Katerina Betts and Donald Hutchinson

Professor Charles Zimmer

Data Structures

6 March 2018

Lab 7: Pointers and Ordered Lists

Task One and Task Two did not contain ay parts that need to be contained in the lab report.

Task Three:

The first version of the class that was created to search from the beginning of the list was trying to implement a basic list that stores items in ascending order without any attempt to improve efficiency. This is an easy to understand list that a user would be able to implement without having to think in any way. This is convenient if the item being searched for is in the beginning of the list; however, it would be inconvenient if the item being search for was at the very end of the list. Without testing these cases, I believe this case might be the best case to go through a list and will have the best performance.

The second version of the class that was created to search from the end of the list was trying to implement a list that was also in ascending order, yet it searched from the back of the list. This would be a strong list to use if searching for a large number or a number that is at the end of the list; however, it would be inconvenient if the item being searched for was at the front of the list. I believe this one will have the second best performance, but I believe that it would be relatively close to the performance of the front searching list because it is not that much different in the code.

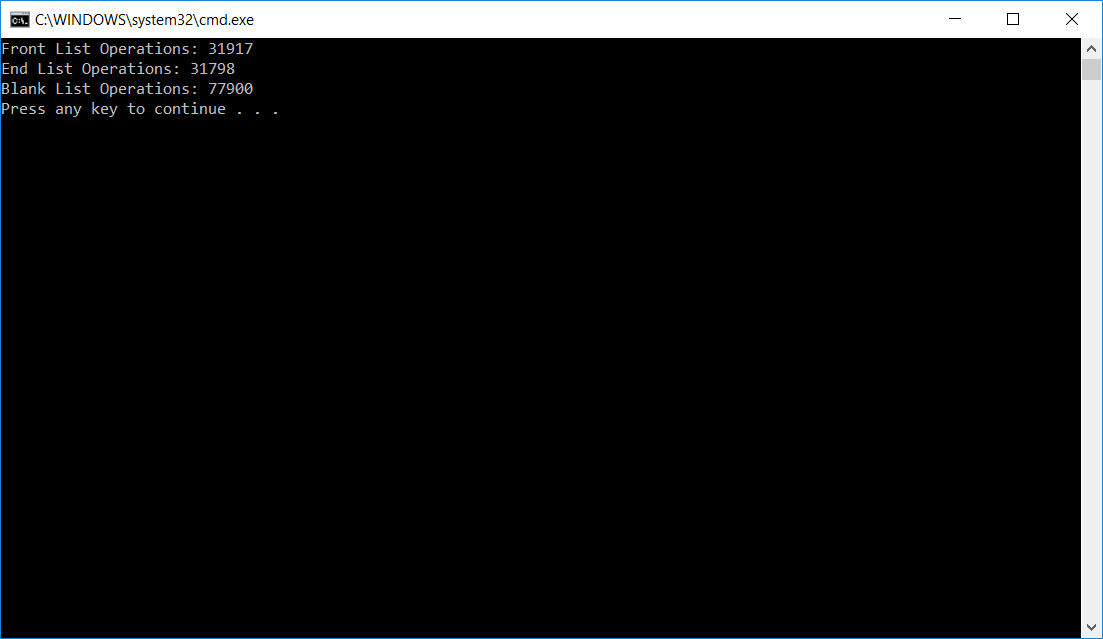
The third version of the class that was created to check how leaving blank spots in the array can reduce the number of moves when inserting more items into the list. The positive aspect of this part of the list is that there is no shifting up and down the list as the item is inserted or removed from the list. The biggest weakness that I see when creating this class is that any items are inserted halfway between the higher and lower items, so the program must search for both these items. This is why I believe that this case will do the worst of the three.

Task Four:

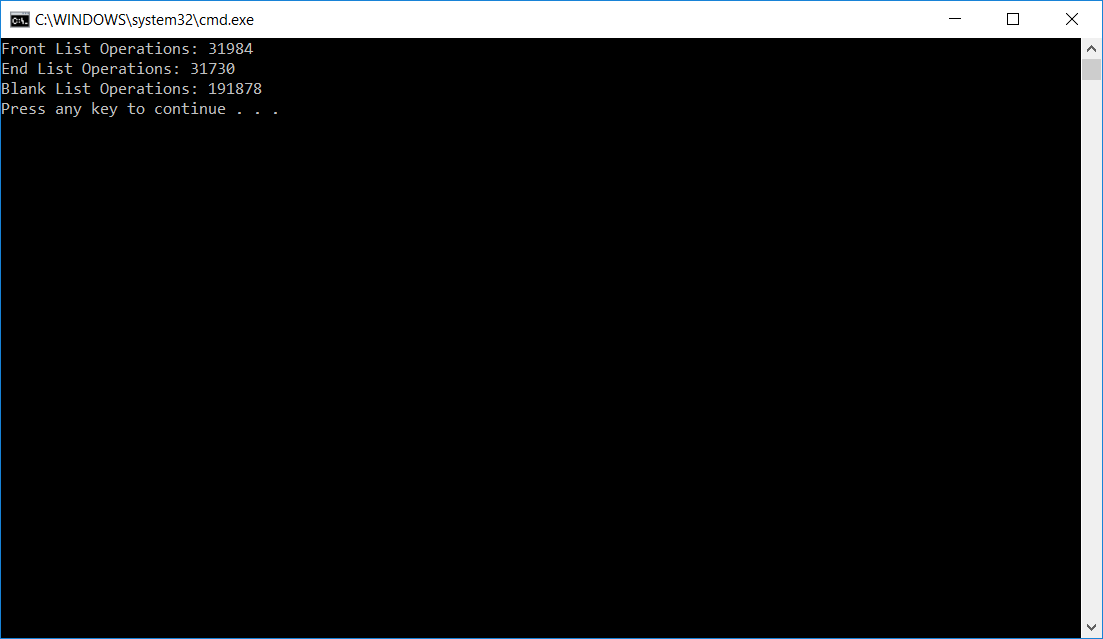
Of the 100 tests that have been run of each of the three classes, the front and end of the list performed similarly with the end of the list doing mildly better than the front of the list. The blanks test, however, did the worst of the three test cases. This was close to the predicted outcome of the three cases that was mentioned earlier; however, the end of the list class worked slightly better than the beginning of the list class.

