Assignment: Project Sorting

Zinedin Bautista

Professor Alex Lai

University of Advancing Technology

Data Structures and Algorithms

August 15, 2020

Assignment: Project Sorting

Introduction: The two sorting algorithms I decided to code and implement are the bubble sorting algorithm and the insertion sorting algorithm with a double link list. Within the bubble sort that operates on the double linked list data structure. The algorithm that checks the elements of two adjacent nodes in the double linked list and see if the node on the left is bigger or smaller than the node on the right. If the node on the left is bigger than the one on the right then the two elements will swap with each other. This procedure is repeatedly done until every element is assigned from smallest to largest. Then you have the insertion sorting algorithm which has been given the sample in which you imagine you're sorting playing cards. You select the first element and keep swapping other elements to the right until the picked element is in the right position. So the difference between the bubble sort and insertion sort is that the insertion sorting algorithm will go through all the elements once, while the bubble sort algorithm might need to go through all the nodes more than once.

Then you have the double linked list which is a data structure that consists of nodes. Each of these nodes contain two links in which the first link will point to the previous node and the second node will point to the next node in the list. The first node will always have its previous node pointing to NULL.

Programmer's Guide:

BubbleSort.cpp:

(This file contains the bubble sort algorithm)

First you have the node structure which contains information such as the data on the node, a pointer reference for the next pointer and a pointer reference to the previous pointer.

Then you have the function declaration for insertBubbleSortNode (function that will insert node), displayBubleSortList (function to display the list), and bubbleSortingAlgorithm (function that contains the actual bubble sorting algorithm.

The next thing is the function definition named insertBubbleSort node which will insert a node at the double linked list. The function definition for insertBubbleSortNode contains a Variable that will create a node and always insert it into the beginning of the double linked list. Which will have a pointer pointing at the beginning node and. Will have access to data and also the beginning pointer node will have access to the next pointer. While being set equal to the address of startReference. After that is an if statement that will basically check if the double linked list is empty and if it is. Then the address pointing at startReference will have access to the previous pointer and set equal to the beginning node. But if the double linked list is empty then the address pointing at startReference will be set to the beginning pointer.

The second function definition is for displayBubbleSortList which is the function that will print out the nodes in the double linked list. The first thing in the function definition is a node for temporary pointer set to the start which is followed by an end line (std::endl). Then comes the while loop that will keep displaying each element in the double linked list until NULL. Within the while loop are two lines of code that prints out the each element of the double linked list and then gets data on the next element of the double linked list.

The third and final function definition for bubbleSortAlgorithm, which is the function that actually puts the elements/nodes through a bubble sort. It starts off by creating and initializing a node with a beginning pointer. Then create and initialize nodes with a linked list pointers and by default set them to NULL. And an int variable that will carry data under

swapped. After that is an if statement in case the double linked list is empty in which if the condition of the if statement is true then it will terminate the execution of the function. But if the if statement's condition is false then a do while loop will be executed which will repeatedly swap the adjacent elements if they are in the wrong order from smallest to biggest element. This is basically the main component behind the bubble sorting. Now the do while loop starts off by setting swapped int value to zero and the beginning pointer node set to the start pointer node. Then a while loop will run that will swap the elements until the linked list pointer and beginning pointer which has access to the next pointer are equal to each other. Inside the while statement contains an if statement that will swap the elements if the beginning pointer has a data value less than the next pointer data value. Underneath the if statement contains the swap function which will swap the data values and then set int swapped to one. Outside the if statement yet still contained the while loop the beginning pointer will have access to the next pointer. Then finally outside the while statement is a line of code where the linked list pointer and beginning pointer will be set to each other, and update the linkedListPtr.

