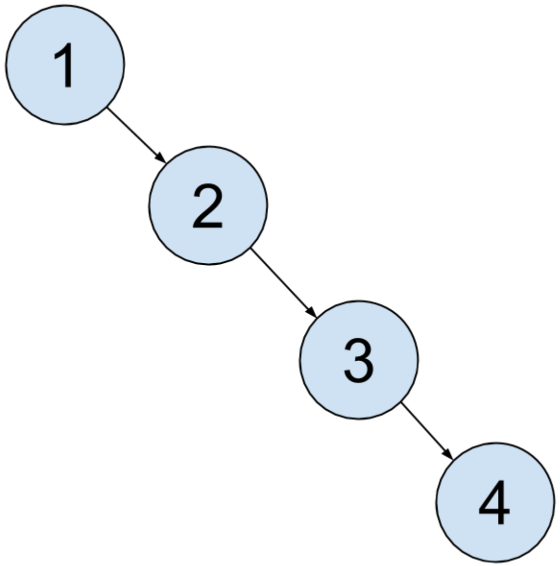
(a) Given 4 nodes labeled 1, 2, 3, and 4, draw a tree whose inorder and preorder traversal are equivalent:

(b) Given 4 nodes labeled 1, 2, 3, and 4, draw a tree whose inorder and postorder traversal are equivalent

(c) Given the preorder and postorder traversals of a tree are identical, what can I say about the tree?

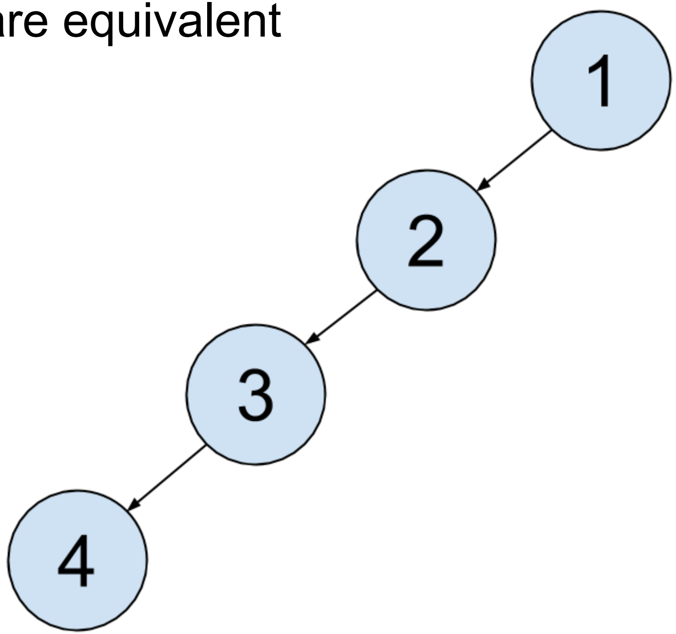
Trees - Traversals

(a) Given 4 nodes labeled 1, 2, 3, and 4, draw a tree whose inorder and preorder traversal are equivalent:



Trees - Traversals

(b) Given 4 nodes labeled 1, 2, 3, and 4, draw a tree whose inorder and postorder traversal are equivalent



Trees - Traversals

(c) Given the preorder and postorder traversals of a tree are identical, what can I say about the tree?

Say I have a preorder traversal 1, 2, … , N
    and a postorder traversal 1, 2, … , N

By postorder, root node is 1. But by postorder, root node is N (why?)

Trees - Traversals

(c) Given the preorder and postorder traversals of a tree are identical, what can I say about the tree?

Say I have a preorder traversal 1, 2, … , N

and a postorder traversal 1, 2, … , N

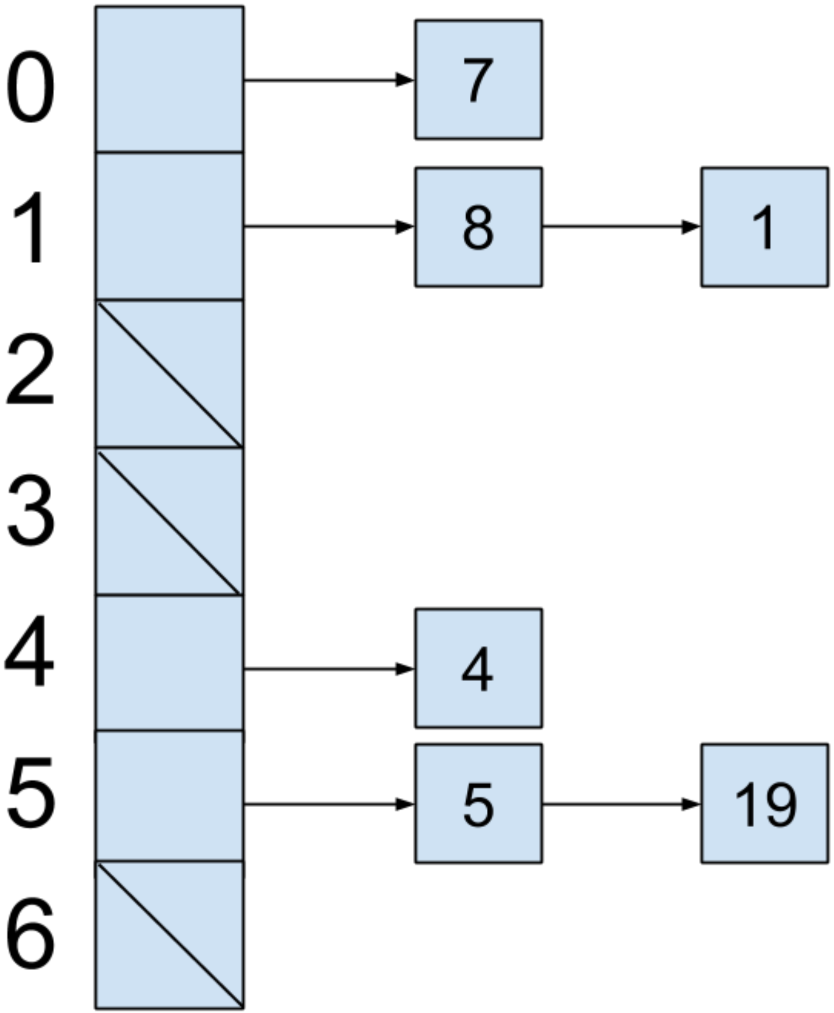
By postorder, root node is 1. But by postorder, root node is N (why?)

