Your name
CS 261, Spring 2021
Homework 1
Due Monday, April 5

NOTE: Homeworks for this class are provided both as a pdf file and as a LaTeX template. You may use the template to produce your answers as a pdf file (for instance by using a local LaTeX installation or by copying the template onto overleaf.com) or you may produce the answers as a pdf file in any other way (including writing them by hand and then scanning them, as long as your handwriting is legible).

1. Binary counter API. The lecture notes described one part of the API of a binary counter, an operation for incrementing the counter. However, without any way to retrieve information from the counter, this operation is useless (any algorithm that used a counter only by incrementing it, with no other operations, could just skip all of those operations without affecting the rest of the algorithm). Describe the API for four other operations that would be reasonable to perform on a binary counter: what arguments do these operations take, what effect do they have on the counter, and what value (if any) do they return? At least two of the operations you list should return a value that depends on the state of the counter, and at least one should change the value of the counter. Do not describe how to implement these operations.

Your solution to problem 1 goes here. ∎

2. Square dynamic arrays. Suppose we implement the dynamic array API in the same way as described in the lecture, except for the part of the dynamic array that resizes it to a larger array when we run out of room (on the "Implementing increase-length operation" slide). In that slide, we compute the new array size to be twice the previous size, so that the sizes are always powers of two. Instead, suppose that we use array sizes that are always square numbers (1, 4, 9, 16, 25, etc), and get a new block of the next square size when the previous block becomes full.

Use the averaging method (not the potential function method) to compute the average time per operation for a sequence of n increase-length operations, using this square resizing strategy. Express your answer using O-notation, as a function of n.

(Hints: you may find the formula for the square pyramidal numbers from `https://en.wikipedia.org/wiki/Square_pyramidal_number` helpful. You may assume that finding the next square number to use as the size of the new block can be done in constant time. Your O-notation should be as simple as possible, omitting unnecessary constant factors and lower-order terms. Do not worry about other operations than increase-length.)

Your solution to problem 2 goes here. ∎

3. Potential functions for array-doubling. In the lecture notes for the array-doubling dynamic array data structure (before the improvements to the space complexity) we used the potential function $|2 \times \text{length} - \text{available}|$. The same analysis can also work for some other functions of the form $\Phi = |L \times \text{length} - A \times \text{available}|$, for other choices of L and A.

1. For which values of L and A will the function Φ defined in this way be ≥ 0 for all states of the data structure?
2. For which values of L and A will the change to Φ, in an increase-length operation that reallocates the array, be a negative number that is proportional to the time of the operation?
3. Find a different choice for L and A than the choice $L = 2$ and $A = 1$ from the lecture notes, that obeys both of these conditions. In order to be sufficiently different, your choice should have $L \neq 2A$.

Your solution to problem 3 goes here. ∎

4. Stacks with lazy deletion. Suppose we wish to modify the stack API and implementation in the following ways.

- Whenever we push an item x onto the stack, the push function returns the location where x is stored, either a node in the linked-list implementation of a stack, or an index into the array in the (non-dynamic) array implementation of a stack.
- We have a new operation "delete", that takes one of these locations as its argument and replaces x by a special flag value that indicates that x has been deleted.
- We modify the top and pop operations so that they skip past all deleted items before returning the topmost non-deleted item (and then updating the top of the stack to point to that item or the next item below it, respectively).

The worst-case time of the new top and pop operations may be large, because many deleted items may need to be skipped. The actual time for these operations is proportional to the number of skipped items, plus one. Define a potential function for which all operations in this modified stack take constant amortized time.

Your solution to problem 4 goes here. ∎