Finally there's the int main() which starts off by Initializing and creating an empty doubly linked list. Then initialize and create an array that will store the numbers that will go through bubble sort and an int variable named listSize that stores the size of the double linked list. This is followed by a for loop that will keep executing the insertBubbleSortNode function 30 times. And inside the for loop there is a function recall for insertBubbleSortNode which will denote the reference on startNode as and set as parameter, while including the array as the parameter to have each element in the array list go through the insertBubbleSortNode function. Outside the for loop is a cout statement informing the user on the content of the double linked list before

bubble sorting. Then a function recall for displayBubbleSortList which will display the double linked list content before bubble sorting. This is then followed by three more cout statements which are an end line, divider, and another end line. Then another cout statement stating "Calculating Time and Sorting..." which will inform the user that the program is calculating how long it takes to bubble sort the double linked list. Then this is when the stopwatch/ the beginning of recording how long it takes for the bubble sorting algorithm to finish by using a std::chrono::system_clock::now(). Once that line of code is processed the function is recalled on bubbleSortingAlgorithm which will bubble sort the double linked list. Once the function recall is over it will be followed by a divider and another cout statement informing the user on the results of the double linked list after bubble sorting. Then finally the double linked list will be displayed after bubble sorting and will stop the stopwatch and stop recording time length. Once the stopwatch is over it will calculate how long it takes to finish bubble sorting the elements. And inform the user on how long it took to finish bubble sort.

InsertionSort.cpp:

(This file contains the insertion sort algorithm)

First you have the node structure which contains information such as the data on the node, a pointer reference for the next pointer and a pointer reference to the previous pointer. The getNewNode will have the integer stored in newData and create a Node from that data. There is a block of code in getNewNode which states that a new Node will be created in heap memory. And that the new Node will be created by filling in the data field as newData, set the previous pointer as NULL, set the next pointer as NULL. And then finally return a pointer to the new Node. All the information in the LList class is under public access. This is followed by the

function declarations for insertInsertionSortNode, displayInsertionSortList, insertionSortingAlgorithm, and pushNode.

The first function definition is for insertInsertionSortNode which is a function that inserts a node at the double linked list. Within the void insertInsertionSortNode is a variable that will create a node and always insert it into the current node of the double linked list. Then this is followed by a block of code consisting of an if, else if, and else statement. The if statement will check if the double linked list is empty and, if the condition is true then it will set the address of headReference to newNode. The else if statement is for when the node has to be inserted at the beginning of the double linked list. The else if statement will give newNode data access to the next pointer and is set to the address of headReference. newNode will have data access to the next pointer and is set to the nextNode and it will set the address of the headReference to newNode. The else statement is for when the double linked list is not empty and is not going to be inserted at the beginning of the double linked list. In the else statement the currentNode will have the address set to the headReference. A while statement that will locate the node after which the new node is to be inserted which will keep giving the current node access to the next pointer while the while condition is not met. Outside that while loop but still inside the else statement is an if statement for the situation in which the new node has not been inserted which will give newNode access to both the next pointer and previous pointer data.

The second function definition is for displayInsertionSortList which will print out the nodes in the double linked list. This will only consist of a while loop that will keep displaying each element in the double linked list until NULL. And in the while loop are two lines of codes

that will print out the each element of the double linked list and then get data on the next element of the double linked list.

The third function definition is for insertionSortingAlgorithm which is the function responsible for insertion sorting the double linked list. The function definition will start off by creating and initializing nodes as sortedNode pointers and by default set to NULL and creating and initializing nodes as currentNode pointers and by default set to the address of headReference. This is followed by a while loop that will keep traversing the double linked list and insert every sorted node. And inside the while loop consists of a node to store the next element iteration and then removing all the links to create the currentNode as a new node for another element iteration. Then followed by a function recall on insertInsertionSortNode with the parameters of currentNode and the sortedNode that has had it's reference denoted. And then set the currentNode to nextNode which will update the currentNode. Outside of the while loop and the last line of code in this function definition it will set the address of headReference to sortedNode which will update headReference to point to insertion sorted doubly linked list.

The fourth and last function definition is for pushNode which will insert a node at the beginning of the double linked list. This function definition starts off by using malloc to create a Node in heap memory by reserving memory in heap. And allocating memory for the nodes. newNode will also have access to data and then have access to the next pointer and make the next of new node as head, and followed by giving newNode access to the previous pointer and set it as NULL by default. Then there’s an if statement that will change the previous pointer of headReference to newNode. And inside the if statement is a line of code that will have a

headReference point to newNode with what access it got from the previous pointer. The last line of code for this function definition will move the headReference to point to the newNode.

