CMSC 341 Data Structures
List Review Questions

Please refer to the List, ListNode and ListItr class definitions found on the last two page. These definitions are the same as those found in the text and in the class notes. You may assume that all member functions of these classes have been written and work properly when answering the questions below.

1. Write a new member function of the List class named **last( )**. Similar to **first( )** and **zeroth( )**, this function returns an iterator to the last data element of the list. If the list is empty, the iterator returned is "past end". What is the asymptotic performance of **last( )**?

2. Overload the equality operator (operator==) for the **ListItr** class that returns true if both iterators refer to the same element of the list

3. Write a new member function of the List class named **ReversePrint** that uses ListItrs to display the elements of the List in reverse order. The one and only parameter to ReversePrint is the output stream to which the elements are displayed. The data elements should be enclosed in angle brackets ("< > ") and separated by commas. Do not construct a copy of the list that is in reverse order, just use iterators. You may assume that operator== (as described above) has been implemented for ListItrs. The prototype for **ReversePrint( )** is shown below

   ```
   template< class Object >
   void List<Object>::ReversePrint( ostream& out );
   ```

4. Write a new function named **Splice( )** whose prototype is shown below. ***Note that Splice( ) is not a member function of the List class***. This function "splices" L2 into L1 at the specified position (pos is an iterator over L1). If pos is past end, **Splice( )** does nothing. For example suppose L1 = {1, 2, 3, 4, 5 } and L2 = {10, 20, 30 } and pos is constructed as list1.first( ) and then advanced twice so it is positioned at the "3". Then the function call L1.Splice(L2, pos); causes L1 to become { 1, 2, 3, 10, 20, 30, 4, 5 } and L2 is unchanged.

   What is the asymptotic performance of **Splice( )**? Bear in mind that there are two lists which may be of different lengths.

```
template< class Object >
void Splice( List<Object>& L1,
             const List<Object>& L2,
             ListItr<Object>& pos);
```

5. Complete the following table of Big-Oh, worst-case asymptotic time performance for the given operations and implementations of the List ADT. Give your answers in terms of, n, the number of elements in the list.

Operation	Singly Linked List	Array/Vector
insert		
find		
remove		
makeEmpty		
isEmpty		
first		

6. Suppose you are provide with a set of N random numbers which are to be inserted into a sorted List (smallest to largest). What would be the worst-case asymptotic time performance for building the entire list?

7. Your boss has insisted that you add a new member function for the ListItr class named retreat( ) which moves the iterator one node "backward" (toward the header node). What is the asymptotic time performance for retreat( ) if the underlying List is singly linked? doubly linked?

8. Describe the advantages (if any) and disadvantages (if any) of each of the List ADT implementation – singly linked list, circular linked list, doubly linked list, doubly circular linked list, array/vector, cursor space.

9. Do the following exercises in chapter 3 of the text – 3.1, 3.2 (write code), 3.9, 3.10.

Definition of the ListNode Class

This definition is taken from the text.

```
template< class Object >
class ListNode
{
    ListNode( const Object& theElement = Object( ) ),
              ListNode *n = NULL )
        : element( theElement ), next( n ) { }
    Object    element;
    ListNode *next;
    friend class List< Object >;
    friend class ListItr< Object >;
};
```

Definition of the List Class

This definition is taken directly from the text.

```
template< class Object >
class List
{
    public:
        List( );
        List( const List& rhs );
        ~List( );
        bool isEmpty( ) const;
        void makeEmpty( );
        ListItr< Object > zeroth( ) const;
        ListItr< Object > first( ) const;
        void insert( const Object& x, const ListItr< Object >& p);
        ListItr< Object > find( const Object& x ) const;
        ListItr< Object > findPrevious( const Object& x ) const;
        void remove( const Object& x );
        const List& operator=( const List& rhs );
    private:
        ListNode< Object > *header;
};
```

Definition of the ListItr Class

This definition is taken directly from the text.

```
template< class Object >
class ListItr
{
    public:
        ListItr( ) : current ( NULL ) { }
        bool isPastEnd( ) const
            { return current == NULL }
        void advance( )
            { if( !isPastEnd( ) ) current = current->next; }
        const Object& retrieve( ) const
            { return current->element; }
    private:
        ListNode< Object > *current;
        ListItr( ListNode< Object > *theNode
          : current( theNode ) { }
    friend class List< Object >; // grant access to private constructor
};
```