ID1021: Linked lists

Alexander Lundqvist

20-09-2022

1 Introduction

For this assignment we were tasked with implementing a simple linked list data structure and compare it to simple arrays.

2 Method

To implement the linked list structure we received a code template in the assignment. However, I decided to implement the solution in another way since I've worked with linked lists before and I didn't find the proposed solution to my liking. A linked list can be described as a series of elements that contain both a value and also a reference to the following element in the series. This is illustrated in figure 1.

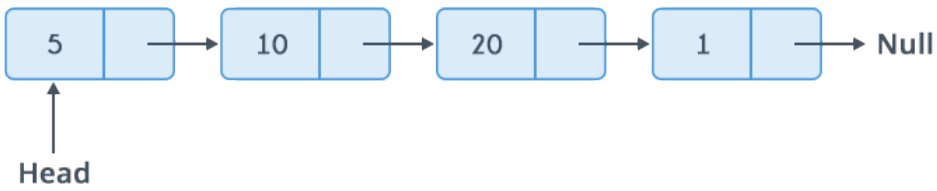

Figure 1: Illustration of a single linked list.

To implement the linked list I used an auxiliary inner node class to represent the elements in the list. The nodes have a value and a reference to the following node.

```
private class Node {
        private int value;
        private Node next;

        public Node(int value, Node node) {
            this.value = value;
            this.next = node;
```

```
        }

        public int getValue() {
            return this.value;
        }
    }
```

For standard linked lists we usually have push/append/add and pop/remove operations that deal with single elements. In this assignment we are also supposed to add another operation that allows for adding another list to the current list. The method is described below.

```
public void appendFirst(LinkedList list) {
    Node current = list.head;
    while(current.next != null) {
        current = current.next;
    }
    current.next = this.head;
    this.head = list.head;
}
```

As we also want to compare the linked list operation with a similar operation on primitive integer arrays and measure the performance of both. The following code is used for this.

```
public static int[] append(int[] static_array, int[] array) {
        int[] new_array = new int[static_array.length + array.length];
        int index = 0;
        for(int i = 0; i < static_array.length; i++) {
            new_array[index++] = static_array[i];
        }
        for(int j = 0; j < array.length; j++) {
            new_array[j+index] = array[j];
        }
        return new_array;
    }
```

However when running the benchmark I got weird results that didn't correlate to my initial theory. I figured that we wanted to do a more fair comparison and thus I changed the method to deal with integer objects instead of primitive integers.

The entire code is available at this github repository.

3 Result

The first benchmark was done on the linked list. In the assignment we were told to test the append functions with different value for both arrays/lists but I didn't feel it was necessary since I already had done some calculations and had a good theory on the outcome. In any case, the performance for the linked list is presented in figure 2.

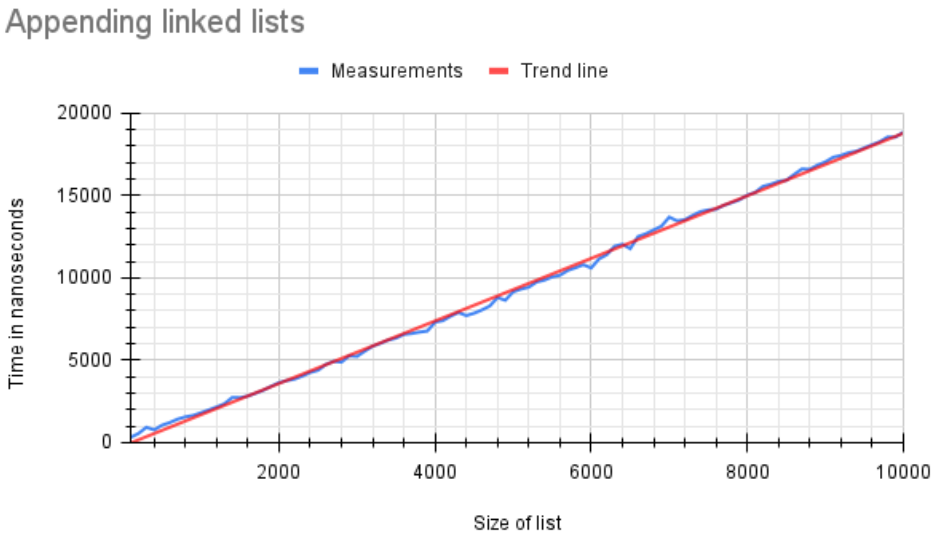

Figure 2: Performance of append operation in linked list.

Here we can see that the operation appears to have $O(n)$ time complexity.