Then finally there is the code that is in the int main() which initialize and create an empty doubly linked list. Then data elements are inserted for the insertion sorting algorithm. A cout statement prints out to inform the user on the content of the double linked list before insertion sorting and, then a function recall for displayInsertionSortList which will display the double linked list content before bubble sorting. This is then followed by an end line, divider, and another end line. Then another cout statement stating “Calculating Time and Sorting...” which will inform the user that the program is calculating how long it takes to insert the double linked list. Then this is when the stopwatch/ the beginning of recording how long it takes for the bubble sorting algorithm to finish by using a std::chrono::system_clock::now(). Once that line of code is processed the function is recalled on insertionSortingAlgorithm which will bubble sort the double linked list. Once the function recall is over it will be followed by a divider and another cout statement informing the user on the results of the double linked list after insertion sorting. Then finally the double linked list will be displayed after insertion sorting and will stop the stopwatch and stop recording time length. Once the stopwatch is over it will calculate how long it takes to finish insertion sorting the elements. And inform the user on how long it took to finish insertion sort.

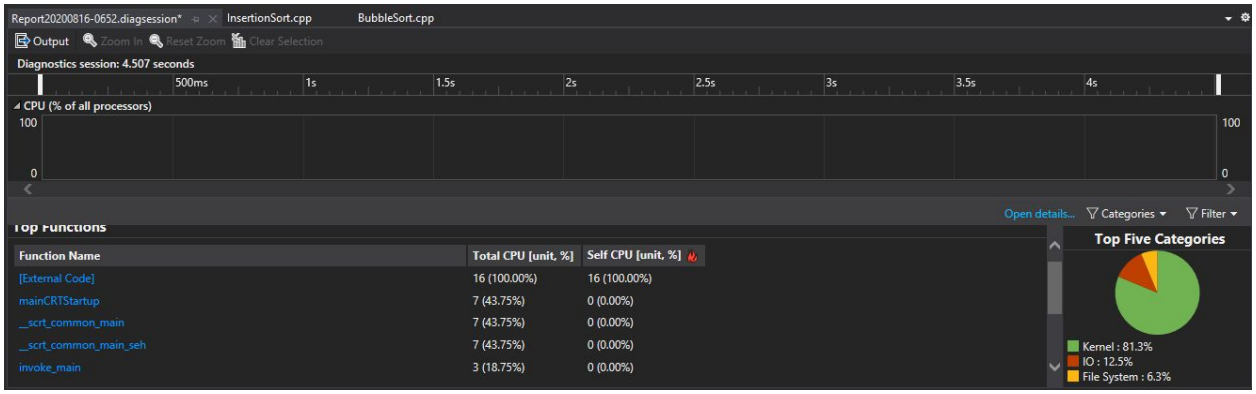
Analysis:

Bubble Sort:

- Worst Case - O(n^2)
- Average Case - O(n^2)

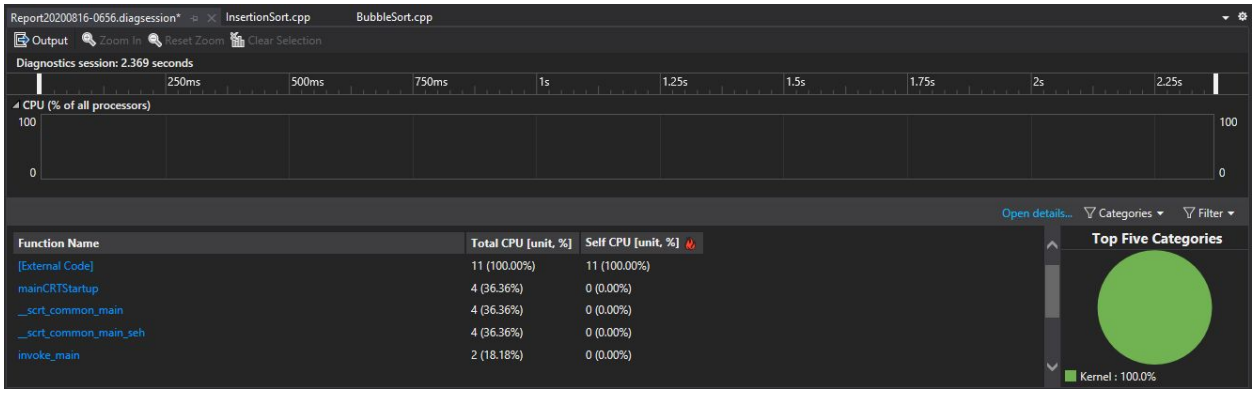
Data Structures and Algorithms: Project Sorting 9

