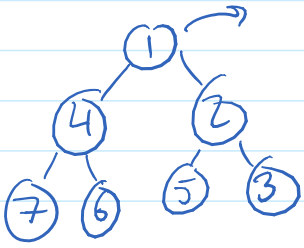
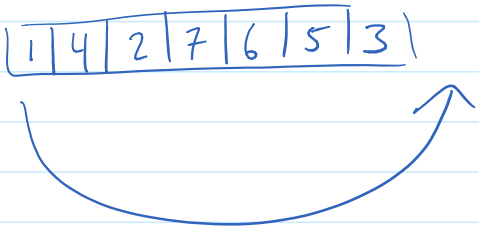
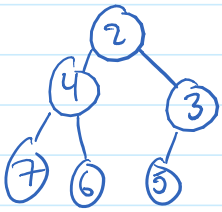
Sorting & Searching

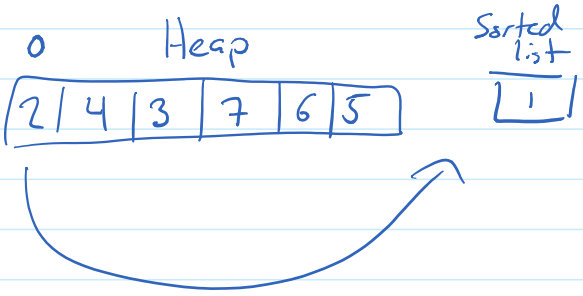
Thursday, November 5, 2015 11:36 AM

Recall Heapsort









Sorting takes $n \lg n$

reversing takes n

Sorting algorithms so far

	Insert	Quick	Merge	Heap
best	n	$n \lg n$	$n \lg n$	$n \lg n$
avg	n^2	$n \lg n$	$n \lg n$	$n \lg n$
worst	n^2	n^2	$n \lg n$	$n \lg n$

