CS 2028C - Data Structures Lab 7
Wednesday lab, Group 6
Rajdeep Bandopadhyay
Yulia Martinez
Sarah George

Compiling Instructions: Make sure all files are within the folder, including a.txt. For the implementation code of of the templated functions for the stack and queue, we used .tpp files instead of .cpp files. If running in visual studio, this may result in linker errors. The code was tested using g++ in a mac terminal, and outputs of tests have been provided below. If running on a mac terminal, running the following commands, one at a time, from within the lab folder:

```
g++ *.cpp
./a.out
```

1. The objective of the lab was to manipulate an ordered list and experiment with possible efficiency options in creating the list. We were to create an ordered list class that was a template which would contain 25 items. The class would have functions such as add and remove as well as functions to check if the list was empty or full. This would allow us to construct our own lists and fully understand how to best implement the functions. The second task was to create a derived class from the first template class. This would essentially be the same as the first however the remove and insert functions would start at the middle of the ordered list rather than at the beginning. This would also require us to reconfigure the list by shifting it in order to accommodate for the inserted item. The third task is to create a second and final derived class that would search for pointers to null inside the array and determine if having blank spots would reduce the number of moves when inserting elements. Creating ordered lists and sorting algorithms are a crucial part of Computer Science since searching for efficiency is a large part of programming. You don't want to just have a program that works you want to have a program that is efficient with solving the problem. This is what is investigated in task 4. We are to create a test program that will run all 3 classes and see which of the classes is the most efficient ordered list. We want to learn how to efficiently optimize our code and test different ways in solving a problem in order to find the best way that the solution can be achieved.

2. Include in your lab submission a paragraph for each of the 3 versions of the class a description of what is trying to be achieved, what the strengths and weaknesses are and how you think it will perform.

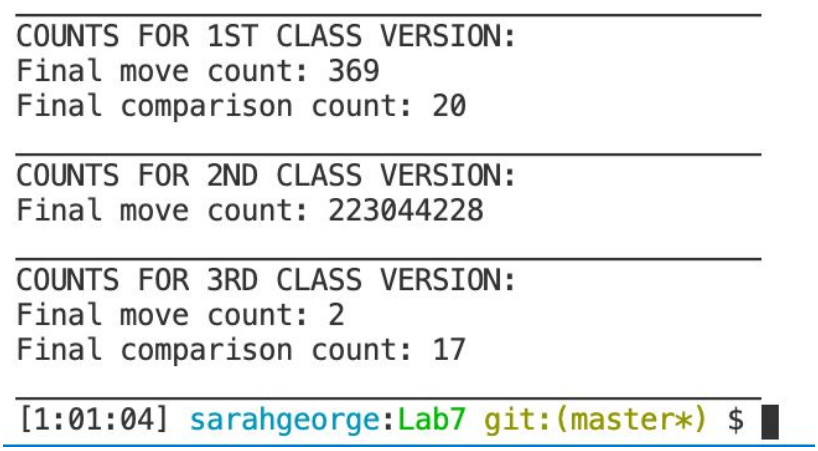
    a. Version 1:
        i. This version of an ordered list is starting at the beginning and searching for the correct spot to place an element or searching for the element that is being removed. It would go through the array and check each element, comparing the new element to the ones in the list. The remove function

also starts at the beginning and would search for the element that is going to be removed. The remove function would also make sure that the array is shifted so that there are no empty spots within the list. The strengths of this class is the simplicity. Because it is so easy to understand it is easy to debug. The weakness is that it has to iterate through all of the elements searching for the one it is looking for. I believe it will perform the best out of the other classes.

b. Version 2:
    i. The derived class version of the ordered list has the same fundamental components. However the add item function will start from the middle of the array when it searches for the correct spot to place the element. The purpose for this version of the ordered list is to test the theory that it would be more efficient since less elements would have to be shifted when inserting a new item. By experimenting with different ways of sorting, we are trying to achieve the most efficient way to create an ordered list. The strengths of this version, starting from the middle, are that if the element needs placed in the second half then it is quicker than the parent class. However, the weakness with this version is that if the element needs to be placed in the first half it would have to loop around meaning it would take the same amount of time as the parent class. I think this class will perform the same as the parent class.

