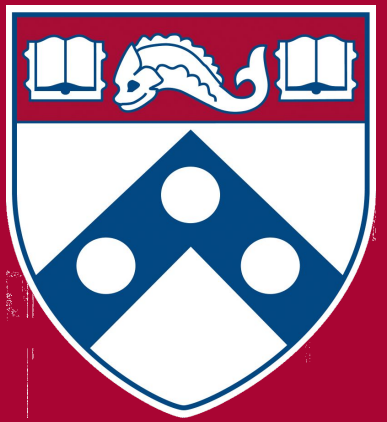
Recitation 5: Binary Search Trees, Heaps, & Priority Queues

CIT 5940

February 23, 2023



Attendance

ENTER CODE:

090909

Announcements

1. HW3 due **March 1st @ 11:59PM ET (Extended)**

2. Look out for **HW2 Regrades** in the future
   a. Grades should come out soon!

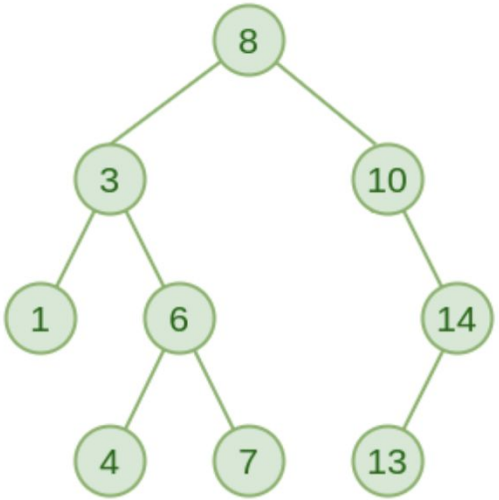
3. Recitation Assignment due **2/24 @ 11:45 PM ET**

4. HW4 will be released next week, but **due in 3 weeks (to account for Spring Break)**

Binary Search Trees (BST)

BST Properties

- The left subtree of a node contains only nodes with keys **lesser than the node's key.**
- The right subtree of a node contains only nodes with keys **greater than the node's key.**
- The left and right subtree each must also **be a binary search tree.**



BST Code Example (Easy)

Q: Write a program to search for a given key in a BST

```java
// Recursive function to search in a given BST
public static void search(Node root, int key, Node parent)
{
    // if the key is not present in the key
    if (root == null)
    {
        System.out.println("Key not found");
        return;
    }

    // if the key is found
    if (root.data == key)
    {
        if (parent == null) {
            System.out.println("The node with key " + key + " is root node");
        }

        else if (key < parent.data)
        {
            System.out.println("The given key is the left node of the node " +
                        "with key " + parent.data);
        }
        else {
            System.out.println("The given key is the right node of the node " +
                        "with key " + parent.data);
        }

        return;
    }

    // if the given key is less than the root node, recur for the left subtree;
    // otherwise, recur for the right subtree

    if (key < root.data) {
        search(root.left, key, root);
    }
    else {
        search(root.right, key, root);
    }
}
```

BST Code Example (Medium)

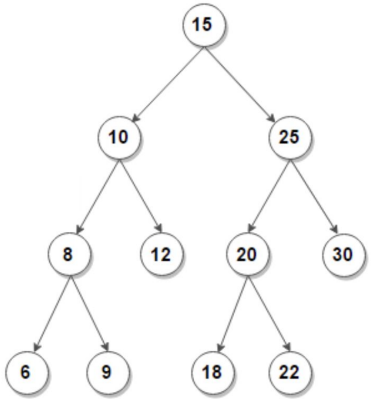
Q: Write a program to determine whether a given binary tree is a BST

```java
// Function to determine whether a given binary tree is a BST by keeping a
// valid range (starting from [-INFINITY, INFINITY]) and keep shrinking
// it down for each node as we go down recursively
public static boolean isBST(Node node, int minKey, int maxKey)
{
    // base case
    if (node == null) {
        return true;
    }

    // if the node's value falls outside the valid range
    if (node.data < minKey || node.data > maxKey) {
        return false;
    }

    // recursively check left and right subtrees with an updated range
    return isBST(node.left, minKey, node.data) &&
        isBST(node.right, node.data, maxKey);
}

// Function to determine whether a given binary tree is a BST
public static void isBST(Node root)
{
    if (isBST(root, Integer.MIN_VALUE, Integer.MAX_VALUE)) {
        System.out.println("The tree is a BST.");
    }
    else {
        System.out.println("The tree is not a BST!");
    }
}
```

