Data Structures, Practice Homework 3, with Solutions (not to be handed in)

1. Carrano, 4th edition, Chapter 9, Exercise 1: What is the order of each of the following tasks in the worst case?

   (a) Computing the sum of the first n even integers by using a `for` loop.

   (b) Displaying all n integers in an array.

   (c) Displaying all n integers in a sorted linked list.

   (d) Displaying all n names in a circular linked list.

   (e) Displaying one array element.

   (f) Displaying the last integer in a linked list.

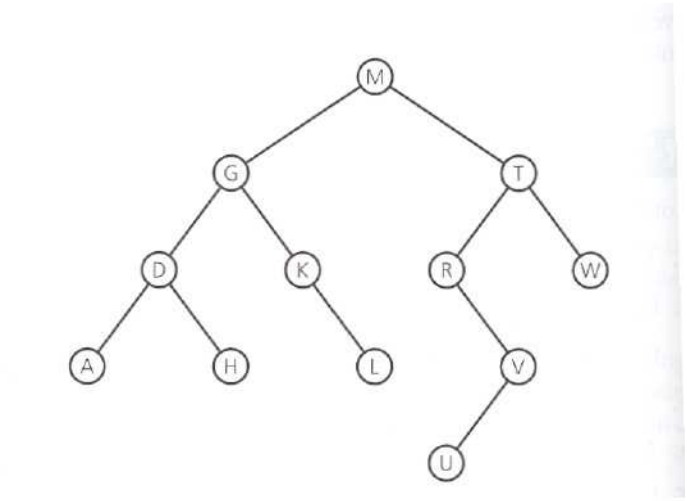
   (g) Searching an array of n integers for a particular value by using a binary search.

   (h) Sorting an array of n integers into descending order by using a merge-sort.

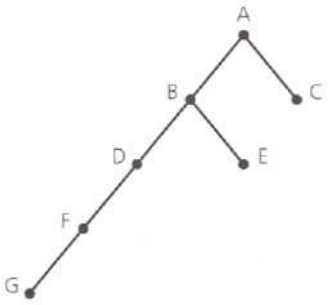
   (i) Adding an item to a stack of n items.

   (j) Adding an item to a queue of n items.

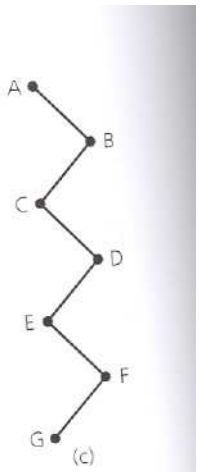
2. Carrano, 4th edition, Chapter 10, Exercise 2: What are the preorder, inorder, and postorder traversals of the following binary trees:



(a)



(b)



(c)

3. Carrano, 4th edition, Chapter 10, Exercise 4: Beginning with an empty binary search tree, what binary search tree is formed when you insert the following values in the order given?

   (a) W, T, N, J, E, B, A

   (b) W, T, N, A, B, E, J

   (c) A, B, W, J, N, T, E

4. Carrano, 4th edition, Chapter 10, Exercise 9:

   (a) Beginning with an empty binary search tree, what binary search tree is formed when the following data is inserted in the order given? $8, 13, 6, 10, 21, 19$.

(b) Given a search key of 12, trace the algorithm that searches the binary search tree that you created in Part a. List the nodes in the order in which the search visits them.

5. Carrano, 4th edition, Chapter 10, Exercise 37. Add an overloaded `==` operator to the class `BinaryTree`.

Solutions

1. Carrano, 4th edition, Chapter 9, Exercise 1: What is the order of each of the following tasks in the worst case?

   (a) Computing the sum of the first n even integers by using a `for` loop.
   Solution: $O(n)$.

   (b) Displaying all n integers in an array.
   Solution: $O(n)$.

   (c) Displaying all n integers in a sorted linked list.
   Solution: $O(n)$.

   (d) Displaying all n names in a circular linked list.
   Solution: $O(n)$.

   (e) Displaying one array element.
   Solution: $O(1)$. For an array, is very fast to access even the last array element `theArray[n-1]`.

