Assignment 5 - Array List

Computer Programming 2

Robert Boerwinkle

Spring 2022

1 Introduction

This is an implementation of an array list written in java. An array list is a list whose items are contiguous in memory. This makes indexing easy, as the location in memory can be easily calculated. The drawback of arrays is that to add a new element, the program must create an entirely new space in memory and copy over all of the previous values. Array lists have a buffer by using an array that is larger than it needs to be. It limits interaction to the parts of the array which are active. When the active part fills up the buffer, the `ArrayList` allocates twice the amount of memory for a new array, and starts over. The behavior of the functions is loosely based on `java.util.ArrayList` . It will be implementing `CP2List` . The constructor is quite simple, as it only needs to store an array and how long the list should be.

```
public class ArrayList implements CP2List{
        private int[] array;
        private int size;

        public ArrayList(){
                array = new int[1];
                size = 0;
        }

        public ArrayList(int[] list){
                array = list;
                size = list.length;
        }
}
```

2 Internal Functions

The most important part of the `ArrayList` is the function that ensures there will always be at least one empty spot. It is called whenever elements will be added to the list.

```
private void ensureSize(){
        if(size == array.length){
                array = Arrays.copyOf(array,array.length*2);
        }
}
```

3 External Facing Functions

One of the goals of an `ArrayList` is to be able to add and remove elements easily. It allows for insertion at an arbitrary index, and is overloaded by a function that only takes a value which appends the element to the end. Notice the `ensureSize` call before anything gets added.

```
public void add(int index, int value){
        if(index<0 || index>size){throw new IndexOutOfBoundsException();}
        ensureSize();
        for(int i=size; i>index; i--){
                array[i] = array[i-1];
        }
        array[index]=value;
        size++;
}
```

If any elements are removed, the internal array doesn't change much. If the removed element is not at the end, the following elements are shifted to follow. The array is never shrunk, because it would be costly to do so and chances are that it will need to be that size again anyway. There are two ways to remove elements: by value and by index. They are named `remove` and `pop` respectively in `CP2List`. In `java.util.ArrayList`, both functions are called `remove`.

```
public int pop(int index){
        if(index<0 || index>=size){throw new IndexOutOfBoundsException();}
        int value = array[index];
        for(int i=index; i<size-1; i++){
                array[i] = array[i+1];
        }
        size--;
        return value;
}

public boolean remove(int value){
        for(int i=0; i<size; i++){
                if(array[i] == value){
                        pop(i);
                        return true;
                }
        }
        return false;
}
```

4 Speeds

Speed tests were done on some of the functions to compare to `java.util.ArrayList` times. These tests were done on a Raspberry Pi 3 Model B with no other unnecessary programs running so that background tasks on my personal computer wouldn't affect times. The speed tests can be found in `Program.java`.

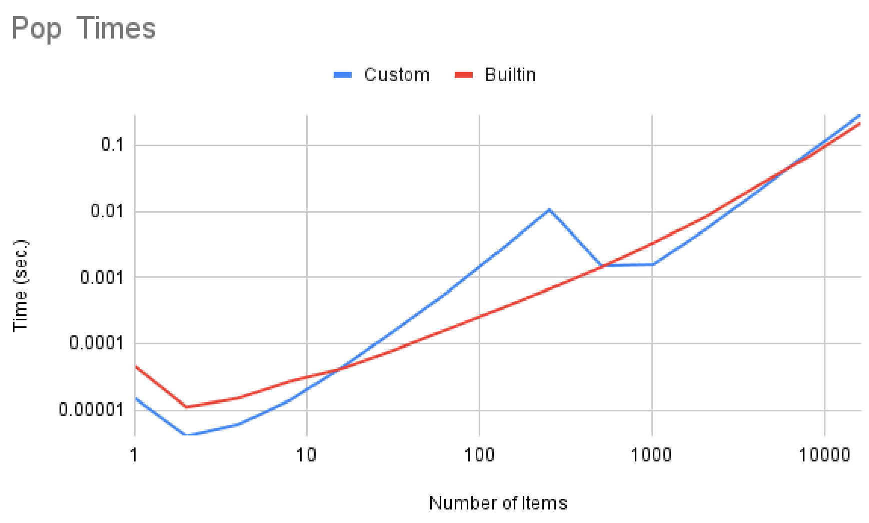

Figure 1: Time it takes to pop all items from the start of a list of a given size

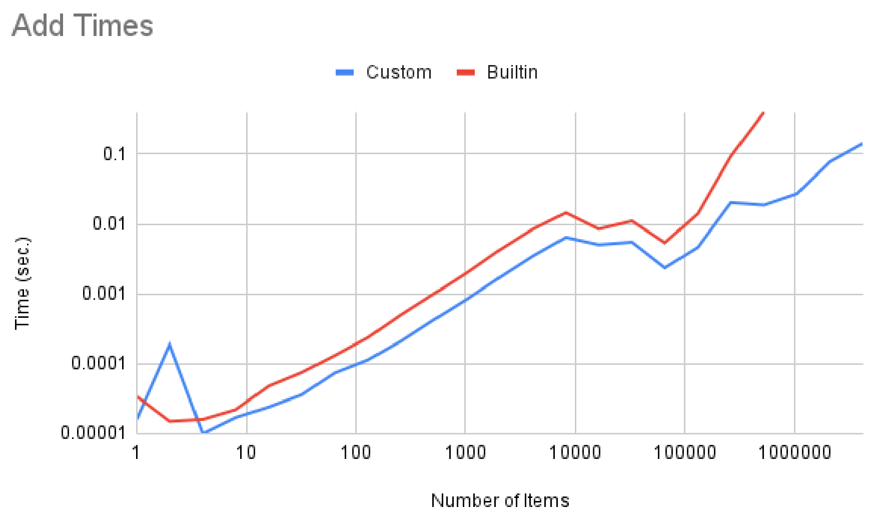

Figure 2: Time it takes to add a given number of items to an empty list

5 Conclusion

This was another recreation project. It was mostly just a proof of concept, though, as a real `ArrayList` can hold other types of objects and not just `int` . The times matched up very well with `java.util.ArrayList` , indicating that my implementation is very close to the real one. My `ArrayList` was usually faster, probably because it only handles `int` .