Project: Sorting Documentation

This project is a sorting algorithm that allows the user to do sort through a list. The main point of the project is to create a blank list that the user can then add, show and sort the nodes on. The user can also choose to run a premade list of commands for a premade list to sort through. This allows the user to customize the list and then test to see if it works. There is also a menu that allows the user to chose what option that they want to take.

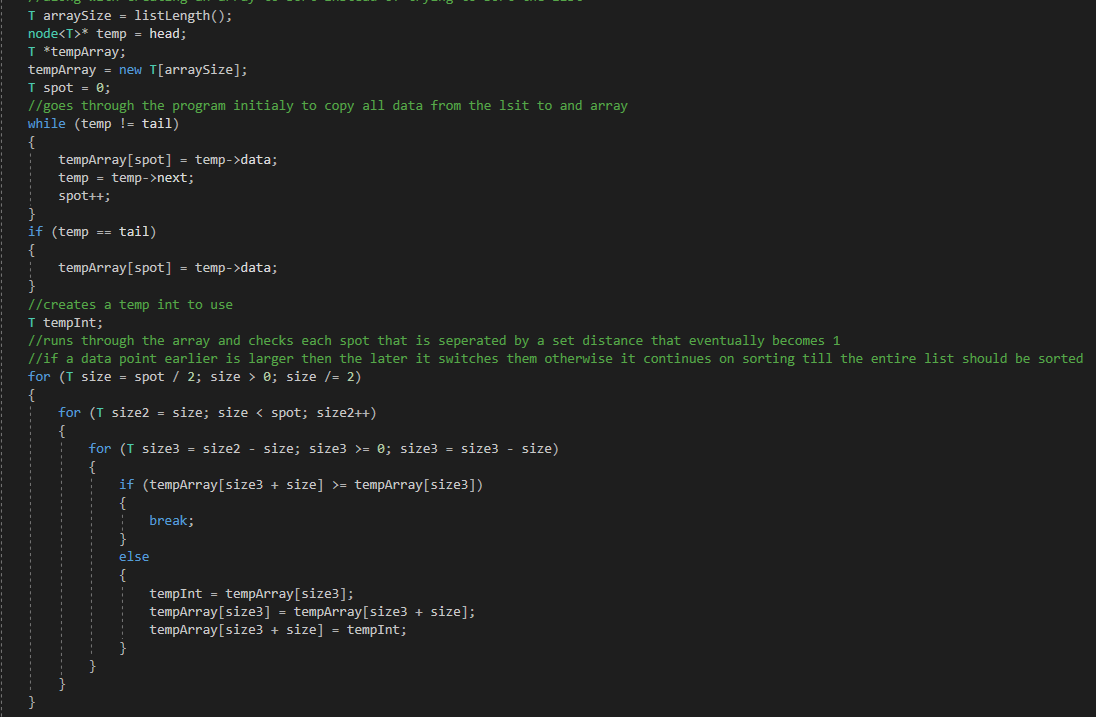
The program starts off with the customary visual studio opening includes of iostream, and time.h to allow the use of measuring the runtime of the sorting algorithms and using namespace std. After that it makes a struct node that encases the data, and next/prev node functions. After that is a sortingList class that has the head/tail under private. In the public of the sortingList it has multiple functions. The first makes the head and tail a nullptr. The next is a void that takes data that has been input and then runs it through a temporary node to implement into the list at the end. The next function is the show that takes the entirety of the sortingList as long as there is a list to go through and runs it through a temp node to print them on to the screen. The last function is one that goes through the list and calculates the length of it based on the number of nodes the list has for when using functions that need it such as in the shell sort function.

The next function is the first of the sorting algorithms. It starts by checking to see if there is a list to sort through. If there is it enables the clock function and takes note of what the time is when the sorting started. It then has a break point that can be looped to and then it sets a temp node to the head and a temp2 for data holding purposes. Then it goes through the program once for starters in order to check to see if the data on the current node is greater or lesser then the one next to it. If it is it takes the current node data and saves it in temp 2 and then makes the current node data equal that of the next nodes data. After that it makes the next node data equal to the temp2 data. Once it runs through the list it checks to see if the list still is not sorted. If it is not it loops to the break point and then runs through the entire process again. Otherwise it displays the time it took to sort the list and then if there are less than 100 nodes in the list it will display them to the screen.

The second to last function is the shell sort function. This one starts off by checking to see if there is a list to sort through. If there is it enables the clock function and then keeps track of the time for use later. Then it creates an array that gets its size from the last function. Then it goes through the list/array and copies the data from the list into the correct position for the data in the array. After that it then creates a tempInt for future use. The main bit of sorting it takes the size of the array and divides that in half then as long as it is not 0 it continues to divide in half till done. Inside the first function is for a second size (size2) that is the same as size and if that is less then the size of the array adds one to it every iteration. That has a function that then creates a third size (size 3) that is equal to size2 minus size and if it is not less than zero will continue to loop and size 3 will equal size3 minus size each loop. It then checks to see if the spot on the array pointed at by size 3 plus size is greater than the spot pointed at by size. If it is the program will then exchange their data using the tempInt and have that hold one of the arrays data points. Then it will have the data point that was held be equal to the other data point. After that the data point that was not held will be set equal to tempInt data thus switching the data in those two spots on the array. Otherwise the program will continue to search the list till the entire thing is sorted. Then it creates another temp node and data holder. The node is set to head and then it will run through the list setting that data points to what the array is holding in the same spot thus creating the sorted list. After that it will display how long it took to sort the list using the clock and if the list has less then 100 items it will show the list.

In the cpp file is the main function that has variables for inputted data and a new list for the user. It also welcomes the user and has a checkpoint that can be looped back to. It then calls the user command function that gives the choices in the main menu. After that it asks for a number input and then asks for input/runs a function depending on the number given. Choices are added to the list, show the entire list, sort the list using bubble sort algorithm, sort the list using shell sort algorithm, run the automated test function for the bubble sort algorithm, run the automated test function for the shell sort algorithm or exit from the system. The other main function is the two test cases that run through a set if predetermined functions and values and then sorts the lists using one of the two algorithms.

Example for shell sort case due to having to run it through an array instead of a list



**Analysis**

Worst-Case Time Complexity

Bubble Sort WCTC – O(n2)

Shell Sort WCTC – O(n^3)

The bubble sort runs best when it is run through small lists because it checks each point on the list each time it runs through the list in order to make sure it is sorted correctly otherwise it starts over again and must run through each one. This means that the larger the list is the slower the sorting algorithm will run due to having to check every node.

The shell sort runs best when there is both a moderate amount of data and even amount of data. This is due to the sorting algorithm taking the data and looking at two points in different places on the list/array and then seeing if they are in the correct place and then the distance between the points being looked at slowly grows smaller till the entire list is sorted.

Data Sets

|  |  |  |
| --- | --- | --- |
| Anaylsis Data | Bubble Sorting | Shell Sorting |
| 10 | 0.0002 | 0.0005 |
| 50 | 0.001 | 0.0025 |
| 100 | 0.002 | 0.005 |
| 200 | 0.004 | 0.01 |
| 500 | 0.01 | 0.025 |
| 1000 | 0.02 | 0.05 |