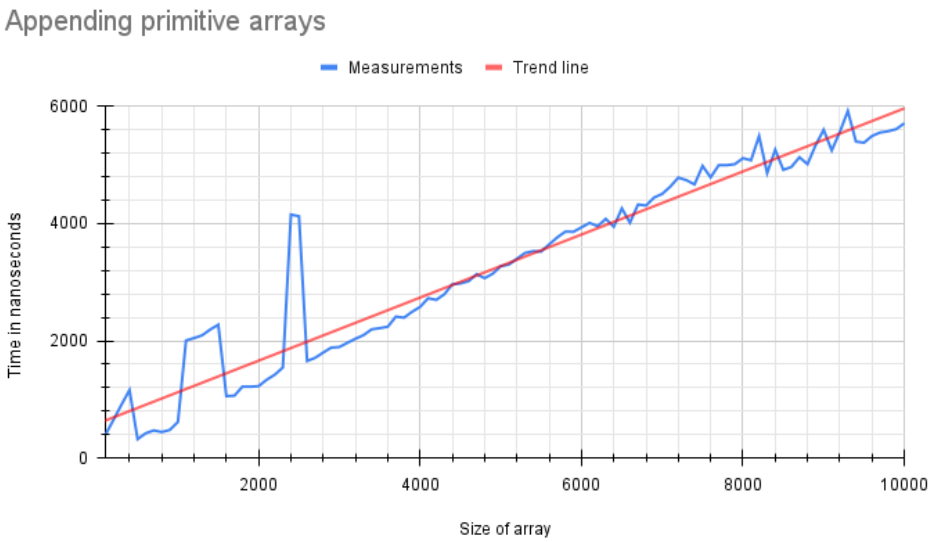

Figure 3: Performance of append operation on arrays.

The second benchmark in figure 3 describes the performance of the comparable array method. However, it was at this stage when I came to the conclusion that I should measure against integer objects instead of primitive values. The new measurements are presented in figure 4.

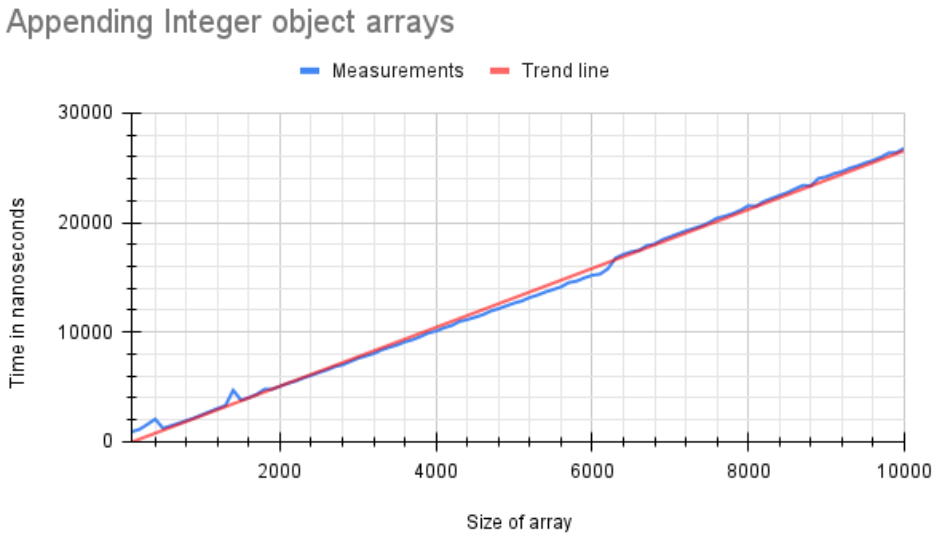

Figure 4: Performance of append operation on integer object arrays.

We can see that also in this benchmark the time complexity appears to be linear or $O(n)$. We can also note that it takes considerably more time to deal with objects instead of primitive values. The method also seems to have slower performance than the linked list method.

4 Discussion

When I first started this assignment I just re-implemented a more complex linked list class that I had created from a previous year. At first I didn't read the instructions fully so I only used push/pop operations and I also started to modify the dynamic stack that we had done in an earlier assignment to use for comparison.

I then read what the instructions actually wanted me to do and got a bit confused since my initial thought was that both the array appending and the list appending would be similar in performance (Which was also proven in the result). To illustrate this I will show some simple calculations.

In the instructions we are supposed to measure different lengths for both arrays but as we will see it won't affect the actual result and as such, I didn't consider it necessary to test extensively. Since we are to essentially merge two separate arrays/lists we will denote the first array with length n and the other with length m.

As described in the code for the linked list append function, to merge two lists we only have to iterate through one of them. If we choose to merge array n into array m we would therefore get a time complexity of $O(n)$.

In the case of merging arrays we first have to create a new array and then iterate through both array n and array b. This gives us a time complexity of $O(n+m)$, which in reality is $O(n)$, the same as for the linked list method. So even if we would change n and m respectively, we would get something like $O(3n+2m)$ which still is only $O(n)$ since we can omit constants in big-o calculations.

This is why I found it weird that we were supposed to examine linked lists in this manner. In my opinion it would be more interesting to examine the differences operations and performance between linked lists and for example dynamic arrays as for example, push will have $O(1)$ in linked list where as in dynamic arrays we would have to deal with the whole resizing issue. Another difference between lists and arrays is that you can't really access an element at a specific index in a linked list as opposed to an array where we can just specify an index.

All in all, this assignment felt pretty unnecessary and boring since there are lots of other ways to study the linked list data structure.

There are lots of ways we can improve the linked list data structure. A common way to add more functionality is to have both a head and a tail reference, thus eliminating the need to iterate through one of the lists when appending. Another way to improve the list is to make it doubly linked to improve the traversal capabilities. There are also sentinel elements that could be added and also link both the head and tail to make the list circular.