BST Code Example (Hard)

Q: Write a program to find a pair with the given sum in a BST

Example: If the given sum is 14, the pair is (8, 6)



```java
    // Function to find a pair with a given sum in the BST
    public static boolean findPair(Node root, int target, Set<Integer> set)
    {
        // base case
        if (root == null) {
            return false;
        }

        // return true if pair is found in the left subtree; otherwise, continue
        // processing the node
        if (findPair(root.left, target, set)) {
            return true;
        }

        // if a pair is formed with the current node, print the pair and return
true
        if (set.contains(target - root.data))
        {
            System.out.println("Pair found (" + (target - root.data) + ", "
                                    + root.data + ")");
            return true;
        }

        // insert the current node's value into the set
        else {
            set.add(root.data);
        }

        // search in the right subtree
        return findPair(root.right, target, set);
    }
```

The idea is to traverse the tree in an **inorder fashion** and insert every node's value into a set. Also check if, for any node, the difference between the given sum and node's value is found in the set, then the pair with the given sum exists in the tree.

Heaps and Priority Queues

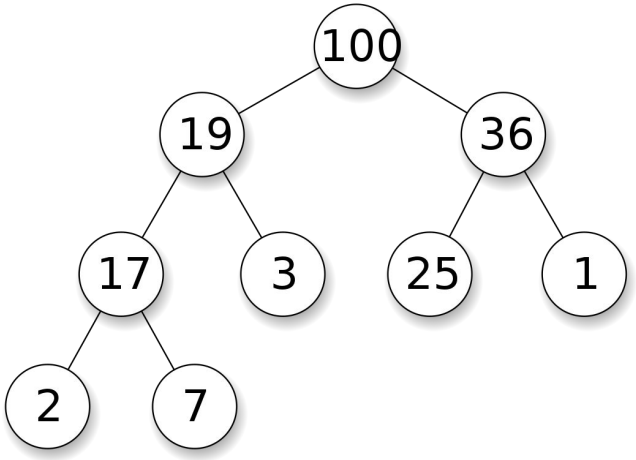
Binary Heap

Definition: a binary heap is a complete binary tree - that is, all levels of the tree, except the last level are fully filled. If the last level of the tree is not complete, the nodes of that level are filled from left to right.

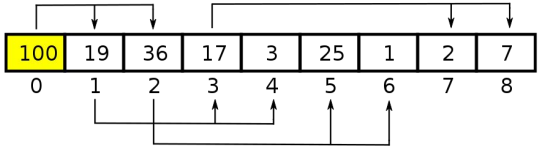
Max Heap: every node stores a value that is greater than or equal to the value of either of its children

Min Heap: every node stores a value that is less than or equal to that of its children

Heap Invariant



Array representation



- Parent(r) =⌊(r −1)/2⌋ if r ≠ 0
- Left child(r) = 2r + 1 if 2r + 1 < n
- Right child(r) =2r + 2 if 2r + 2 < n
- Left sibling(r) =r − 1 if r is even and r ≠ 0
- Right sibling(r) = r + 1 if r is odd and r+1< n

Heap Functions

- **Heapify (swim, sink) - O(n)**
  - Appropriately sorts the heap
- **BuildTree - O(nlogn)**
  - Builds tree from scratch
- **Insert - O(logn)**
  - Inserts at the bottom level of the tree
- **Remove - O(logn)**
  - Removes the root of the tree

