Lab 7 – Pointers and Ordered Lists

Ember Roberts, Maggie Lyon, and Nyla Spencer

The concepts explored in this assignment include pointers and ordered lists. These concepts are important when studying computer science/engineering since they are core concepts. Pointers allow for dynamic memory allocation, efficient memory management, and are fundamental in the implementation of data structures where nodes often reference other nodes. Ordered Lists allow for searching and sorting. They help organize data in a way that is efficient in accessing and manipulation.

In task one, an ordered list template class is created using an array of pointers. This class has an array that holds 25 items. This class includes methods such as AddItem, RemoveItem, IsEmpty, IsFull, and MakeEmpty. Also, any error conditions encountered throws an exception. The strengths and weaknesses of this class are

In task two, a derived class is created from task one. This version has a modified AddItem method. This method begins its search from the insertion point of a new item from the middle of the current collection each time. The RemoveItem method is also modified, it starts from the middle of the current collection when searching for a place to remove the mentioned item each time. Every comparison and move operation is counted/incremented. The strengths and weaknesses of this class are

In task three, another derived class is created from task one. This class will leave blank spots to reduce the number of moves when inserted. The modified AddItem method and RemoveItem. The AddItem method is modified to insert an item halfway between any two items in the array where it belongs. It only moves items in the array if the inserting item sits between two items that are in contiguous locations. The RemoveItem is modified so that it will not move any items in the array. Instead, it will just assign NULL value to the spot. The strengths and weaknesses of this class are

Task four was the creation of the test program. This Lab also included an automated test. This test instantiates each of the three classes and inserts random items. It then compares the performance of each class.

```
Press 1 for user test
Press 2 for automated test
Press 3 to quit
2
Orignial Ordered List:
10 13 17 24 28 30 31 36 43 50 58 66 74 88 100 x x x x x x x x x x
Comparisons: 162
Moves: 71
Binary Sorted List:
10 13 17 24 28 30 31 36 43 50 58 66 74 88 100 x x x x x x x x x x
Comparisons: 84
Moves: 82
Blank Space Sorted List:
10 13 17 24 28 30 31 36 43 50 x x 58 x x 66 x x x x x 74 x x 100
Comparisons: 58
Moves: 57
```

Shown above is the output of the Automated test when run once. The same random elements were either added or removed from each list 30 times.

```
Press 1 for user test
Press 2 for automated test
Press 3 to quit
2
Averages over 100 trials:
Regular ordered list:
Comparisons: 166
Moves: 60
Binary sorted list:
Comparisons: 82
Moves: 73
Blank Space Sorted list:
Comparisons: 42
Moves: 47
```

The code was then altered to show the amount of comparisons and moves made on average through 100 trials. This was done by placing the logic to add and remove 30 random elements within a for loop that counts to 100, and adding to the total number of comparisons and moves for each method each time it iterates. Then, these variables are printed when the loop terminates.

```
Averages over 100 trials:
Regular ordered list:
Comparisons: 1065
Moves: 467
Binary sorted list:
Comparisons: 313
Moves: 505
Blank Space Sorted list:
Comparisons: 380
Moves: 407
```

Shown above With the number of elements in the array upped to 75 instead of 25, and the number of operations changed from 30 to 80, the number of comparisons and moves made drastically increases. Following the trend of the first trial, the regular sorted list makes the most amount of comparisons, but less moves than hte binary sorted lists. The blank space sorted lists does the least amount of comparisons,

and also less moves than the regular sorted list.

```
Averages over 100 trials:
Regular ordered list:
Comparisons: 42
Moves: 13
Binary sorted list:
Comparisons: 28
Moves: 18
Blank Space Sorted list:
Comparisons: 14
Moves: 11
```

With only 10 elements in the array, and 15 operations being performed each iteration of the 100 trial loop, we see that both the binary sorted list and hte blank space sorted lists do less moves and less comparisons than the regular ordered list.

Across all tests, it is clear that the blank space sorted list requires the least number of moves and comparisons, especially as the number of values in the list increases. The regular ordererd list requires the greatest number of moves, making it the least efficient. Thus, we can assume that the blank space sorted list is the most efficient of the three data structures. The trade-off to using this type of data structure versus the regular sorted list comes from is difficulty in programing, and its potential for data leaks. If a non-null value is accidentally pushed off the end of the list, for example, you might have system memory issues.

Contributions:

Nyla – Task 2, lab report

Maggie – Task 1, Task 4, lab report

Ember – Task 3