**Data Structures Lab Report 7**

Group Members: Vivek Kunapareddy, Yuan Cheng

**Discussion of Objectives/Concepts:**

The objectives explored in this lab are creation of ordered lists and multiple ways to increase efficiency in adding and removing items. These concepts are helpful in transitioning into the workforce as it shows us how underlying data structures work and how to implement them.

**Discussion on how the ordered list implementations differ:**

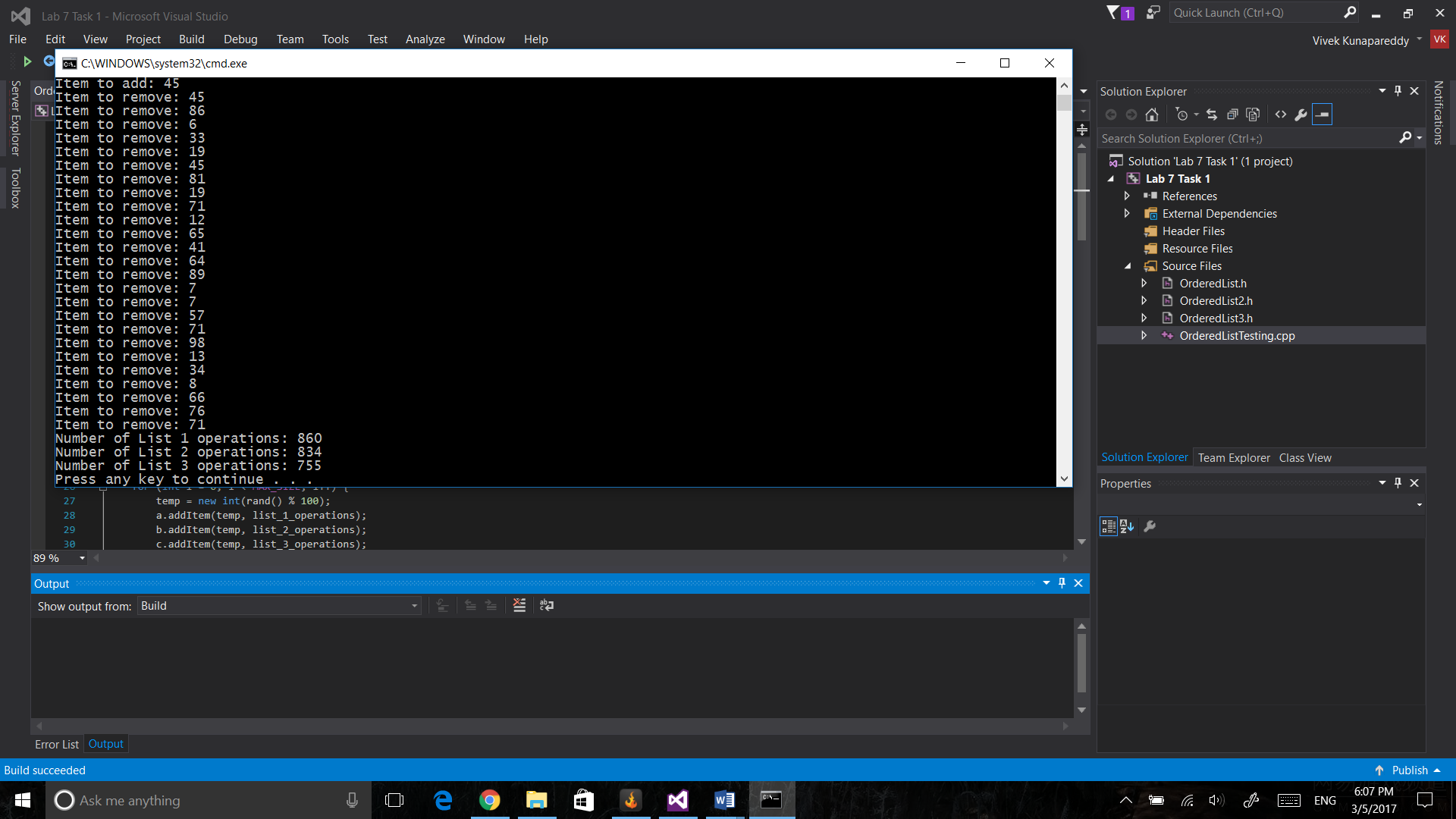
**Implementation 1:** This is a classic implementation of an ordered list. It’s simple to understand the implementation as the program flow is extremely linear. However, the number of operations in the addItem and removeItem are always consistently in the terms of O(n)

**Implementation 2:** This implementation is also very like the classic ordered list. However, the only difference lies in the addItem which tracks the underlying array from the end of it as compared to from the start of it. If used in lists which tend to grow in size and have operations centered around the end of the list, it reduces the number of operations by a large amount.