   (f) Displaying the last integer in a linked list.
   Solution: $O(n)$. For a (singly) linked list, we have to access every element from the beginning starting from the `head` pointer. The code will look something like this (assuming the list is not empty):

```
for (ListNode* prev = head; prev->next != NULL;
     prev = prev->next)
  ; // empty body of loop
cout << prev->item;
```

   (g) Searching an array of n integers for a particular value by using a binary search.
   Solution: $O(\log_2 n)$.

   (h) Sorting an array of n integers into descending order by using a mergesort.
   Solution: $O(n * \log_2 n)$. Recall that the mergesort algorithm always has the same number of steps (so there is no best or worst case).

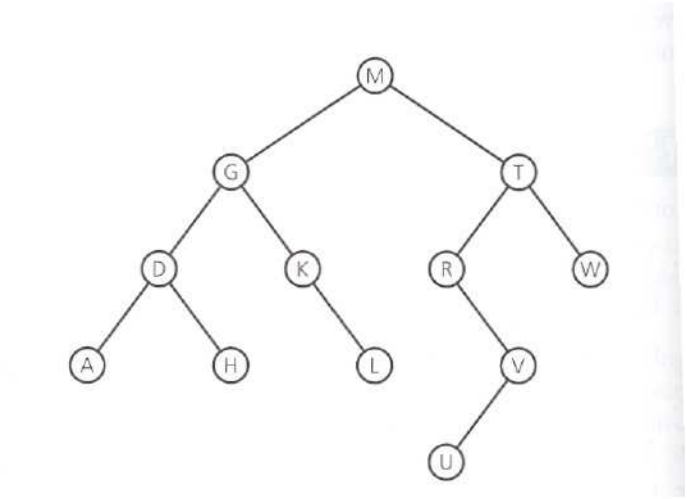
   (i) Adding an item to a stack of n items.
   Solution: $O(1)$. All that needs to be done is something like this:

```
newTop = new StackNode;
newTop->item = anItem;
newTop->next = top;
top = newTop;
```

   (j) Adding an item to a queue of n items.
   Solution: $O(1)$. All that needs to be done is this (at least in the case the queue is not empty):

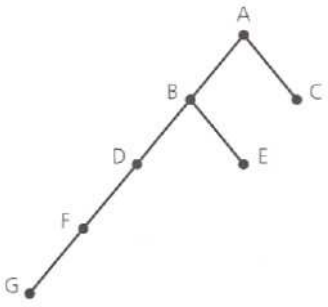
```
newPtr = new QueueNode;
newPtr->item = anItem;
newPtr->next = NULL;
backPtr->next = newPtr;
backPtr = newPtr;
```

2. Carrano, 4th edition, Chapter 10, Exercise 2: What are the preorder, inorder, and postorder traversals of the following binary trees:



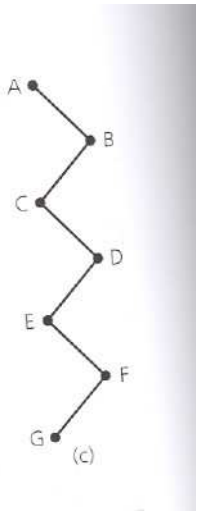
(a)

Solution: preorder: MGDAHKLTRVUW
inorder: ADHGKLMRUVTW
postorder: AHDLKGUVRWTM



(b)

Solution: preorder: ABDFGEC
inorder: GFDBEAC
postorder: GFDEBCA

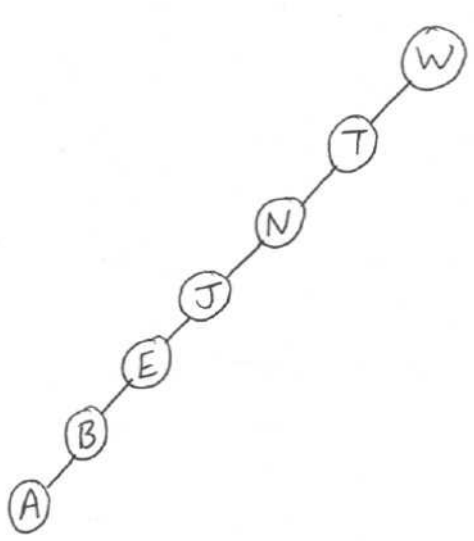


(c)