- Best Case - O(n)



Insertion Sort:

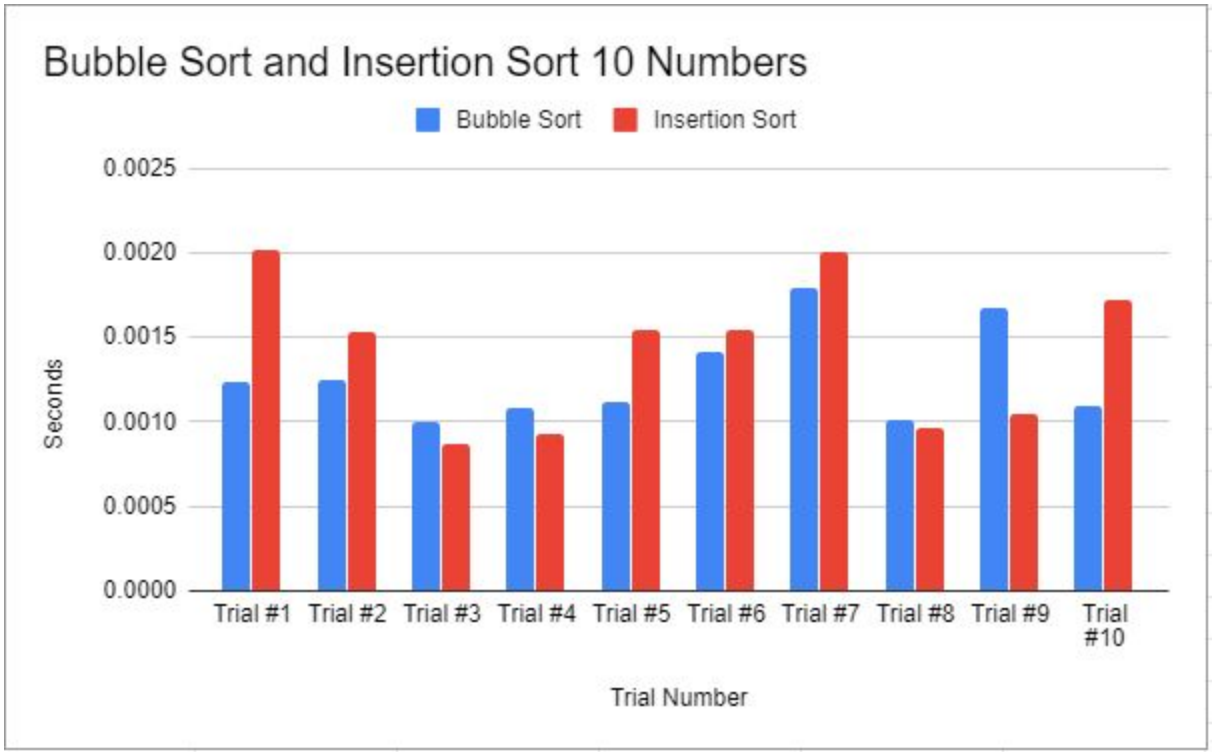
- Worst Case - O(n^2)
- Average Case - O(n^2)
- Best Case - O(n)

