COMP 340 **Analysis of Algorithms**

Problem Set 02

Assigned: February 02, 2024 *Due: February 16, 2024*

1. A few years ago I was writing a special *event logger*. The idea was that my program would run alongside a user's program, recording a particular type of event. Whenever the event occurred, my program would save some details about it, and once the user's program ended my program would write all the event details to a log[1]. In other words, I needed some kind of container that allowed inserting at the end and traversing through the elements in the order they were inserted. Importantly, nothing else was required: no search, no delete, etc.

   After a few prototypes, I came to the conclusion that both plain linked lists and dynamic arrays were too slow. A linked list wasted too much space on pointers, required too many memory allocations[2], and was too slow to iterate through since the items were not contiguous in memory. A dynamic array spent too much time copying elements during resizing[3].

   (a) Briefly explain a hybrid design a hybrid between linked lists and dynamic arrays with the following properties:
   - $O(1)$ **worst-case** `push` (i.e. insert at end)
   - Simple $O(n)$ traversal through all the elements
   - No copying of elements ever occurs
   - Large groups of elements are stored contiguously
   - Only $O(\log n)$ memory allocations

   (b) Briefly explain why your design satisfies the above properties.

   (c) Say I wanted to access some particular index. How would I do this and how long would it take in the worst case, assuming the above data structure with n items.

2. 3-3
3. 3-4
4. 3-11
5. 3-12
6. 3-15
7. 3-18
8. 3-19
9. 3-25
10. 3-27. Notice that the black box only returns *true* or *false*; we want to find the actual subset S.
11. LeetCode 3-1: `https://leetcode.com/problems/validate-binary-search-tree/`
12. [Bonus] 3-28
13. [Bonus] 3-31

[1] For technical reasons my program could not simply write the event information to a file at each individual event; it had to collect them into a batch.

[2] In practice, memory allocation can be expensive

[3] Even though some data structures and algorithms are efficient in terms of Big-O, sometimes the constants that Big-O hides are quite large and are important in practice.