Solution: preorder: ABCDEFG
inorder: ACEGFDB
postorder: GFEDCBA

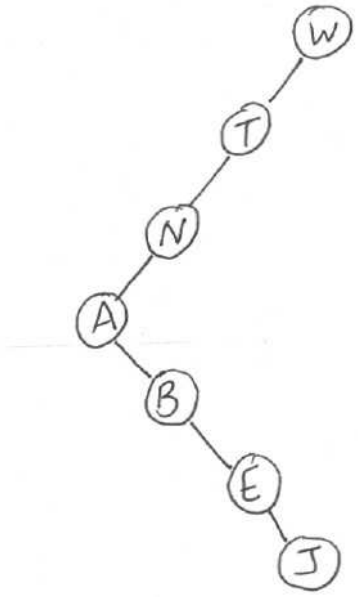
3. Carrano, 4th edition, Chapter 10, Exercise 4: Beginning with an empty binary search tree, what binary search tree is formed when you insert the following values in the order given?

   (a) W, T, N, J, E, B, A



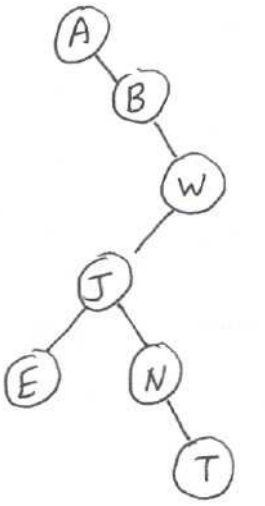
Solution:

   (b) W, T, N, A, B, E, J



Solution:

(c) A, B, W, J, N, T, E

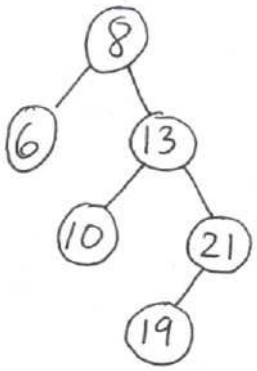


Solution:

4. Carrano, 4th edition, Chapter 10, Exercise 9:

   (a) Beginning with an empty binary search tree, what binary search tree

is formed when the following data is inserted in the order given? $8, 13, 6, 10, 21, 19$.



Solution:

(b) Given a search key of 12, trace the algorithm that searches the binary search tree that you created in Part a. List the nodes in the order in which the search visits them.

Solution: 8 is the root: $12 > 8$: go right
node 13: $12 < 13$: go left
node 10: $12 > 10$: go right
cannot go right: 10 has no right child node: search fails

5. Carrano, 4th edition, Chapter 10, Exercise 37. Add an overloaded `==` operator to the class `BinaryTree`.

Solution: It's useful to implement this using an auxiliary function `subtreeEqual`, in order to make the recursion work

```
bool BinaryTree::subtreeEqual (TreeNode* lhsRoot,
                               TreeNode* rhsRoot) const{
  if (lhsRoot == NULL && rhsRoot == NULL)
    return true; /* base case: if both trees are empty, they're
                    equal
                 */
  // now we know at least one tree is not empty

  if (lhsRoot == NULL || rhsRoot == NULL)
    return false; /* base case: if one tree is empty, now we know
                     the other is not, and so they're not equal
                  */

  return (lhsRoot->item == rhsRoot->item)
          && subtreeEqual(lhsRoot->leftChildPtr,
                          rhsRoot->leftChildPtr)
```

```
          && subtreeEqual(lhsRoot->rightChildPtr,
                          rhsRoot->rightChildPtr);
  /* Note we first check to see if the items of the roots are
     equal.  If they are not, the function returns false.  If
     they are, then we recursively call subtreeEqual with the
     left subtree, and then with the right subtree.  The trees
     are equal only if the roots' items and each subtree are
     equal.
  */
}  // end subtreeEqual

bool BinaryTree::operator==(const BinaryTree& rhs) const{
  return subtreeEqual (root, rhs.root);
}  // end operator==
```