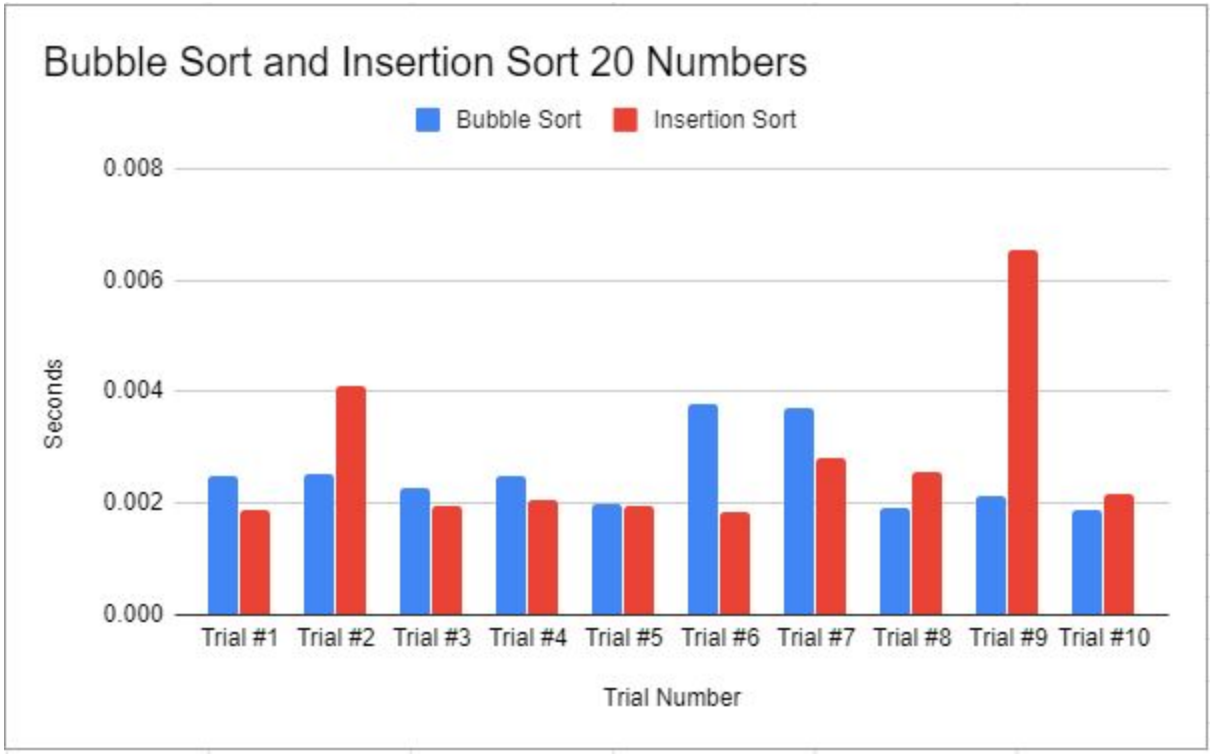


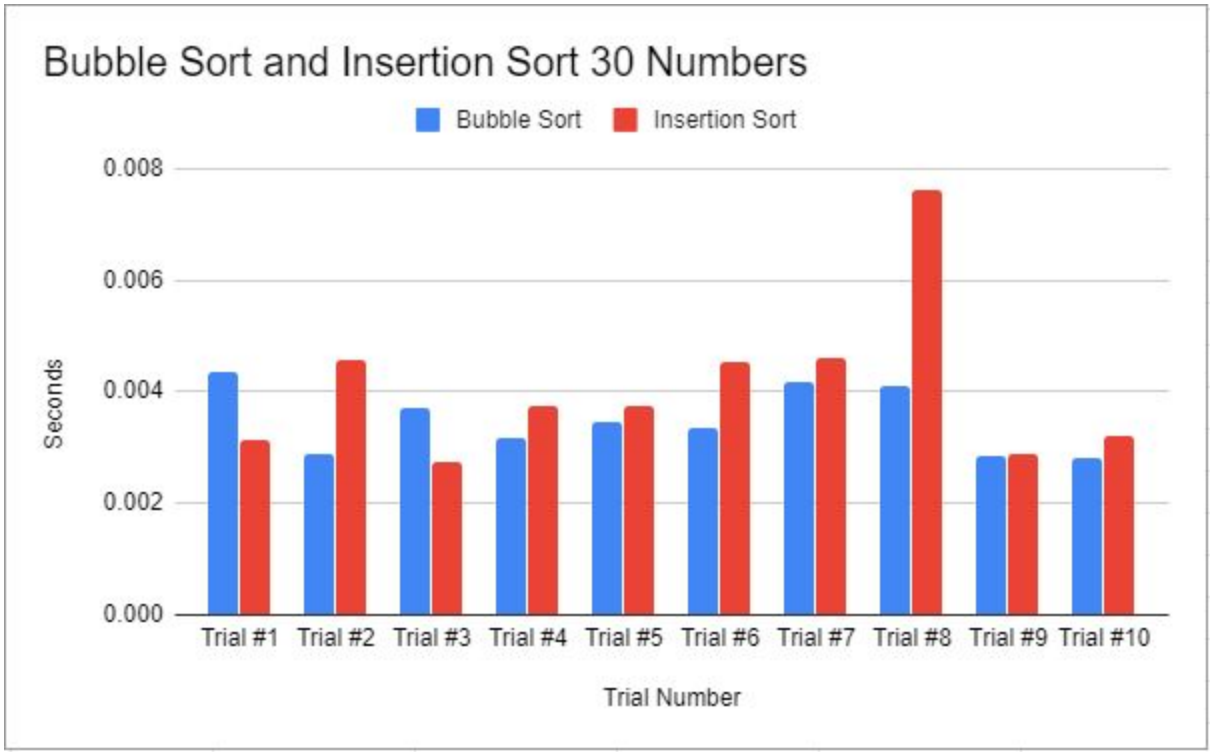
Sorting Algorithms with 10 Numbers											
Algorithm	Trial #1	Trial #2	Trial #3	Trial #4	Trial #5	Trial #6	Trial #7	Trial #8	Trial #9	Trial #10	Average
Bubble Sort	0.0012336	0.001254	0.0010039	0.0010861	0.0011197	0.0014189	0.0017897	0.0010101	0.0016696	0.0010963	0.00126819
Insertion Sort	0.0020181	0.0015382	0.000873	0.0009299	0.00154	0.00154	0.0020098	0.0009673	0.0010494	0.001722	0.00141877

