CSC 228-60 Data Structures and Algorithms (12 PM section)
Spring 2018
Instructor: Dr. Natarajan Meghanathan
Project 1: Algorithm Design and Time Complexity Analysis for Operations on the Dynamic Array Implementation of the List ADT

Due: Feb. 7th, 2018 @ 12 PM

You are given the code for the implementation of the List class using dynamic arrays. The code has been written in such a way that it can calculate the actual run time (in milliseconds) for the two functions (reverseList and segregateEvenOdd) you will be adding to the List class.

As mentioned in the code, you will be getting three inputs from the user: (i) the list size (ii) the maximum value for an element in the list and (iii) the number of trials to run your program. The list size is varied from 1000000 to 10000000 (i.e., 10^6 to 10^7), in increments of 1000000 (i.e., 10^6). That is, the values for the list size are: 1000000, 2000000, 3000000, ..., 9000000, 10000000. For all values of list size, the maximum value for an element in the list is 5000000 (i.e., $5*10^6$) and the number of trials to run the program for each value of list size is 20.

The code is structured as follows: After getting the input for the above three variables, the code will create a random array of integers of the specified list size (with values ranging from 1... maximum value). You will then call the reverseList( ) function (the code for which is to be implemented by you) on the created integer list and reverse the list. On the reversed integer list, you will call the segregateEvenOdd function and rearrange the values in the list in such a way that it has all the even integers appearing first followed by the odd integers.

Each time you run the code (with a particular value of list size), it will run for all the trials and give you the average run time (in milliseconds) for the reverseList and segregateEvenOdd operations on the list.

Your algorithms for both reverseList( ) and segregateEvenOdd( ) should each run in O(n) time with O(1) additional memory used (i.e., a constant amount of additional memory that is independent of the size of the list). More details about the reverseList( ) and segregateEvenOdd( ) functions are as follows:

The **reverseList( ) member function** for the List class should reverse the contents of the list. For example if the contents of the List are (before reverse): 10 12 5 7 19 21, then after calling the reverseList function, the contents of the List should be updated as: 21 19 7 5 12 10. You should implement the reverseList function without using any additional memory or temporary array that is proportional to the size of the list. Just one temporary variable should be sufficient.

```
void reverseList( ){
.........
}
```

The **segregateEvenOdd( ) member function** for the List class should segregate the contents of the list into a sequence of even integers followed by odd integers. The order of appearance of the even integers (and odd integers) could be different from that in the original list. For example, if the contents of the List are (before segregation into even and odd): 12 34 45 9 8 90 3, then the contents of the List after calling the segregateEvenOdd function could be: 12 34 90 8 9 45 3. Again, you should implement the **segregateEvenOdd( )** function by using only O(1) additional memory.

```
void segregateEvenOdd( ){
........
}
```

Submission (through Canvas)

The report should contain the following:

(i - 50 pts) Complete code for the List class containing all the functions (included in the original code as well as the code for the two functions implemented in this project).

(ii - 10 pts) Pseudo code and explanation of the algorithms that you have come up with to implement for each of the two functions reverseList and segregateEvenOdd

(iii - 10 pts) Theoretical time complexity analysis of the algorithms for each of the two functions and show that they are each O(n).

(iv - 10 pts) Theoretical space complexity analysis of the algorithms for each of the two functions and show that it is O(1); i.e., only constant space (i.e., fixed amount of additional space independent of the list size), if any, is used.

(v - 10 pts) Two Excel plots (showing the list size in X-axis and the actual run time, measured in milliseconds in Y-axis for the function): one plot for the reverseList function and another plot for the segregateEvenOdd function.

(vi - 10 pts) Interpret the actual run-time results for the two functions. Even though the two functions are designed to have the same theoretical time complexity, the actual run time for one of the two functions is expected to be much larger than that of the other function. What is your observation? Why do you think there is a significant difference in the run-time for the two functions?

Note: ***If your system runs out of memory for larger list sizes, run for the largest multiple of 10^6 until which your program will not run out of memory (for example: if your program ran out of memory for list size of 9000000 and ran fine for 8000000, you would report the run times observed for list size of 1000000, 2000000, 3000000, ..., 8000000)***