SLC Report

Assignment 1 – Linked Lists and Arrays

Ken Rivard

CS 3310 10am

Initial Design

In my initial design phase I decided from the start to use Bubble Sort for sorting both the array and linked list. I figured since Bubble Sort is a simple yet effective sorting algorithm that it would be the best option for this program. Initially I started implementing the array first instead of the linked list. I assumed that the array portion of the program would be easier but soon discovered that was not the case.

Changes/Problems/Refinement

One problem I encountered when running the final program was my run times were very long. I then realized that most of the time was spent printing out the large array and linked list. When running for time I simply commented out the printing statements to maximize my run times.

Methods

Array

*randomArray(int low, int range, int max)* method generates a set of random numbers and assigns them to an array then calls my bubbleSort() method/algorithm

*bubbleSort(int randArray[])* method makes use of the bubble sort algorithm and sorts the array

*numGreaterArr(randArray, 50, range)* method checks to see how many integers are larger than the value 50 within the specified range. As per the assignment, if that value > 5 then the array is sorted into a non-decreasing order while deleting the 5th element and inserting the value 10 to its appropriate spot. If the value < 5 then the array is sorted in a non-increasing order while deleting the 2nd element and inserting the value 10 to its appropriate spot.

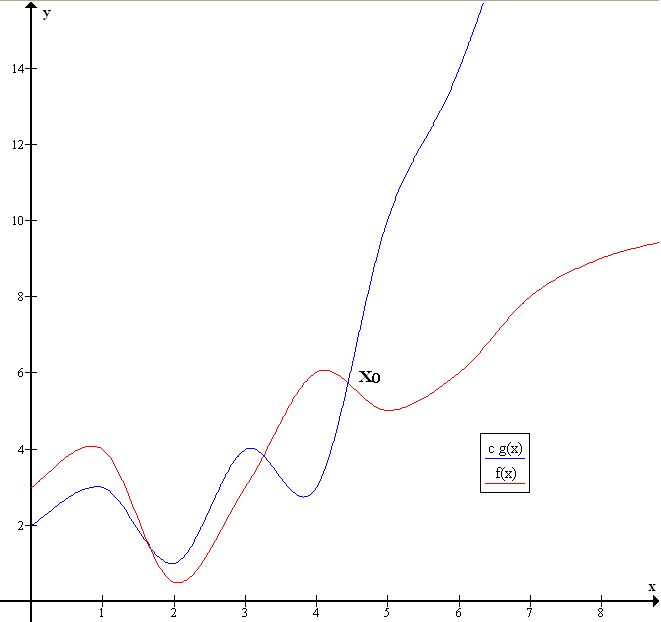
Linked List

*randLinkedList(int low, int range, int max)* method generates a set of random numbers and inputs them into a linked list

*numGreaterList(int val, int range)* is similar to the *numGreaterArr()* method but instead of an array it is for a linked list. It checks if values in the list are greater than 50 and if that number is greater than 5 the linked list is sorted in a non-decreasing order while deleting the 5th element and inserting the value 10 to its appropriate spot. Then if the values greater than 50 are less than 5, the linked list is sorted in a non-increasing order while deleting the 2nd element and inserting the value 10 to its appropriate spot.

*sortLinkedListIn()* sorts the linked list in an increasing order

*sortLinkedListDe()* sorts the linked list in a decreasing order

*display()* simply outputs the array to the console

*deleteAtPosition(pos)* method deletes integer at val

*insertAtPosition(val, pos)* method inserts val at pos

This image explains what happens to the runtime of the program as the input values get larger

Part 7 – Theoretical Complexity

I used Bubble Sort for sorting through both the array and doubly linked list. Bubble sort uses 2 for-loops which, as discussed in class, is O(n^2). Throughout my testing process, I have come to the conclusion that Bubble Sort is efficient and practical when the numbers used aren’t substantially large.

When given the range from 1-100 the program is compiled almost instantaneously. But as soon as the numbers exceed 10,000 the program takes significantly longer to compile and finish. The second largest value I tested, given my machine’s hardware specs, was 50,000 and that took 427747ms (7.12 minutes) to compile and finish running (without printing out the linked list and array). The largest value that I tested was 100,000 and that took 2584338ms (43.07 minutes) to compile and finish running (without printing out the linked list and array).

Part 8 – Empirical Measurements

Bubble Sort (Array) Non-Increasing and Non-Decreasing

Number of times two values are compared: 9

Number of times two values are swapped: 3n^2 + 2n

Final formula: **3n^2 + 2n + 9**

Bubble Sort (Doubly Linked List) Non-Increasing and Non-Decreasing

Number of times two values are compared: 8

Number of times two values are swapped: 3n^2 + 2n

Final Formula: **3n^2 + 2n + 8**

Part 9 – Comparison (7 and 8)

From part 7, it is very noticeable that the jump from 50,000 to 100,000 was almost squared in terms of time (milliseconds) per the big oh of bubble sort.

Sample Output:

Enter a maximum value to generate numbers to:

10000

Enter a maximum number of values to generate:

10000

How many times do you want the program to loop?

10

22840ms for [0] Doubly Linked List

181ms for [0] Array

27694ms for [1] Doubly Linked List

176ms for [1] Array

23113ms for [2] Doubly Linked List

143ms for [2] Array

20302ms for [3] Doubly Linked List

141ms for [3] Array

19886ms for [4] Doubly Linked List

142ms for [4] Array

24139ms for [5] Doubly Linked List

141ms for [5] Array

22135ms for [6] Doubly Linked List

149ms for [6] Array

23202ms for [7] Doubly Linked List

140ms for [7] Array

25315ms for [8] Doubly Linked List

141ms for [8] Array

22356ms for [9] Doubly Linked List

141ms for [9] Array

Process finished with exit code 0

As seen above, when looped through 10 times, the Array function is completed in about the same amount of time whereas the Doubly Linked List function is completed within roughly 5000ms of the others. There is no correlation between the first and last runs of the Doubly Linked Lists and same for the Array, there does not seem to be any direct correlation as to if they run faster after going through a few iterations before.