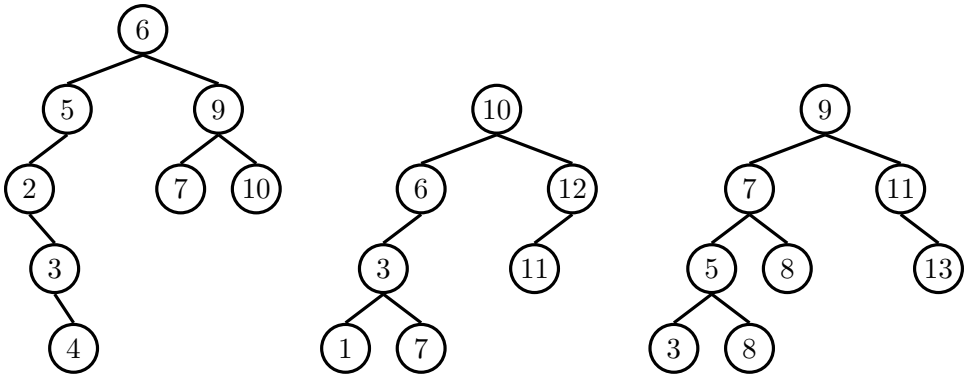
CSM 61B
Fall 2017

Balanced Search

Mentoring 11: November 6, 2017

1 Unbalanced Trees

1.1 For each of the following binary trees, determine if the height of the tree is the same as the height of the optimal tree with the same elements.



1. Not balanced
2. Not balanced
3. Balanced

Intuitively, if we can "stuff" all the elements in the lowest level into a higher level, then we can reduce the height of the tree and determine that the height of the tree is not the same as the height of the optimal binary tree.

1.2 Provide tight asymptotic runtime bounds in terms of N, the number of nodes in the tree, for the following operations and data structures.

Operations	Binary Search	Balanced Search
`boolean contains(E e);`	$\Omega(1)$ $O(N)$	$\Omega(1)$ $O(\log N)$
`boolean add(E e);`	$\Omega(1)$ $O(N)$	$\Theta(\log N)$

For `boolean contains(E e)`, the best case is always $\Theta(1)$ for both binary search trees and balanced search trees. This is the case if the node we are looking for is at the root of the tree. The worst case for binary search trees is if the tree is completely unbalanced (a spindly tree), and the node we are looking for is a leaf, yielding a time complexity of $\Theta(N)$ because we have to visit every node.

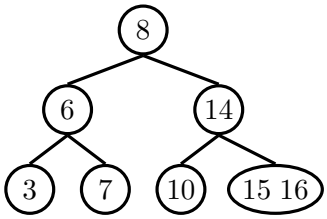
However, for a balanced search tree, we know that the height of the tree will never exceed $\Theta(\log N)$. Thus, even if the node we are looking for is a leaf, we will never have to search more than $\Theta(\log N)$ nodes.

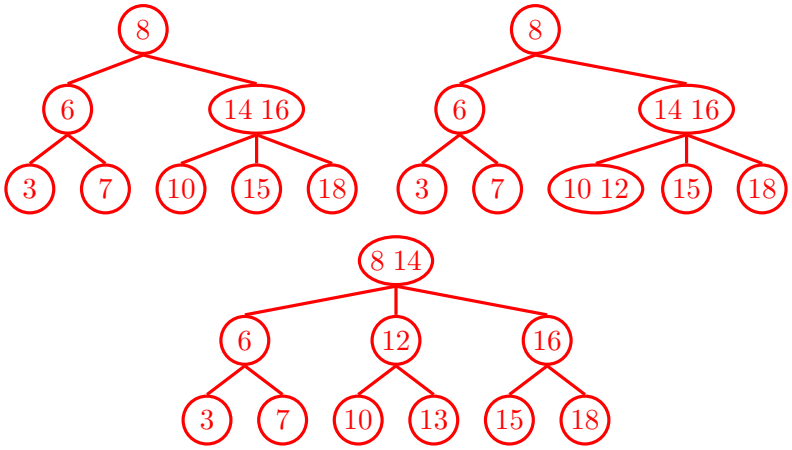
For `boolean add(E e)`, we always have to insert nodes as a a leaf. For a binary search tree, imagine a right-leaning spindly tree (where all of the root's children are in its right subtree). To insert a node in the root's left subtree, this would simply be a constant time operation because there are no nodes in the left subtree, so we would just need to set the root's left child to the node we are inserting. This is the best case for binary search trees, so the runtime is $\Theta(1)$ in this case. However, to insert a node in the root's right subtree, we would have to move all the way down the spindly tree, passing every node, to reach the leaf node to insert the new node. This is the worst case for binary search trees, so the runtime is $\Theta(N)$.