**Implementation 3:** This implementation is an extremely nuanced version of ordered list which has a massive reduction in operations. Since ordered lists use add and remove items frequently, this implementation keeps track of the empty spaces by not moving items unless absolutely needed.

**Screenshots of results:**

Result of inserting 30 items and removing 25 items in a MAX\_SIZE 30 list:

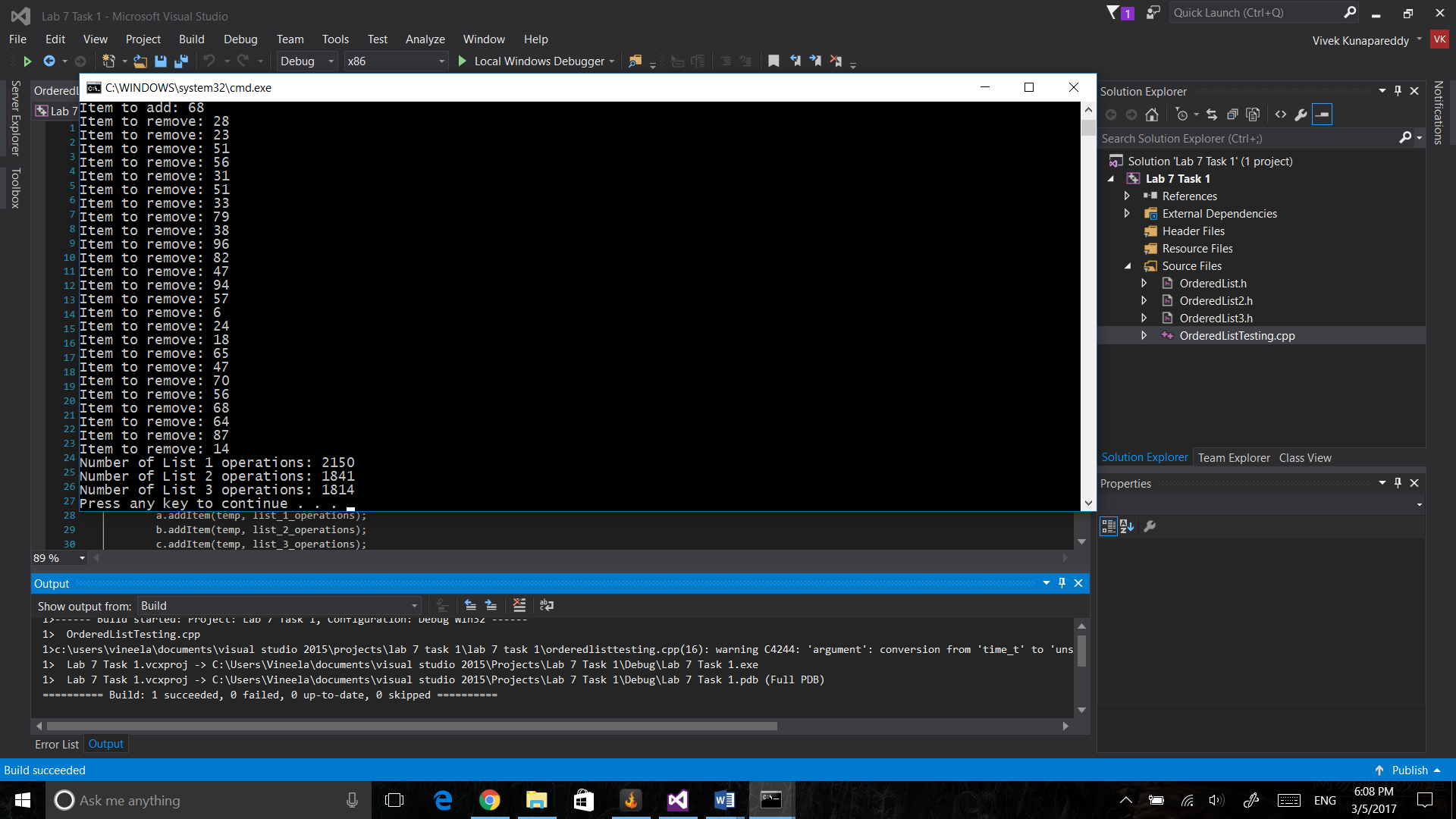


Discussion:

As expected the results match the predictions. The second list implementation beats the first one very narrowly as it’s a medium sized array.