Heap Sort

1. With given array, build a unsorted heap
2. Heapify tree to get a max heap
3. Swap root with lowest node, remove the lowest node (this node is sorted)
4. Repeat to step 2

Priority Queues

- Queue Data structure that is implemented using heaps
- Ordered by priority rather than position
  - Natural ordering
  - Custom Comparator
- Can be used to define Min/MaxHeap
  - Depending on the priority, items may be dequeued in descending order (MaxHeap) or ascending order (MinHeap)

Priority Queues/Binary Heap Example

list = {4, 7, 2, 10, 3, 8, 5, 1}

MinHeap

```
PriorityQueue<Integer> pq =
        new PriorityQueue<>()
pq.addAll(list)
System.out.println(pq)
while (!pq.isEmpty()) { System.out.println(pq.poll());}
```

pq = [1, 3, 2, 7, 4, 8, 5, 10]
prints: 1, 2, 3, 4, 5, 7, 8, 10

MaxHeap

```
PriorityQueue<Integer> pq = new
        PriorityQueue<>(Comparator.reverseOrder())
pq.addAll(list)
System.out.println(pq)
while (!pq.isEmpty()) { System.out.println(pq.poll());}
```

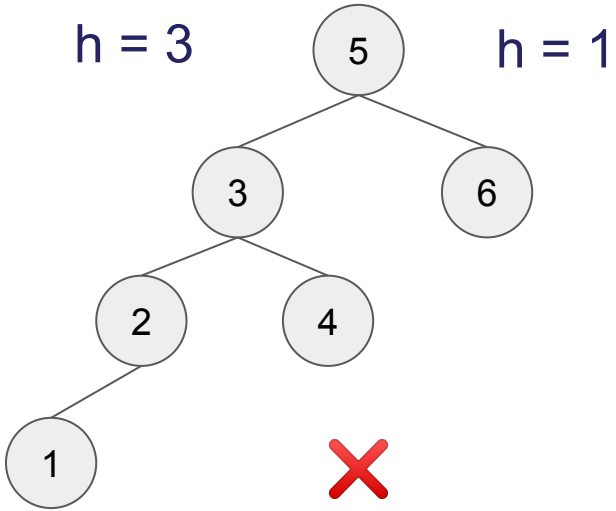
pq = [10, 7, 8, 4, 3, 2, 5, 1]
prints: 10, 8, 7, 5, 4, 3, 2, 1

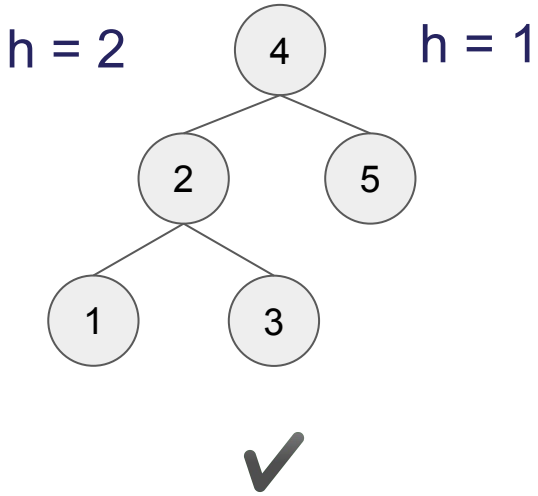
Recitation Activity

Problem 1: Creating a Balanced BST

What makes a balanced BST?

