Data Structures Final Project – Fall 2017

Implementation of Ordered Integer Lists for IntLinkedBag and IntArrayBag

Fiona Liu & Varun Garg

Table of Contents

[Objective 2](#_Toc500872796)

[Procedure 3](#_Toc500872797)

[Results 3](#_Toc500872798)

[Theoretical Expectations 3](#_Toc500872799)

[Discussion 3](#_Toc500872800)

[Contributions 6](#_Toc500872801)

[Appendix 6](#_Toc500872802)

# Objective

The goal of this report is to articulate the run time analysis of the add, addAll, and remove methods for the IntArrayBag and IntLinkedBag. The IntArrayBag class uses the array data structure, a fundamental data structure that is easy to program and use. There are benefits of using an array, namely its speed because of its cache locality. However, there are disadvantages as well. The size of an array is fixed meaning an allotted memory is created but seldom reached. Secondly, it can be costly to insert new elements because room has to be created for the new elements and for each existing element to be shifted. Linked lists serve to resolve these issues by being dynamic in size and having an ease of insertion and deletion.

This project will discuss the run time analysis of adding and removing values to a linked list and array list in sequential order. It will also discuss the analysis of adding and removing values to a linked list and array list while using the merge sort algorithm.

# Procedure

The procedural steps are similar for both IntLinkedList and IntArrayBag.

First, the add method is rewritten to include sorting the target element in ascending order.   
Then, the addAll method is modified to implement the merge sort algorithm.

Lastly, the remove method is preserved from the initial assignment.

To implement a run-time analysis, a main method was created in the IntLinkedList and IntArrayBag class. To preserve the integrity of processing speed, the simulations for both classes were performed on the same computer. The run times of each method are then recorded and graphed with respect to its add, remove, and addAll implementation of a bag of length N. In our demonstration, the N value is 5000.

# Theoretical Expectations

It is anticipated that ArrayList will be faster than IntLinkedList because an array list has direct references to every element in the list, so it can traverse to the n-th element in constant time. LinkedList has to traverse the list from the beginning to get to the n-th element, making it slower for addition.

A big-O run time analysis was performed to anticipate the outcomes of the run time analysis. A analysis of the three methods in contention are discussed below.

add() method:

For IntArrayBag, the time complexity is O(N) because the method needs to traverse through some length of an array before pushing the other elements to the next position.

For IntLinkedBag, the time complexity is O(N) because the method needs to find the value in the bag that is greater than the target element, which could be all the elements in the bag.

addAll() method:

For IntArrayBag, the time complexity is O[(N1+N2)log(N1+N2)] where N1 elements are added to the list and N2 elements are already in the bag. N1 elements are first added to the end of the bag, before merge sort is called to sort the final array of (N1 + N2).

For IntLinkedList, the time complexity is O[([(N1+N2)log(N1+N2)] as the linked list elements (N1) are added to the start of the array. Similarly, merge sort is called to sort the final array.

remove() method:

For IntArrayBag, the time complexity is O(N) because the element is found, removed, and then all the elements before it are shifted back one position.

For IntLinkedBag, the time complexity is O(N) because the element is found, removed, and then all the elements before it are shifted one position.

# Results

The results of graphing the run time analysis of each method is described below.

add() method:

addAll() method

remove() method

# Discussion

The add() and remove() method performed as expected. The run time analysis shows that for adding and removing, IntArrayBag is substantially faster than IntLinkedList even at higher values of N.   
  
The addAll() method performed as expected for the IntArrayBag. After adding the array to the end of the ArrayBag, the merge sort algorithm sorts the new array.

The addAll() method for IntLinkedList did not perform as expected. There was a consistent Stack Overflow error when N exceeded 3000, caused by too many recursive loops in the mergeSort algorithm. This is an expected outcome, because recursive mergeSort uses a lot of auxiliary memory. For large data sets, mergeSort is not a preferred method to sort data.

In an attempt to circumvent the Stack Overflow error, we investigated non-recursive methods to writing mergeSort. Because neither of us have experience in developing our own algorithm, we pursued resources online, and found that even non-recursive methods implemented the stack similarly, which will still give a overflow exception. We considered several versions of a non-recursive linked list and noticed that all implementations on different sites gave the same response. Therefore, in conclusion, it is not recommended to use mergeSort for an analysis of large datasets exceeding N=3000.

# Contributions

Fiona developed the IntArrayBag methods for add, addAll, and remove.

Varun developed the IntLinkedList methods for add, addAll, and remove.

# Appendix