For balanced binary search trees, the height will always be $\Theta(\log N)$ nodes. This means that every insertion will always have to visit $\Theta(\log N)$ nodes to reach a leaf. Thus, the runtime is always in $\Theta(\log N)$.

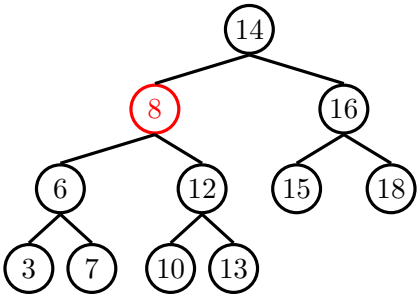
2 2-3-Forever

2.1 Draw the 2-3 tree would look like after inserting 18, 12, and 13.



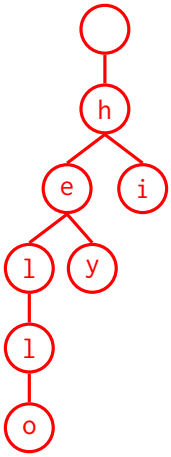


2.2 Now, convert the resulting 2-3 tree to a left-leaning red-black tree.



3 New World Order

3.1 Draw the trie that results from inserting `"hi"`, `"hello"`, and `"hey"`.



3.2 Given a list of words (possibly repeated), devise a strategy to efficiently return a list of all the words that start with a given prefix.

Put all the names into a trie, lookup the prefix in the trie, and iterate across all the children rooted at that node.

3.3 Given a dictionary of words, describe a procedure for checking if a new word can be created out of the concatenation of two words in the dictionary. For example, if our dictionary contains the words, `"news"`, `"paper"`, `"new"`, and `"ape"`, we should be able to discover the new word, `"newspaper"`.

We can put all of the words in the dictionary into a trie. Then we can check a prefix of the word and see if the remainder of the word is in the trie as well.

4 Sorting Mechanics *Extra Practice*

4.1 Each column below gives the contents of a list at some step during sorting. Match each column with its corresponding algorithm.

· Merge sort · Quicksort · Heap sort · LSD radix sort · MSD radix sort

For quicksort, choose the topmost element as the pivot. Use the recursive (top-down) implementation of merge sort.

	Start	A	B	C	D	E	Sorted
1	4873	1876	1874	1626	9573	2212	1626
2	1874	1874	1626	1874	7121	8917	1874
3	8917	2212	1876	1876	9132	7121	1876
4	1626	1626	1897	4873	6973	1626	1897
5	4982	3492	2212	4982	4982	9132	2212
6	9132	1897	3492	8917	8917	6152	3492
7	9573	4873	4873	9132	6152	4873	4873
8	1876	9573	4982	9573	1876	9573	4982
9	6973	6973	6973	1897	1626	6973	6152
10	1897	9132	6152	3492	1897	1874	6973
11	9587	9587	7121	6973	1874	1876	7121
12	3492	4982	8917	9587	3492	9877	8917
13	9877	9877	9132	2212	4873	4982	9132
14	2212	8917	9573	6152	2212	9587	9573
15	6152	6152	9587	7121	9587	3492	9587
16	7121	7121	9877	9877	9877	1897	9877

From left to right: unsorted list, quicksort, MSD radix sort, merge sort, heap sort, LSD radix sort, completely sorted.

MSD Look at the left-most digits. They should be sorted. Mark this immediately as MSD.

LSD One of the digits should be sorted. Start by looking at the right most digit of the remaining sorts. Then check the second from right digit of the remaining sorts and so on. As soon as you find one in which at

least something is sorted, mark that as LSD.

Heap Max-oriented heap so check that the bottom is in sorted order and that the top element is the next max element.

Merge Realize that the first pass of merge sort fixes items in groups of 2. Identify the passes and look for sorted runs.

Quick Run quicksort using the pivot strategy outlined above. Look for partitions and check that 4873 is in its correct final position.