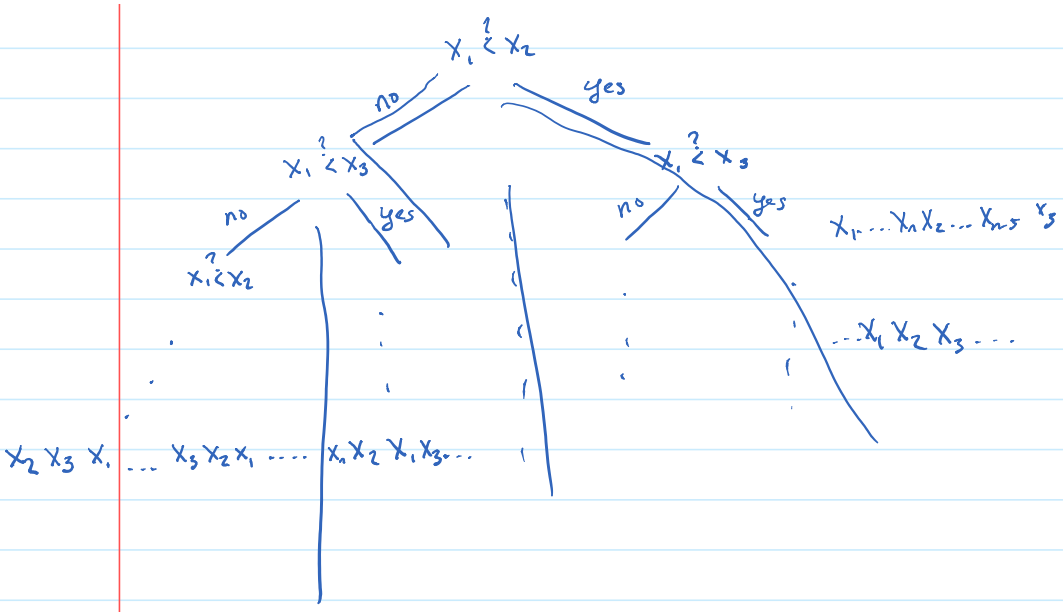
Comparison based sorts

Input: $x_1, x_2, x_3, \ldots, x_n$

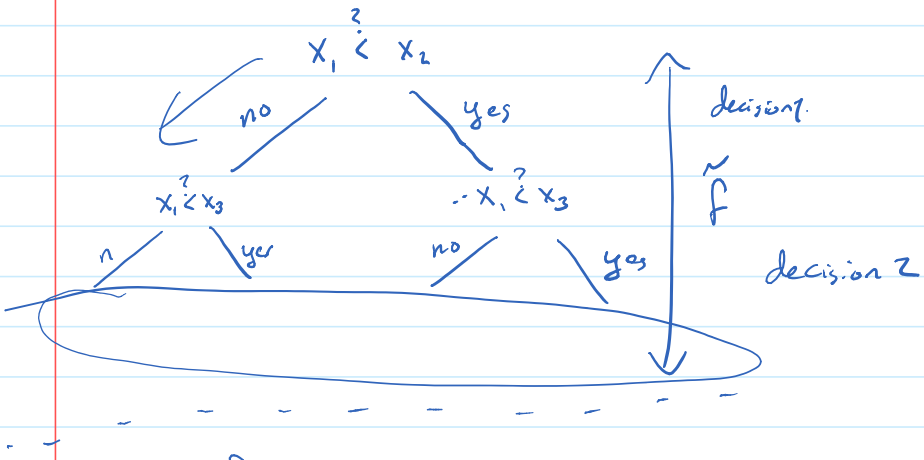
$x_1\ x_2\ x_3 \ldots x_n$

$x_1 \overset{?}{<} x_2$

no / yes



Minimum: $n!$ leaves



$\#$ of leaves $= 2^f$

$\#$ of leaves $\doteq \Omega(n!)$

$2^f \geq n!$

$\lg 2^f \geq \lg n!$

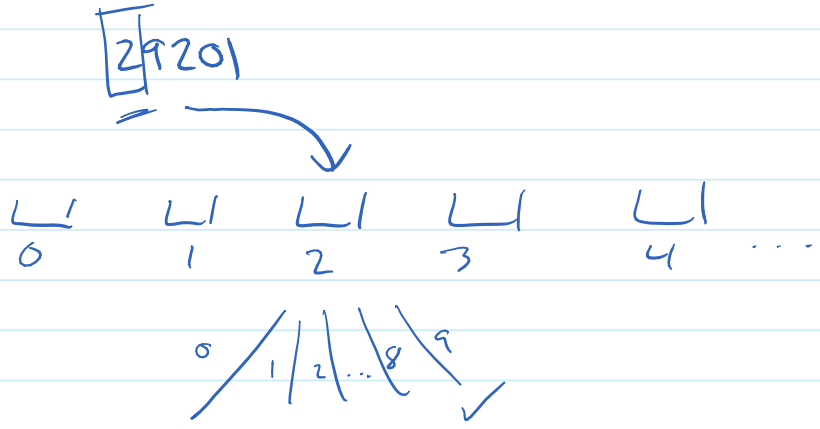
$f \geq \ln n!$ Stirling's Approx

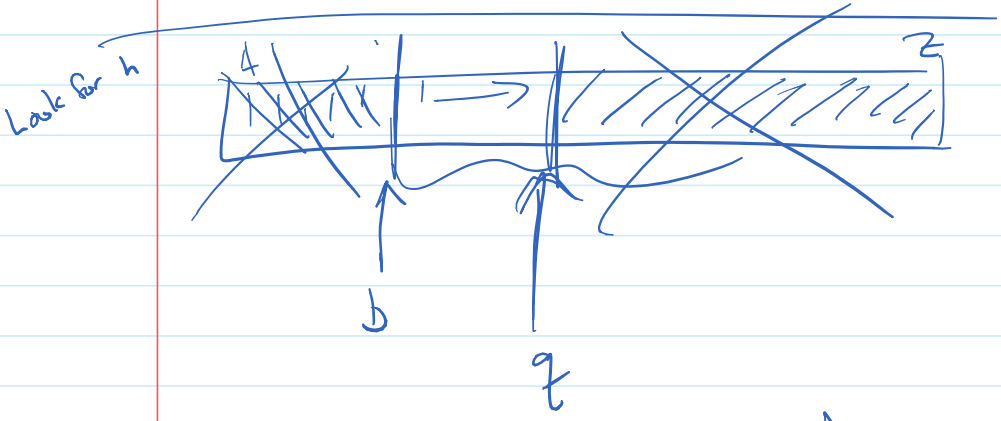
$f \geq n \ln n - n + \Theta(n)$

$f = \Omega(n \lg n)$ Avg case analysis (Comparison based sort)

best insert n ← not quite a sort

Radix Sort (Bucket sort)



