1. Exercise 9 (pages 536)

- add c: Q (same)
  - K (same)
    - D
      - B
      - C
- add z: Q (same)
  - T
    - R
      - W
    - Y
  - N
- add x: Q (same)
  - T
    - R
    - Y
    - X

- delete H: Q (same)
  - K (same)
    - P
      - N

- delete Q: Q (same)
  - D
    - M
    - N

- delete R: Q (same)
  - T
    - Y
    - W

- This solution uses the predecessor.
- Another solution would be using the successor (i.e., R).
2. Exercise 10 (page 536 – the tree is shown on page 535)
   
   (a) BDJKMNQORTWY
   (b) BJDNPMKRWYTO
   (c) QKDBJMPNTRYW

3. Exercise 19 (page 537)
   
   (a) Elements inserted in random order:
       Linked list: O(N)
       Binary search tree: O(log2N)
   (b) Elements inserted in order:
       Linked list: O(N)
       Binary search tree: O(N)

4. Exercise 29 (page 538) – using big-O notation, analyze running time requirements.

   (a)
   ```cpp
   bool IsBST(); // prototype
   // Post: true is returned if root is the root of a binary
   // search tree; otherwise, false is returned.
   template<class ItemType>
   bool IsTrue(TreeNode<ItemType>*)
   // Returns true if root is the root of a binary search tree; //
   // returns false otherwise
   
   (b)
   bool TreeType<ItemType>::IsBST()
   // Calls recursive function IsTrue.
   { return IsTrue(TreeNode<ItemType>* tree); }
   bool IsTrue(TreeNode<ItemType>* tree)
   { if (tree == NULL)
       return true;
     else if (tree->left != NULL &&
              tree->left->info > tree->info)
       return false
     else if (tree->right != NULL &&
              tree->right->info <= tree->info)
       return false
     else
       return IsTrue(tree->left) && IsTrue(tree->right);
   }
   
   Running Time: O(N) since every node is visited in the worst case.

5. Exercise 36 (page 539) – using big-O notation, analyze running time requirements.
bool MatchingItems(TreeType<ItemType> tree, SortedType<ItemType> list)
// Post: True is returned if tree and list contain the same
//       values.
{
    bool same = true;
    int length = list.LengthIs();
    int count = 0;
    ItemType item;

    while (count < length and same)
    {
        list.GetNextItem(item);
        tree.RetrieveItem(item, same);
        count++;
    }
    return same;
}

Assuming that the list contains $N_1$ elements and the tree $N_2$ elements, then the running
time: $O(N_1 \log(N_2))$ if the tree is balanced or $O(N_1N_2)$ if the tree is unbalanced.