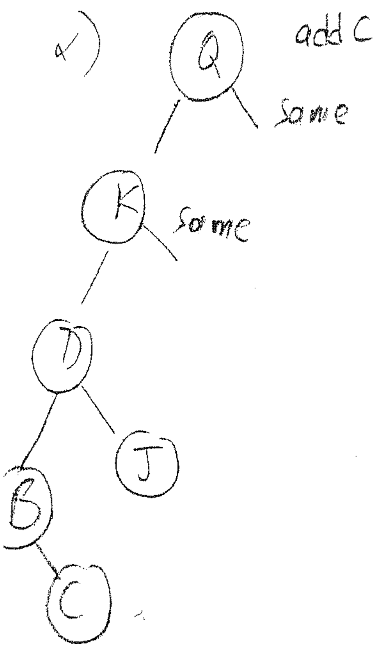
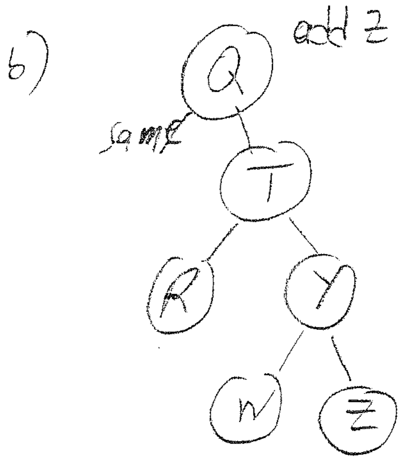
CS302 Data Structures

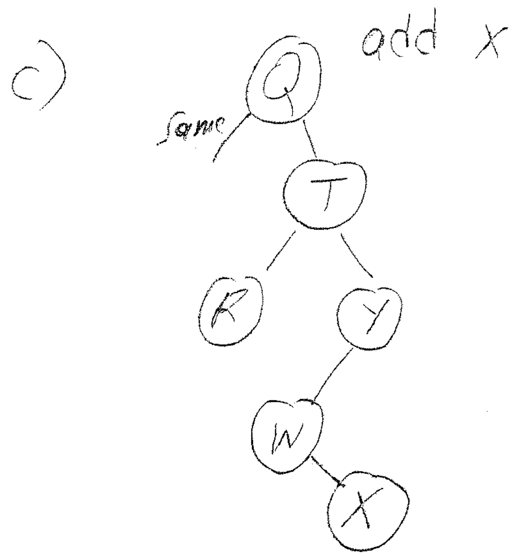
Spring 2010 – Dr. George Bebis

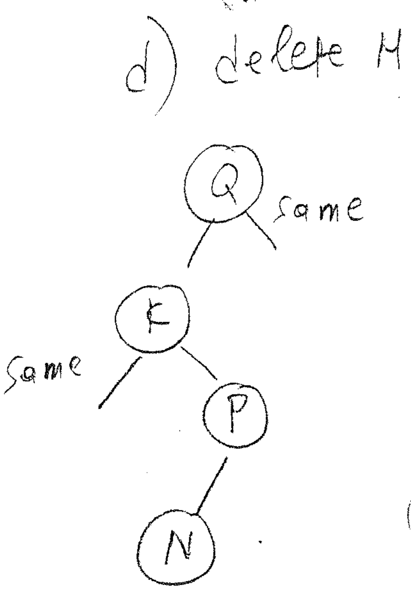
Homework 5 - Solutions

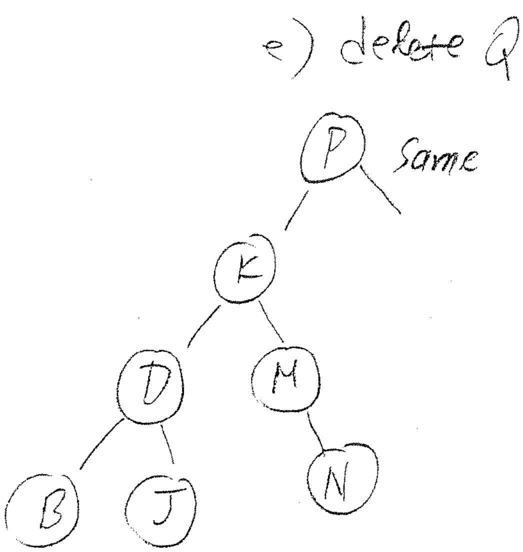
1. Exercise 9 (pages 536)

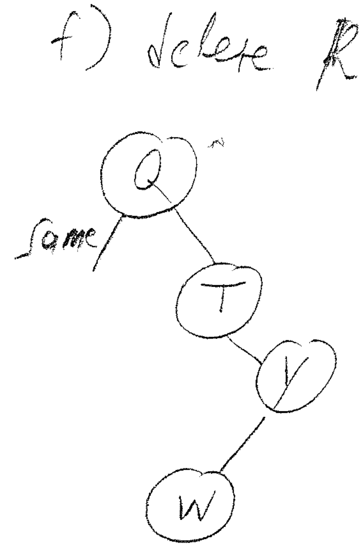












- this solution uses the predecessor
- another solution would be using the successor (i.e., R)

2. Exercise 10 (page 536 – the tree is shown on page 535)

(a) BDJKMNPQRTWY
(b) BJDNPMKRWYTQ
(c) QKDBJMPNTRYW

3. Exercise 19 (page 537)

(a) Elements inserted in random order:
Linked list: O(N)
Binary search tree: $O(\log_2 N)$
(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)
```
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; //
retuns 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.