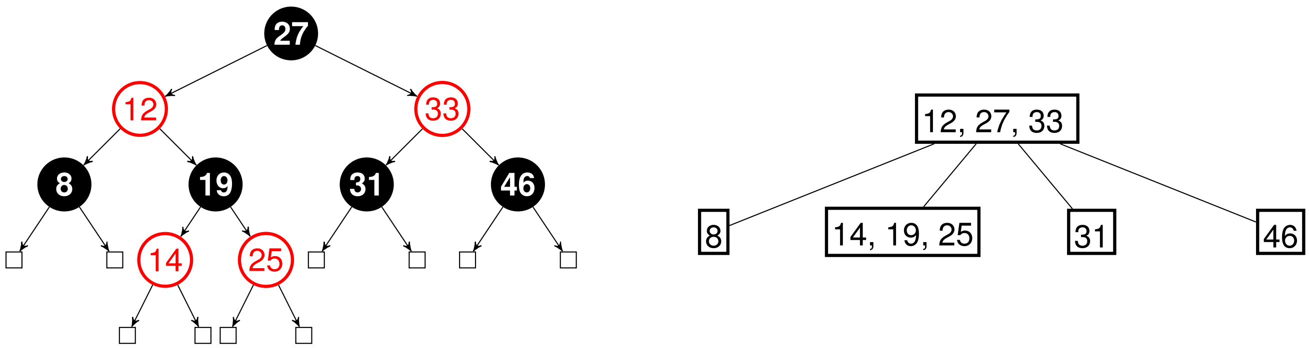
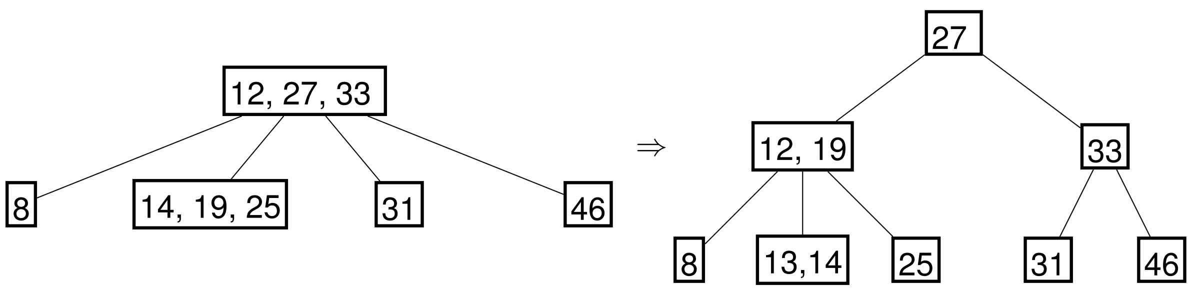
1 Balanced Search Trees

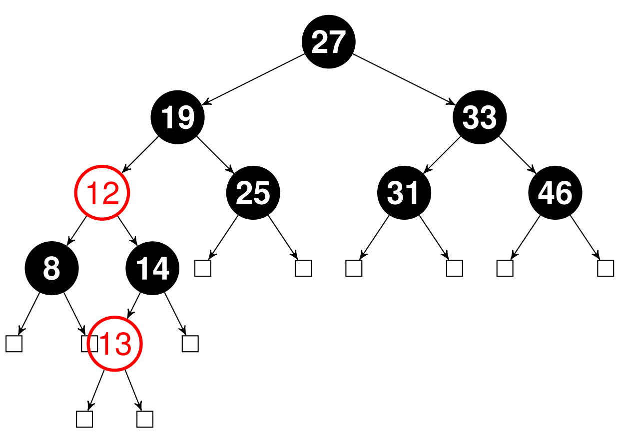
(a) Convert the red-black tree into a 2-4 tree. (Solid nodes are black.)



(b) Insert 13 into the resulting 2-4 tree. Assume that, if a node has 4 keys, we choose to push up the right of the 2 middle keys (so the 2^{nd} key from the right).



(c) Convert the resulting 2-4 tree into a valid left-leaning red-black tree.



(d) Given a (2, 4) tree containing N keys, how would you obtain the keys in sorted order in worst case O(N) time? We don't need actual code—pseudo code or an unambiguous description will do.

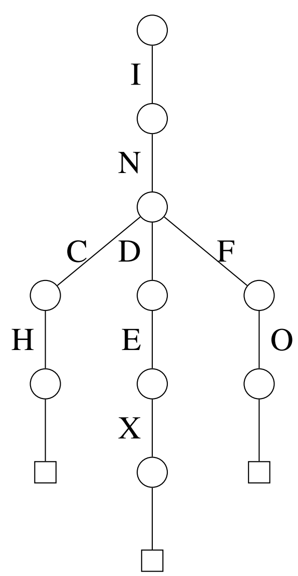
Generalize an inorder traversal: traverse the left (first) child of the node, emit the first key of the node, traverse the second child of the node, emit the second key of the node, etc.

(e) If a (2,4) tree has depth h (that is, the leaves are at distance h from the root), what is the maximum number of comparisons done in the corresponding red-black tree to find whether a certain key is present in the tree?

2h comparisons. The number of black nodes from root to leaf is the same for all nodes in a red-black tree and it is equal to the height of its equivalent (2,4) tree. The maximum number of comparisons occurs from a root to leaf path with the most nodes. We can also have at most one red node after each black node because we can't have more than two red nodes in a row in a red-black tree. Therefore our longest path can be 2h nodes long, and we make at most 2h comparisons.

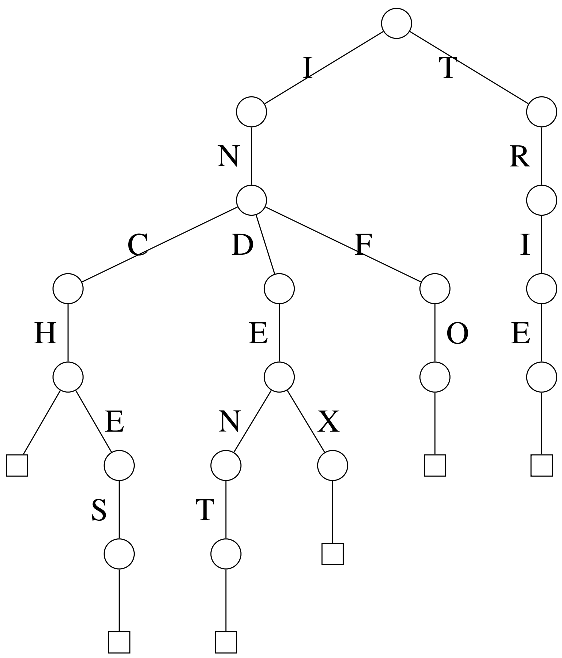
2 Tries

(a) List the words encoded by the following trie.



Encoded words: index, info, inch

(b) Draw the trie after inserting the words *indent*, *inches*, and *trie*.



3 Skip Lists

Draw the resulting skip list after adding the following numbers at the specified random height. Then highlight the links used to find 148.

Number	41	48	59	77	40	131	148	54	139	179	43	128	161	189	170
Height	1	1	1	4	2	2	1	3	1	1	3	2	3	1	2

The links used to traverse to the item are in blue, and the links that overshoot are in red.