CSC 360 Programming Assignment #10: Implementing List ADTs
Due Date: Friday, December 5

In this assignment you will implement 4 List ADTs as classes. All four of your classes will implement the following methods (all are void unless they are turn an int):

- constructor(int) or constructor( ) (see below)
- insert(int)
- search(int) : int (return the index of the item)
- delete(int)
- traverse( )
- getComparisons( ): int
- sort( ) (this is only required for your third class)

The 4 classes implement Lists and 3 of them will maintain the lists as ordered lists. The 4 classes are:

- Class 1: An ordered linked list
- Class 2: An array where inserts are done in order
- Class 3: An array where inserts are done at the end followed by calling a sort method
- Class 4: An array where inserts are done at the end of the list and is unsorted

The purpose of the assignment is to compare how each of these implementations performs when presented by random and non-random int values. In general, we should expect that the ordered array with ordered inserts performs the best and the unsorted array performs the worst.

Classes 2-4 are array implementations and so should be straight-forward to implement. For classes 2 & 3, use binary search for the search operation, which will return the index of the item located (or -1 if not found). Similarly, in delete, call binary search to get the index and if it is not -1 delete the given item by shifting all remaining items down 1 position. Insertion in Class 2 requires that you start at the end of the array and shift elements up one at a time until you find the proper insertion point. For Class 3, insert the new item at the end of the array and then call your sort method. Use the Bubble Sort, given below. For Class 4, always insert at the end. For Class 4, to delete, you must find the element first and then move the last item of the array into this position. For instance, if we are deleting 12 out of: 3 19 12 9 18 4 11 then we would move 11 into the position currently held by 12 and then decrement the number of elements in the array effectively giving us the array 3 19 11 9 18 4. Searches require sequential search (not binary).

The bubble sort, given an array a whose length is number

```
sorted=false
while(!sorted)
      sorted=true
      for(i=0;i<number-1;i++)
            if(a[i]>a[i+1])
                  swap a[i], a[i+1]
                  sorted=false
```

In all 4 of your classes, count each comparison made in your search method, sort method and insert method. The getComparisons accessor returns the total number of comparisons made by that class to that point of the program. We will use this to compare the performance of the four lists.

Class 1 requires that you implement an ordered linked list. First, implement an inner class called Node. Do not make Node or your linked list Generic as we will only deal with int values. The Node class requires the following methods:

Constructor(data: int, next: Node)
getNext( ): Node
setNext(next: Node)
getData( ): int
setData(data: int)

Your LinkedList class will have two instance data, Node front (the reference to the linked list, initialized to null), and int comparisons. Your insert method will need two references, the current pointer and the previous pointer. You will move the pointers down the linked list until you find the first Node storing a datum greater than x. You will insert a new Node between previous and current. To delete an item, similarly use two pointers. If the item is found, current will point at it and previous will point at the previous Node. You will need to redirect previous's next field to point at current's next field. The search and traverse methods only need a single pointer.

All four Classes need a getComparisons accessor. Although you should implement a traverse method, you will not actually use it in the assignment but you might find it useful for debugging purposes.

When done, create a fifth class, a User class. The user class will create an array of 100 elements. The user class will first create a list of values and store them in an int array list. This will be used so that you can insert those values and later go back and search or delete those values. Next, iterate through list and insert each list[i] into all four lists. Next, iterate through the elements of list and alternately either search for list[i] or delete list[i]. The User program code will look something like this:

```
// create the 4 lists, set max to 100
int[] list=new int[max];      // to store the values to be used
int i, temp;
for(i=0;i<max;i++) list[i]= // * generate values to be used
for(i=0;i<max;i++)  {
      // insert list[i] into all 4 lists
}
for(i=0;i<max;i++) {
      temp=list[g.nextInt(max)];  // get random element to use
      if(i%2==0) {
            // search for list[temp] in all 4 lists
      }
      else {
            // delete list[temp] from all 4 lists
      }
// Output the number of comparisons performed in each of the 4 lists
```

You will run your program 5 times. In the first run, you will generate the list[i] array to be elements in order (see * above) using list[i]=i. In the second run, use list[i]=100-i to insert items in descending order. For the final 3 runs, set list[i]=g.nextInt(1000). This will insert items in random order.

Add to your User class comments that explain the results you achieved in the five runs. That is, state the order that the classes performed from best to last for each run and try to explain why as best you can. Hand in your five classes and the output of the three runs. Your output might look like this (this is the output I received for one of the randomly generated runs):

```
Ordered linked list comparisons:  6290
Ordered array comparisons:  946
Sorted array comparisons:  4582
Unordered array comparisons:  5043
```