Formal Definition: The absolute difference between heights of left and right subtrees at any node should be less than 1

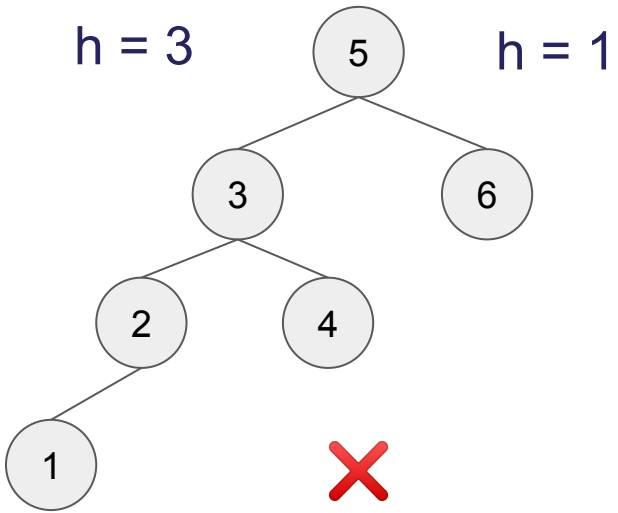




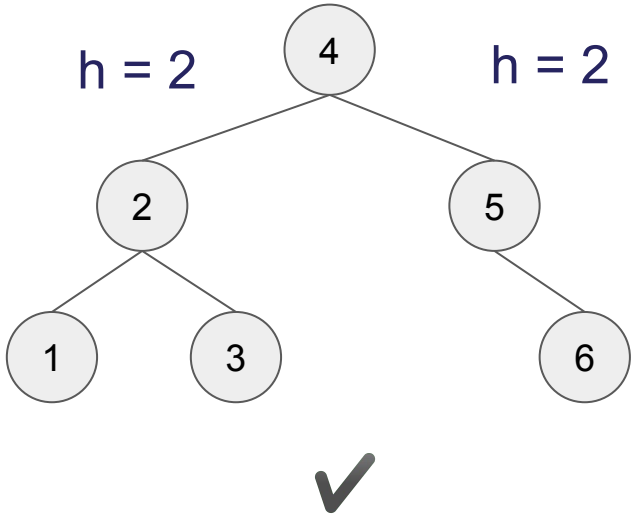
Problem 1: Creating a Balanced BST

Can we make something unbalanced into something balanced?

1, 2, 3, 4, **5**, 6



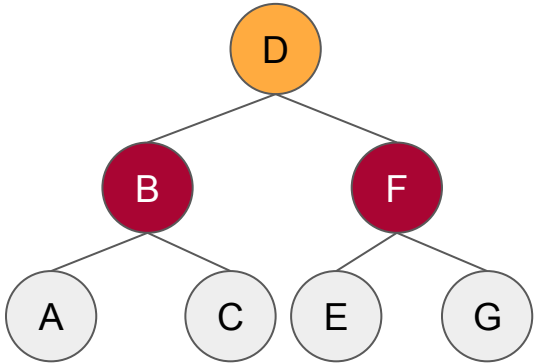
1, 2, 3, **4**, 5, 6



Problem 1: Creating a Balanced BST

Given a list of items, turn it into a balanced BST

A, B, C, D, E, F, G



Hint: Pay attention to what becomes the root at every level!

Problem 2: Fun with Comparators and Records

You've been given a Patient **record**:

```
public record Patient(String name, int pain, boolean preexisting)
        implements Comparable<Patient> {
        …
 }
```

Records are a special type of class that aggregates the canonical methods/fields of class, such as **getters/setters**, and **equals/hashcode/toString**.
For instance, printing out a *new Patient("Voravich", 10, true)* gets you:

Patient[name=Voravich, pain=10, preexisting=true]

Problem 2: Fun with Comparators and Records

- Patient record implements **Comparable, allowing it to be ordered in a PriorityQueue properly**

- You just need to fill out the compareTo() method with the proper conditions:
  - Descending order of pain
  - If there is a tie, order **those with preexisting conditions** ahead of **those that don't have preexisting conditions**
    - That implies: If both don't have preexisting conditions or both do, order doesn't matter

Attendance

ENTER CODE:

090909