Evan Akers

Smit Patel

Kyle O’Connor

Saylee Dharne

Lab 7

Efficiency is vital to large scale programming because the speed of the program is the difference between waiting to search and searching instantly and in some cases life and death. This is obviously important because making an efficient program helps shorten user time with the program. Using arrays is not the best way at all to look at efficiency because linked lists, vectors, and other data types make much more sense for this use case.

**Task 3**

In the first version of the class, a generic version of an ordered list is trying to be achieved. This ordered list should not have any empty spaces between numbers. Any empty spaces in the list will be located near the end of the list. I think the strengths of this class are that the empty spaces are in known positions, so it is easy to find the empty spaces while moving items in the array. A weakness is that this class must move nearly every item in the list, if an item is inserted in the first position. I think it will perform well if all the numbers added or removed are stored towards the end of the array. It will be more inefficient if the numbers are smaller and closer to the beginning.

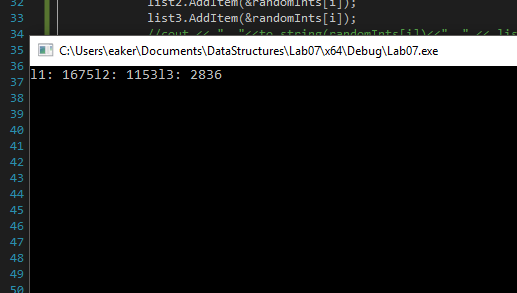
In the second version of the class, a generic version of an ordered list is trying to be achieved. This ordered list should not have any empty spaces between numbers. Any empty spaces in the list will be located near the end of the list. The difference between this ordered list and the first is that the AddItem function for this list starts looking for the item from the end of the list. A strength of this is once again, the knowledge of where the empty spaces in the array and inserting numbers which should be near the end of the list would be easier to find. This method is once again inefficient for the inserting and removing items near the beginning of the list.

In the third version of the class, an ordered list which allows spaces between the items in the list is created. It is expected that the functions for add and remove methods will insert items in the list, leaving in the spaces. One strength is that there will be very few items moved in the list. A weakness is that this class has more comparisons in the function for add and remove.

**Task 4**

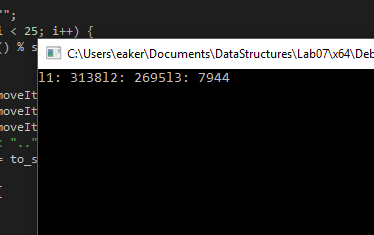
Size 30 averages: l1: 1675 l2: 1153 l3: 2836

Results are consistent with our predictions



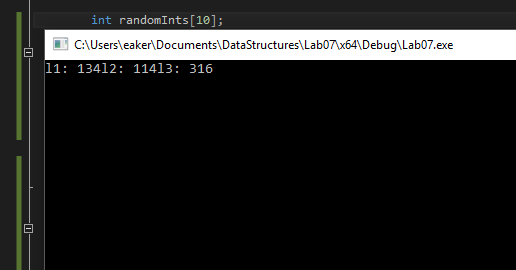
50 l1: 3144 l2: 2699 l3: 7945

There are more operations because there are more values



10 l1: 134 l2: 114 l3: 316

There are many less operations because there are many less values



Because it wasn’t required, we just put the average number of operations for each list type as the output. These findings are consistent with our predictions from lab 3, as the binary search uses many more operations but less moves, and the first two being almost identical.

Compilation Instructions

This has been tested by creating a new project within Visual Studios with the following options:

Win32 Console Application

Create directory for solution OFF

Empty project ON

Precompiled header OFF

SDL OFF

Then:

Add the following files to the projext:

1. OrderedList.h
2. Lab7-Task4-TestProgram.cpp

Build and run

Contribution of Team Members

Each person wrote the portion of the lab report for their task,

Kyle O’Connor did task 1

Smit Patel did task 2

Saylee Dharne, Kyle O’Connor, and Evan Akers did task 3

Evan Akers did task 4