The results when the list size was increase to 50 was the same as the list size at 20. The end of the list class did the best with the front of the list class close behind, and the insertion list was also significantly worse than the other two lists. This time, the insertion list was even worse than the initial test run with 20.

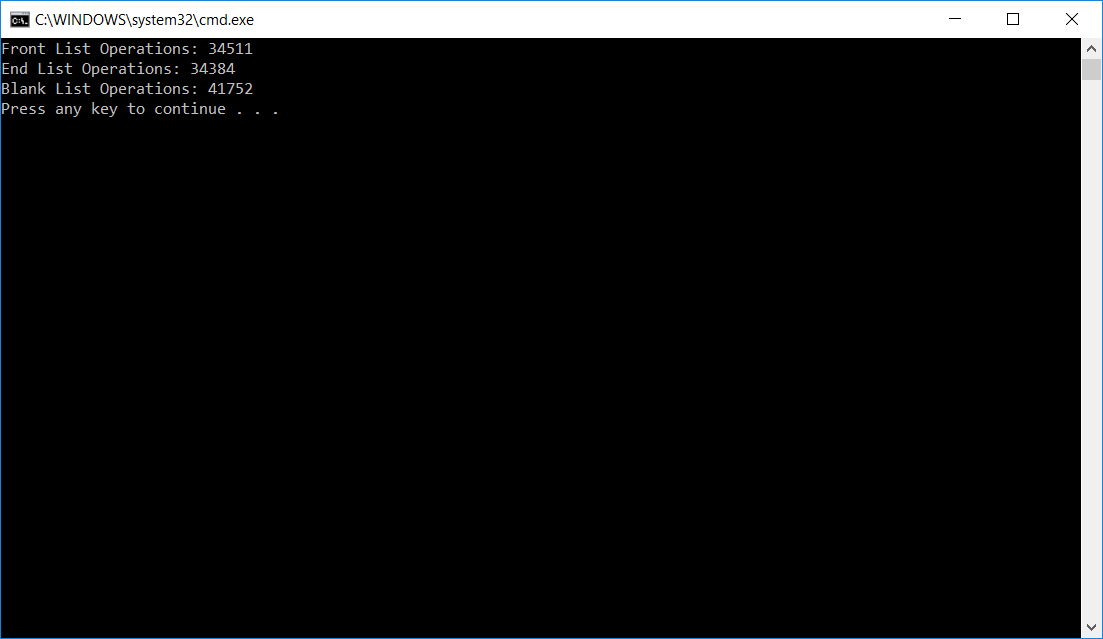
The test with ten instead of twenty or fifty also followed this same trend as the other two tests. The end list class did the best while the front of the list followed behind. How poorly the insertion list did was less noticeable in this test run however. It performed better as the numbers (50, 20, 10) got smaller.

SCREEN SHOTS HERE 

This is the case where the original 20 item list.



This is the figure for the 50 item list test case.



This is the 10 item list test case.

When it came to simplifying running and reporting 100 results, a loop was used to test the class multiple times with each changed list size. The number of operations performed by each list was summed and displayed after running 100 tests.

The results were analyzed by incrementing an operation counter for the lists each time they compared or moved values. This is a valid way of analyzing efficiency because these operations take the most amount of time. Since these operations were all summed up 100 times, the final output is a good measure of average efficiency.

Another method that was considered at first by group members was running each test case individually 100 times. This was not a good strategy as it was time consuming, and it was not the easiest way to go about getting these test cases.

Contribution:

Both team members contributed an equal amount to the project as one partner did half and other also did half of the lab.

Instructions:

* The program was run and compiled in Visual Studio.
* Make sure all the required files, all of those included in the zipped folder being turned in with this lab, are present and run Task4.cpp.