Binary Search Trees (10 points)

1. [5 points] Suppose that we have numbers between 1 and 1000 in a binary search tree, and we want to search for the number 363. Which of the following sequences could not be the sequence of nodes examined?

   a. $2, 252, 401, 398, 330, 344, 397, 363.$

   b. $924, 220, 911, 244, 898, 258, 362, 363.$

   c. $925, 202, 911, 240, 912, 245, 363.$

   d. $2, 399, 387, 219, 266, 382, 381, 278, 363.$

   e. $935, 278, 347, 621, 299, 392, 358, 363.$

   Solution: Options c and e are not possible.

   Option c could not be the sequence of nodes explored because we take the left child from the 911 node, and yet somehow manage to get to the 912 node which cannot belong the left subtree of 911 because it is greater. Option e is also impossible because we take the right subtree on the 347 node and yet later come across the 299 node.

2. [2 points] An alternative method of performing an inorder tree walk of an n -node binary search tree finds the minimum element in the tree by calling TREE-MINIMUM and then making $n-1$ calls to TREE-SUCCESSOR. Prove that this algorithm runs in $\Theta(n)$ time.

   Solution: To show this bound on the runtime, we will show that using this procedure, we traverse each edge twice. This will suffice because the number of edges in a tree is one less than the number of vertices.

   Consider a vertex of a BST, say x. Then, we have that the edge between $x.p$ and x gets used when successor is called on $x.p$ and gets used again when it is called on the largest element in the subtree rooted at x. Since these are the only two times that that edge can be used, apart from the initial finding of tree minimum. We have that the runtime is $O(n)$. We trivially get the runtime is $\Omega(n)$ because that is the size of the output.

3. [3 points] We can sort a given set of n numbers by first building a binary search tree containing these numbers (using TREE-INSERT repeatedly to insert the numbers one by one) and then printing the numbers by an inorder tree walk. What are the worst-case and best-case running times for this sorting algorithm?

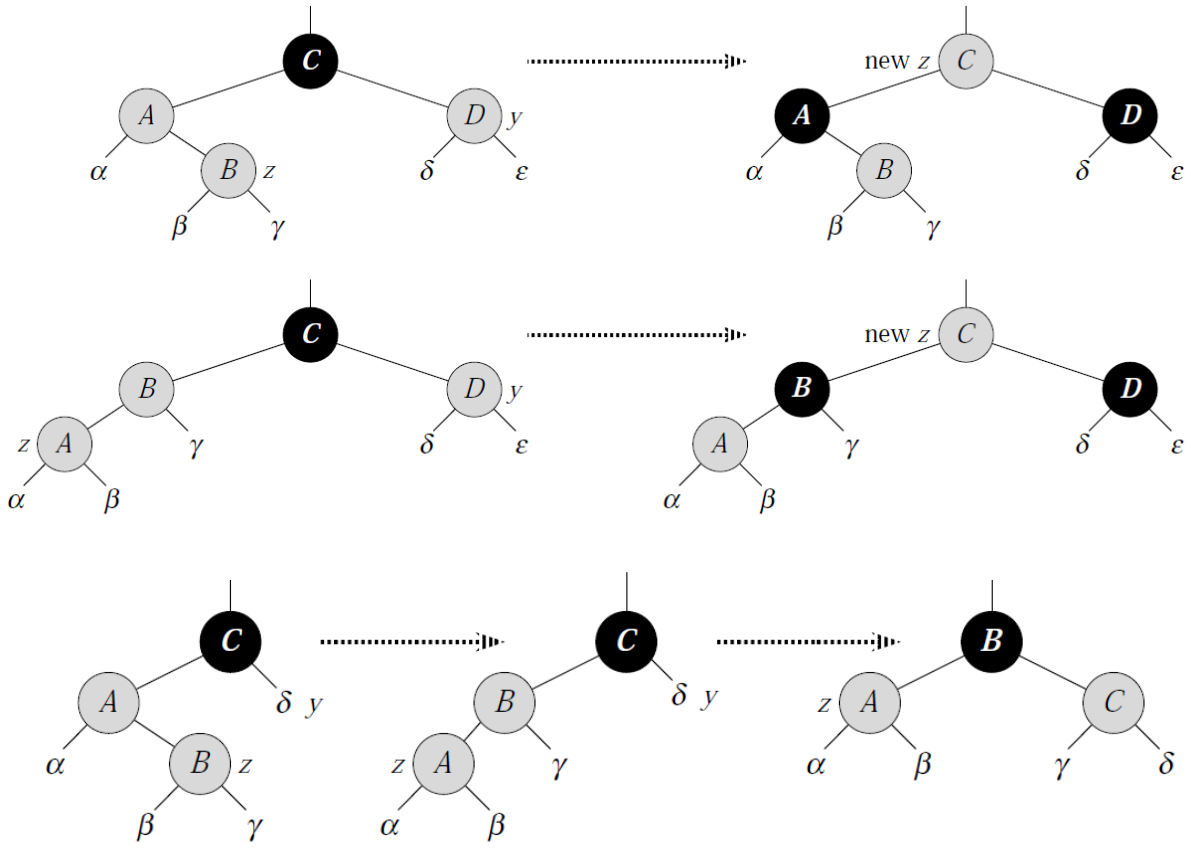
   Solution: The worst case is that the tree formed has height n because we were inserting them in already sorted order. This will result in a runtime of $\Theta\left(n^2\right)$.