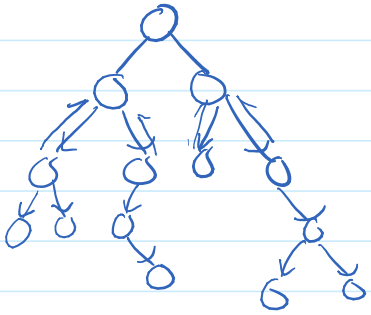


binary search ($\Theta(\lg n)$)

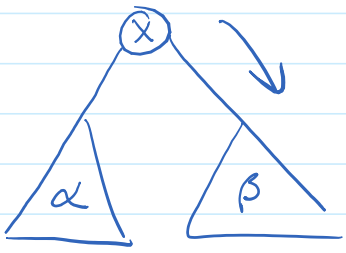
Binary Search Tree

Invariant

Binary search tree



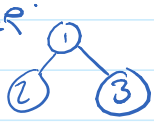
Invariant



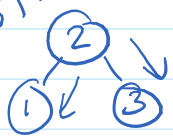
$\forall y \in \alpha \quad y \leq X$

$\forall z \in \beta \quad z > X$

Heap:

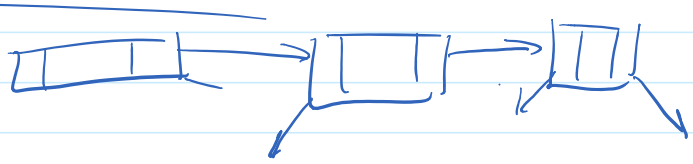


BST:



Binary Search Trees

- size
- insert
- lookup

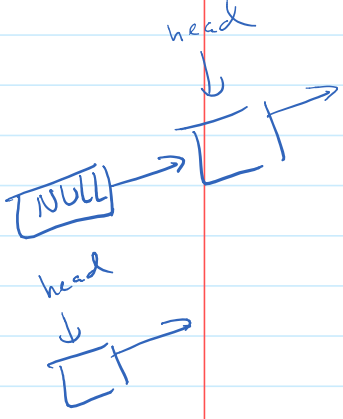


```
class BST<E extends Comparable<E>> {

    private class Node<E> //implements Com... {
        public E element;
        public Node<E> left;
        public Node<E> right;
    }
    private Node<E> root;
    private int size;

    public int size() {
        return size;
    }

    public void insert(E elem) {
        // if (root == null)
        if (size == 0) {
            Node<E> newRoot = new Node<E>();
            newRoot.element = elem;
```



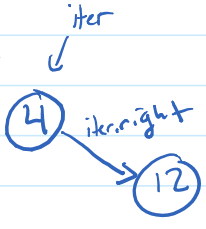
```
        Node<E> newRoot = new Node<E>();
        newRoot.element = elem;
        newRoot.left = null;
        newRoot.right = null;
        root = newRoot;
        size++;
        return;
    }
    Node<E> iter = root;
    while (true) {
        if (elem.compareTo(iter.element) < 0) {
            if (iter.left == null) {
                Node<E> newNode = new Node<E>();
                newNode.element = elem;
                iter.left = newNode;
                return;
            }
            iter = iter.left;
        } else {
            // Same for right (as above)
        }
```

list [4, 12, 16, 8, 3, 31, 9]

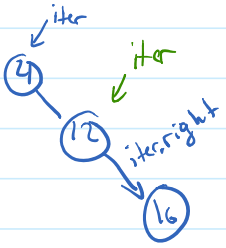
```
BST<Integer> tree = new ...

for (int i : list) {
    tree.insert(i);
}
```

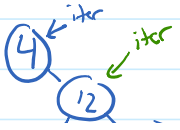
step 2:



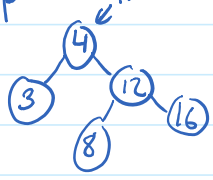
step 3:



step 4:

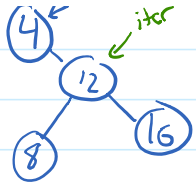


step 5:

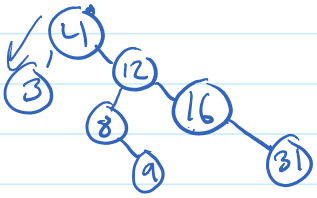


step 6-7:

step 4:



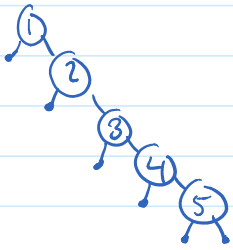
step 6-7:



lookup(3) in 2 comparisons

lookup(40) in 4 comparisons

[1, 2, 3, 4, 5]



[5, 4, 3, 2, 1]