=> N = 1

=> **tree has at most one node**

Hash Tables

Given a hash table of size 7, show the contents of the hash table after inserting the elements {8, 5, 4, 19, 7, 1}. The hash function for the elements is given as `h(x) = x`. Assume that the hash table uses chaining to address collisions.


0
1
2
3
4
5
6
7
8
1
4
5
19

Hashing - Important methods

- If we invoke the `contains(E key)` method in Java's implementation of `HashSet<E>`, the method will invoke `key.hashCode` and may call `key.equals` if necessary.
  - What is the purpose of the call to `.hashCode`?
    - `.hashCode` determines which bucket the hash set will search through to find `key`
  - When will `.equals` be used?
    - `.equals` finds the equivalent object in the collection stored in the bucket, if the bucket contains the collection.

Hashing

- What are some properties of a good hash function? Try remembering this as “DUQ”
  - Deterministic
    - repeated calls returns the same thing
  - Uniform
    - keys spread out evenly across buckets
  - Quick
    - should try to be close to constant time

Make a hash function!

A binary string is a subset of strings that contain only the characters ‘0’ and ‘1’. For example, “01100001”, “0” and “100” are binary strings.

Edit: The solution provides a one-to-one mapping from binary strings to non-negative ints iff there are no leading zeroes in the string! Good job with catching this inconsistency in the review =)

Create a hash function for binary strings, such that each possible input maps to a unique positive integer.

Can you extend this to ternary strings?

Solution

`s[0]*b^(n-1) + s[1]*b^(n-2) + s[2]*b^2 + … + s[n-1]*b^0`

- `b` is the number of characters
- `s[i]` is the `i`-th character of the string, mapped to a number between 0 and (b - 1)
- `n` is the length of the string

Design an algorithm

```
// Input: Two non-null int arrays, a and b, both of length >= 1
// Output: Return true if there a and b share any common numbers.
//         Otherwise, return false.
// Note: Your algorithm must be as efficient as possible.
public boolean hasCommonElements(int[] a, int[] b){









}
```

Design an algorithm! Solution

```
// Input: Two non-null int arrays, a and b, both of length >= 1
// Output: Return true if there a and b share any common numbers.
//         Otherwise, return false.
// Note: Your algorithm must be as efficient as possible.
public boolean hasCommonElements(int[] a, int[] b){
    HashSet<Integer> nums = new HashSet<Integer>();

    for(int x : a){
        nums.put(x);
    }

    for(int y : b){
        if (nums.contains(y))
            return true;
    }

    return false;
}
```

Algorithmic Analysis II

Suppose $f(n)$ is in $O(g(n))$

True or False:

1. $f(n)^2$ is in $O(g(n)^2)$
2. $2^{f(n)}$ is in $O(2^{g(n)})$
3. $g(n)$ is in big Omega of $f(n)$
4. $f(n)$ is in big Theta of $g(n)$

Algorithmic Analysis II

Suppose $f(n)$ is in $O(g(n))$

True or False:

1. $f(n)^2$ is in $O(g(n)^2)$ True
2. $2^{f(n)}$ is in $O(2^{g(n)})$
3. $g(n)$ is in big Omega of $f(n)$
4. $f(n)$ is in big Theta of $g(n)$

Algorithmic Analysis II

Suppose f(n) is in O(g(n))

True or False:

1. $f(n)^2$ is in $O(g(n)^2)$ True

$f(n) <= M*g(n)$ for all $n > N$ implies

$f(n)^2 <= M^2 g(n)^2$ for all $n > N$

Algorithmic Analysis II

Suppose f(n) is in O(g(n))

True or False:

1. $f(n)^2$ is in $O(g(n)^2)$ True
2. $2^{f(n)}$ is in $O(2^{g(n)})$ False
3. g(n) is in big Omega of f(n)
4. f(n) is in big Theta of g(n)

Algorithmic Analysis II

Suppose f(n) is in O(g(n))

True or False:

2. $2^{f(n)}$ is in $O(2^{g(n)})$ False

Suppose f(n) = 2*n and g(n) = n

Then for any M, for all n>M,

$2^{f(n)} = 2^{2n} = 2^n*2^n > n*2^n >= M*2^n = M*g(n)$

Algorithmic Analysis II

Suppose $f(n)$ is in $O(g(n))$

True or False:

1. $f(n)^2$ is in $O(g(n)^2)$ True
2. $2^{f(n)}$ is in $O(2^{g(n)})$ False
3. $g(n)$ is in big Omega of $f(n)$ True
4. $f(n)$ is in big Theta of $g(n)$

Algorithmic Analysis II

Suppose f(n) is in O(g(n))

True or False:

3. g(n) is in big Omega of f(n) True

f(n) <= M*g(n) for all n > N implies

g(n) >= (1/M)*f(n) for all n > N

Algorithmic Analysis II

Suppose f(n) is in O(g(n))

True or False:

1. $f(n)^2$ is in $O(g(n)^2)$ True
2. $2^{f(n)}$ is in $O(2^{g(n)})$ False
3. g(n) is in big Omega of g(n) True
4. f(n) is in big Theta of g(n) False

Algorithmic Analysis II

Suppose f(n) is in O(g(n))

True or False:

4. f(n) is in big Theta of g(n) False

Suppose f(n) = log(n) and g(n) = n

Then f(n) is in O(g(n)) but f(n) is not in big Omega of g(n)

Trees - Binary Search Trees

(a) Say we have a Binary Search Trees with nodes whose items are Integers, and Integers take the natural ordering.

Recreate the BST produced by the preorder traversal: `4, 2, 1, 3, 5, 6`

Trees - Binary Search Trees

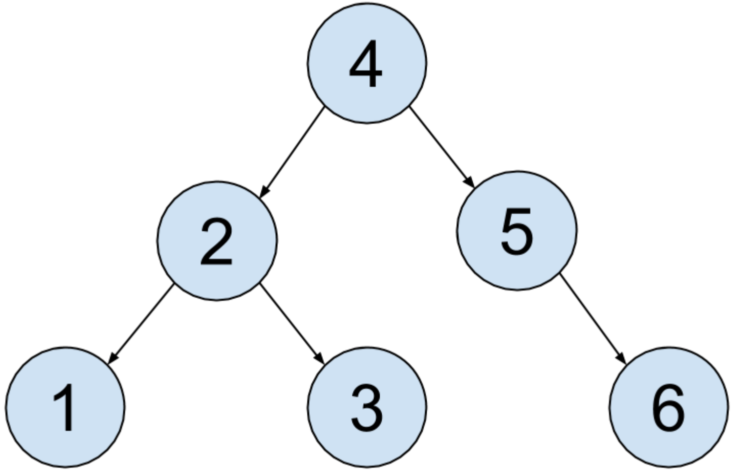
(a) Say we have a Binary Search Trees with nodes whose items are Integers, and Integers take the natural ordering.
Recreate the BST produced by the preorder traversal: `4, 2, 1, 3, 5, 6`

Hint: What is the inorder traversal?

Trees - Binary Search Trees

(a) Say we have a Binary Search Trees with nodes whose items are Integers, and Integers take the natural ordering.

Recreate the BST produced by the preorder traversal: `4, 2, 1, 3, 5, 6`



Trees - Binary Search Trees

(b) Complete the following function definition: Assume that we are working with BinaryTrees of Integers and the nodes have unique values.

The maximum integer: Integer.MIN_VALUE   The minimum integer: Integer.MAX_VALUE

```
/** Returns true iff binary tree T is a binary search tree. */
boolean isSearchTree(){
    // implement code here
}

boolean isSearchTree(TreeNode t, int min, int max) {
    // implement code here
}
```

Trees - Binary Search Trees

(b)

```
boolean isSearchTree(){
 return isSearchTree(root, Integer.MIN_VALUE, Integer.MAX_VALUE)
}

boolean isSearchTree(TreeNode t, int min, int max) {
    if (t == null) {
        return true;
    } else if (t.item < min || t.item > max) {
        return false;
     } else {
        return isSearchTree(t.left, min, t.item) &&
               isSearchTree(t.right, t.item, max);
    }
}
```

Trees - TreeMaps

(a) What two things should the key of a TreeMap satisfy?

(b) Anything that the value of a TreeMap must satisfy?

Trees - TreeMaps

(a) What two things should the key of a TreeMap satisfy?

- Implement comparable
- Immutable

(b) Anything that the value of a TreeMap must satisfy?

Trees - TreeMaps

(a) What two things should the key of a TreeMap satisfy?

- Implement comparable
- Immutable

(b) Anything that the value of a TreeMap must satisfy?

- Nothing