The third type beats out the other two implementations massively due to reduced operations

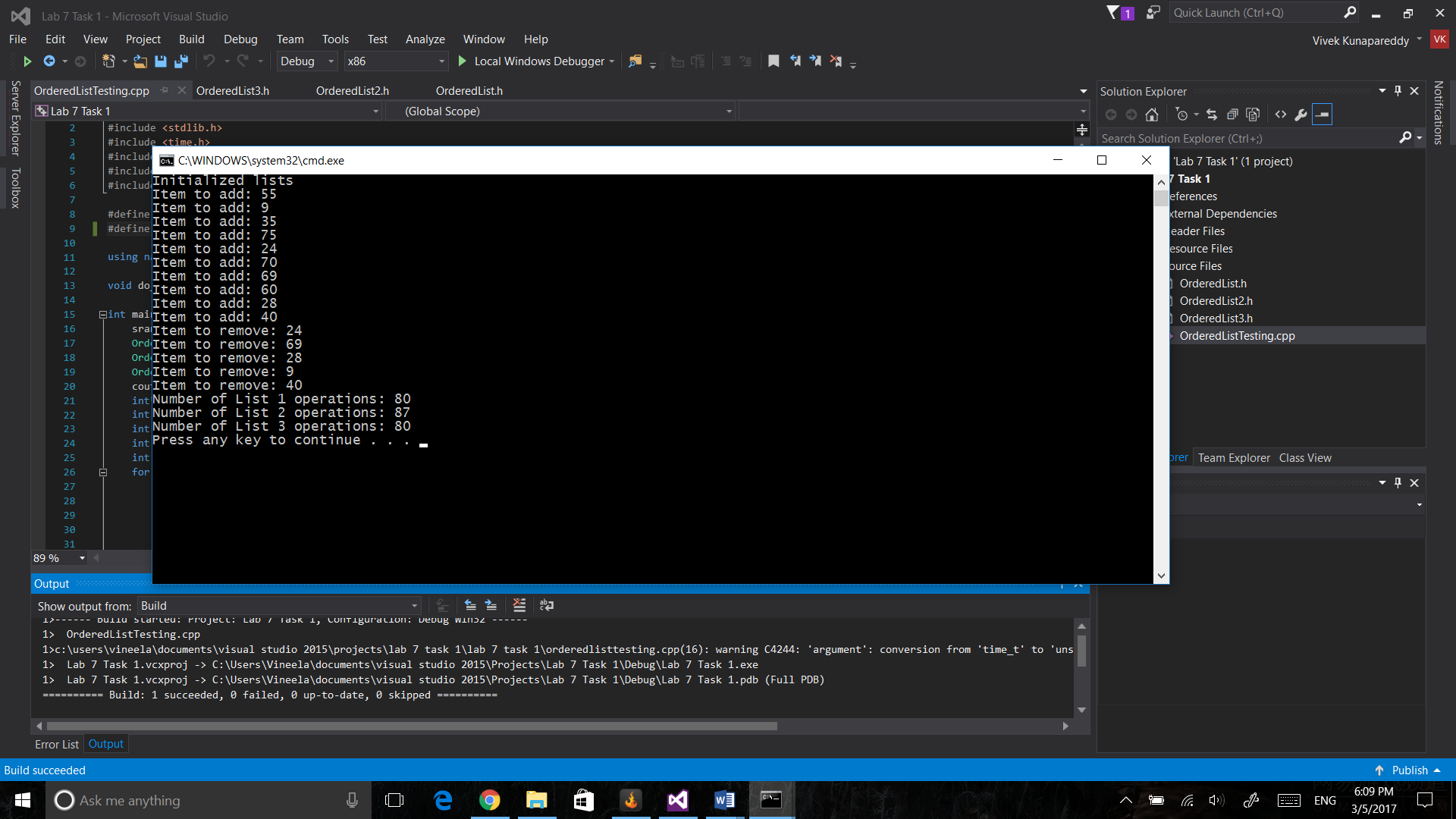
Result of inserting 30 items and removing 25 items in a MAX\_SIZE 50 list:



Discussion:

Since the size of the array increases the difference in the total number of operations also increases. However it still manages to match predictions

Result of inserting 10 items and removing 5 items in a MAX\_SIZE 10 list:



Since the size is extremely small compared to previous iterations, the number of operations is almost similar or equal in implementation 1 and 3. However implementation 2 lags behind due to there not being enough size to exercise its capabilities.

**Discussion of methods for analyzing results:**

To analyze the results, we have received, we compared the total number of operations and compared it with the size of the underlying list.

If the list is small, all implementations have a similar number of operations as there is not enough data samples for it to truly converge to a point where we can draw conclusions. This is what happened in the case of the size 10 array.

In the other two tests, however, we can see the rapidly growing difference between the implementations. As the first implementation realizes its lack of efficiency, the other two slowly gain the edge as compared to it.

Finally, the implementation 3, due to reducing number of operations in removeItem shows the vast difference in number of operations when the MAX\_SIZE of the array is 50.

**Group Contributions:**

All the programming was done together.

**Compilation Instructions:**

The lab seems to fail at runtime when compiled few times. In order to get past this, just rerun the program.