   In the best case, the tree formed is approximately balanced. This will mean that the height doesn't exceed $O(\lg(n))$. Note that it can't have a smaller height, because a complete binary tree of height h only has $\Theta\left(2^h\right)$ elements. This will result in a runtime of

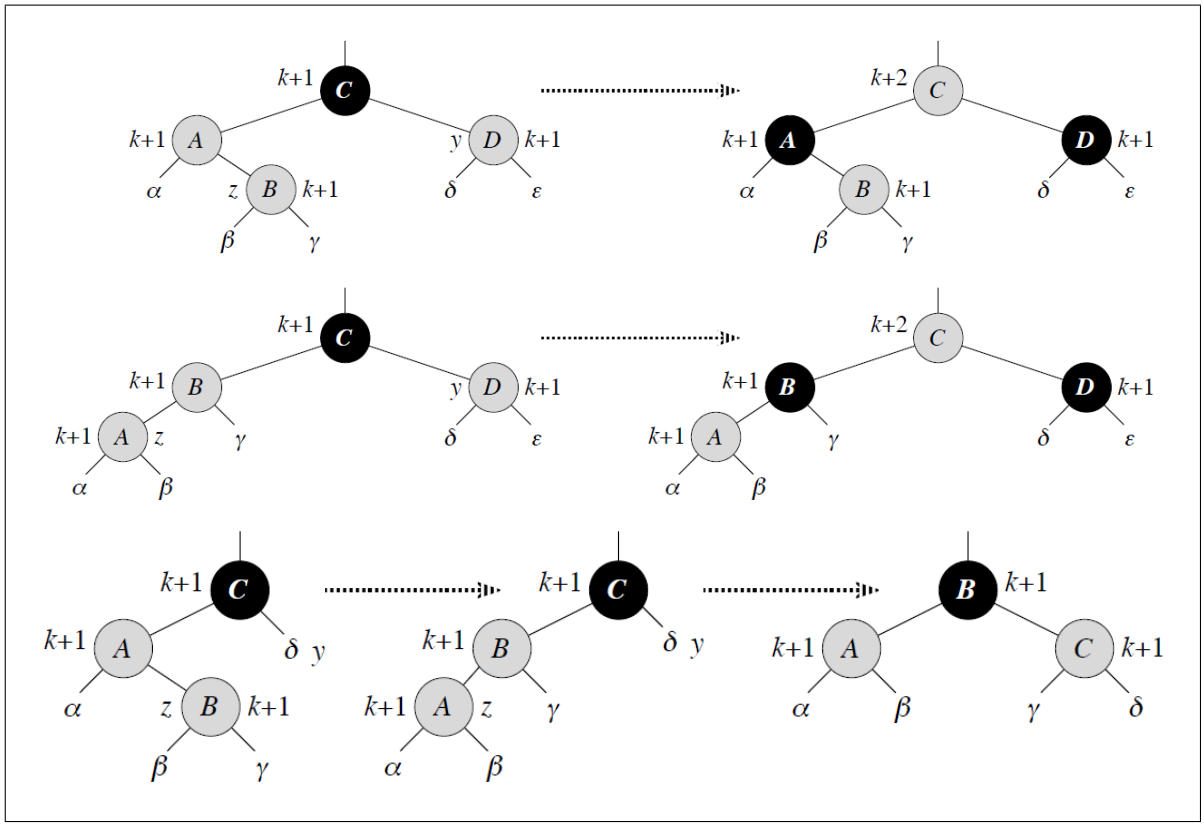
$O(n \lg(n))$. To show $\Omega(n \lg(n))$ (exercise 12.1-5): suppose to a contradiction that we could build a BST in worst case time $o(n \lg(n))$. Then, to sort, we would just construct the BST and then read off the on elements in an inorder traversal. This second step can be done in time $\Theta(n)$ by Theorem 12.1. Also, an inorder traversal must be in sorted order because the elements in the left subtree are all those that are smaller than the current element, and they all get printed out before the current element, and the elements of the right subtree are all those elements that are larger and they get printed out after the current element. This would allow us to sort in time $o(n \lg(n))$ a contradiction.

Red-Black Trees (5 points)

4. [4 points] Suppose that the black-height of each of the subtrees $\alpha, \beta, \gamma, \delta, \varepsilon$ in Figures 13.5 and 13.6 (both below) is k. Label each node in each figure with its black-height to verify that property 5 is preserved by the indicated transformation.



Solution:


k+1
C
k+1
A
y
D
k+1
α
z
B
k+1
δ
ε
β
γ
k+2
C
k+1
A
D
k+1
α
B
k+1
δ
ε
β
γ
k+1
C
k+1
B
y
D
k+1
k+1
A
z
γ
δ
ε
α
β
k+2
C
k+1
B
D
k+1
k+1
A
γ
δ
ε
α
β
k+1
C
δ
y
k+1
A
α
z
B
k+1
β
γ
k+1
C
δ
y
k+1
B
k+1
A
z
γ
α
β
B
k+1
k+1
A
C
k+1
α
β
γ
δ