Sorting Algorithms with 20 Numbers											
Algorithm	Trial #1	Trial #2	Trial #3	Trial #4	Trial #5	Trial #6	Trial #7	Trial #8	Trial #9	Trial #10	Average
Bubble Sort	0.0024767	0.0025194	0.0022569	0.0024813	0.0019954	0.0037958	0.003691	0.0019116	0.0021409	0.0018824	0.00251514
Insertion Sort	0.0018722	0.0041154	0.0019392	0.0020645	0.0019406	0.0018369	0.0028009	0.00254543	0.0065567	0.002178	0.002784983

Sorting Algorithms with 30 Numbers											
Algorithm	Trial #1	Trial #2	Trial #3	Trial #4	Trial #5	Trial #6	Trial #7	Trial #8	Trial #9	Trial #10	Average
Bubble Sort	0.004367	0.0028716	0.0037001	0.0031558	0.0034729	0.0033347	0.0041808	0.0041112	0.0028412	0.0028096	0.00348449
Insertion Sort	0.0031202	0.0045619	0.0027545	0.0037339	0.0037515	0.0045196	0.0045918	0.007599	0.0028794	0.003209	0.00407208


Bubble Sort and Insertion Sort 10 Numbers
Bubble Sort
Insertion Sort
0.0025
0.0020
0.0015
0.0010
0.0005
0.0000
Seconds
Trial #1
Trial #2
Trial #3
Trial #4
Trial #5
Trial #6
Trial #7
Trial #8
Trial #9
Trial #10
Trial Number


Bubble Sort and Insertion Sort 20 Numbers
Bubble Sort
Insertion Sort
0.008
0.006
0.004
0.002
0.000
Seconds
Trial #1
Trial #2
Trial #3
Trial #4
Trial #5
Trial #6
Trial #7
Trial #8
Trial #9
Trial #10
Trial Number


Bubble Sort and Insertion Sort 30 Numbers
Bubble Sort
Insertion Sort
0.008
0.006
0.004
0.002
0.000
Seconds
Trial #1
Trial #2
Trial #3
Trial #4
Trial #5
Trial #6
Trial #7
Trial #8
Trial #9
Trial #10
Trial Number

I would say that the run-time complexity would match the measured run-time complexity. As it would seem that the sorting algorithms have their performance correlating to the square of the size of the input data set. And I would say from my experimentation on bubble sorting and insertion sorting that bubble sort performed better. The reason for this is because even though both perform good under small data sets due to their simplicity. As simple algorithms will do less calculations than complex algorithms. And bubble sort is a more simple sorting algorithm than the insertion sorting algorithm. Which is why the bubble sorting algorithm would probably perform the best under this specific case. Now there is the quicksort sorting algorithm which has been considered the best sorting algorithm but isn't perfect. So even though quicksort does not use additional storage space to perform sorting, Efficient memory usage, and is a very fast sorting algorithm. Under my particular case bubble sort would perform better than the quicksort due to quicksort being more complex in its calculation and require more resources than the bubble sort. Which is something that isn't optimal for a small data set.