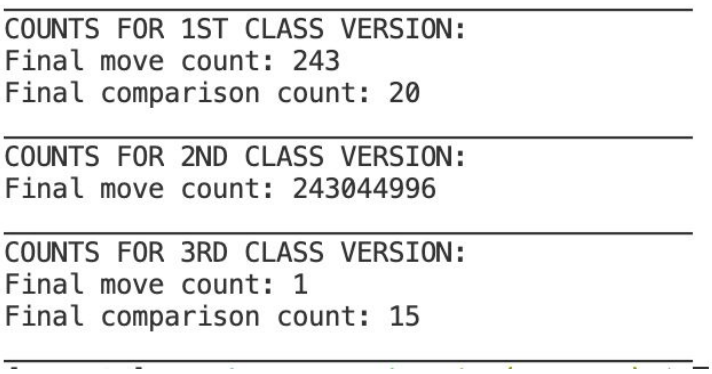
c. Version 3:
    i. The third and final version of the class has the most significant change from the parent class. It includes a modification to the add item function that would insert the item and push the surrounding elements to the sides if there are surrounding elements. The remove function would also not move any items in the array. This means that the remove function is simply deleting the data being pointed at. This version of the derived class is trying to make the least amount of moves possible when inserting an element into the ordered list. The strengths of this version of the class are the remove item would not move the items when removing an element. The weaknesses are that if the item needs to be inserted between two elements then the outer elements would also be shifted. However, because the moves only happen if the inserting item sits between two items that are in contiguous locations.

3. Testing Results:
    a. Running 100 times:



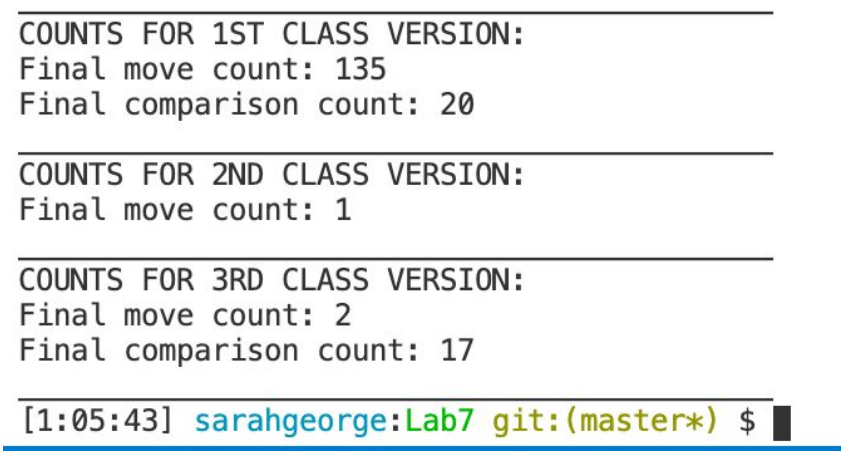
The results of the first test were not very different from what we had predicted. The test revealed that the third class was the most efficient and the second class was the least efficient. We had predicted that the first class and the second would have about the same efficiency however the test resulted with the second class having an extremely large amount of moves.

    b. Running 100 times (Size = 50):



The results of the second test were similar to the results of the first. The difference between both tests was that the size of the array was increased to 50 items from 25. The second class resulted with the most amount of moves meaning it was the least efficient. The first class was more efficient by a large amount. However the 3rd class was the most efficient with a shockingly large difference between it and the second class.

c. Running 100 times (Size = 10)



The last test for the classes required the array size to be changed to 10. This meant that there would be the least amount of moves total compared to both the previous tests. The results of this test were very different from the previous results. This test resulted with the first class being the least efficient and the second class being the most.

4. Discussion
    a. In order to simplify the program to make it run 100 times with the different parameters i.e array size, we used three different loops for the three different tests. The program would compute all three tests at once. Instead of user input for inserting and removing items we used a random number generator for a number between one and ten. This meant that the array would be populated with random numbers, and then we would remove and insert numbers that were also generated randomly. The results would be consolidated to one output that counted the total number of moves within each loop for each test. The results were analyzed using the total number of moves within each test and comparing them to each other. This method allowed us to see the efficiency in the classes because we were counting how many total moves each class was taking with the same parameters.