Alejandra Reyes

3/25/2021

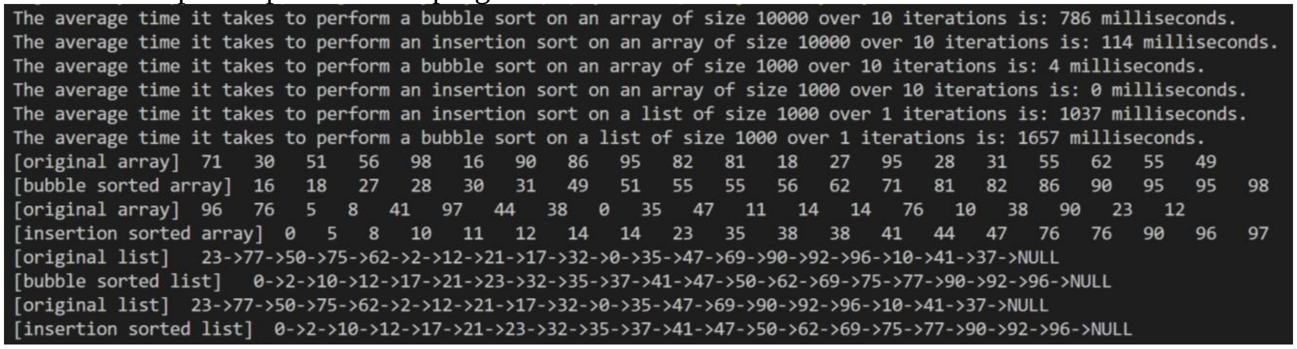
Homework 1 Question 4

Compare the times it takes to sort an array filled with random numbers vs a linked list via bubble sort and insertion sort.

Program Description:

This program compares two different sorting algorithms on arrays and linked lists. To test the sorting algorithms on an array, an array is filled with random numbers, and each sorting algorithm is timed over a set amount of iterations. Insertion sort on an array selects an unsorted array element, and compares it with all of the elements before it. The current unsorted element is then inserted into the correct position, and then the next element is sorted, if necessary. Bubble sort on an array compares each element in the array to the element next to it, and swaps the elements until the current element is placed in the correct position.
The program compares list sorting times by using templated node and linked list classes. The key functions in the linked list class that allow the search to function properly are the insert, remove, getData, and getNode functions. These functions allow for the construction of a linked list bubble and insertion sort that have a similar structure to the array sorting algorithms. However, this implementation has the consequence of requiring that the list be traversed beginning from the head each time we need to make a comparison, or move a node to another spot. While not an efficient algorithm, it successfully sorts the lists. A better implementation would have been to keep track of the current node, and check the node next to it. However, insertion and swapping would have still required traversing the list from the head. This is a consequence of the singly linked list structure because we cannot randomly access the nodes, we must traverse to a node from the head of the list.

Here is a sample output from the program:



```
The average time it takes to perform a bubble sort on an array of size 10000 over 10 iterations is: 786 milliseconds.
The average time it takes to perform an insertion sort on an array of size 10000 over 10 iterations is: 114 milliseconds.
The average time it takes to perform a bubble sort on an array of size 1000 over 10 iterations is: 4 milliseconds.
The average time it takes to perform an insertion sort on an array of size 1000 over 10 iterations is: 0 milliseconds.
The average time it takes to perform an insertion sort on a list of size 1000 over 1 iterations is: 1037 milliseconds.
The average time it takes to perform a bubble sort on a list of size 1000 over 1 iterations is: 1657 milliseconds.
[original array]  71   30   51   56   98   16   90   86   95   82   81   18   27   95   28   31   55   62   55   49
[bubble sorted array]  16   18   27   28   30   31   49   51   55   55   56   62   71   81   82   86   90   95   95   98
[original array]  96   76   5   8   41   97   44   38   0   35   47   11   14   14   76   10   38   90   23   12
[insertion sorted array]  0   5   8   10   11   12   14   14   23   35   38   38   41   44   47   76   76   90   96   97
[original list]   23->77->50->75->62->2->12->21->17->32->0->35->47->69->90->92->96->10->41->37->NULL
[bubble sorted list]   0->2->10->12->17->21->23->32->35->37->41->47->50->62->69->75->77->90->92->96->NULL
[original list]  23->77->50->75->62->2->12->21->17->32->0->35->47->69->90->92->96->10->41->37->NULL
[insertion sorted list]  0->2->10->12->17->21->23->32->35->37->41->47->50->62->69->75->77->90->92->96->NULL
```

Problems Encountered:

The linked list sorts were difficult to implement because it was tricky to decide how to move the nodes into the proper position. A workaround for this includes the use of getData and getNode, as well as insertion and removal in a particular spot. An offset had to be included in the list insertion sort so we could keep track of the index of the element to be removed, as its position changes each time a node is inserted.

Because the linked list sorts try to access a node at a specified index frequently, its sort functions perform poorly, and it is difficult to test sorting on lists larger than 1000 because of how long it takes.

Conclusion:

It is clear from the results that insertion sort is slightly faster than bubble sort for both arrays and linked lists. Insertion sort and bubble sort are both O(n^2) because they use two loops. However, insertion sort is slightly faster because it makes fewer comparisons overall. Sorting a linked list in this case took much longer because of the way the sorts were implemented. The list sorts utilized node address and node data getter functions to make the sorting algorithm similar to an array sorting algorithm. This greatly increases the sorting time because each time we need to get, remove, or insert a list element at a specific position, we need to traverse the list from the head until we reach the index we are looking for. The sorts, however are functional, although not the most efficient.