0 Overview

Instructions

Due Date: Sunday, Oct 13 @ 11:59PM

Total points: 50

Your written solution may be either handwritten and scanned, or typeset. Either way, you must produce a PDF that is legible and displays reasonably on a typical PDF reader. This PDF should be submitted via autolab as WA3. You should view your submission after you upload it to make sure that it is not corrupted or malformed. Submissions that are rotated, upside down, or that do not load will not receive credit. Illegible submissions may also lose credit depending on what can be read. Ensure that your final submission contains all pages.

You are responsible for making sure your submission went through successfully.

Written submissions may be turned in up to one day late for a 50% penalty.

No grace day usage is allowed.

1 Proof by Induction

In this written assignment, you will use induction to prove the runtime of the standard Towers of Hanoi algorithm.

In the towers of Hanoi, you are given 3 stacks of blocks. The blocks must be kept in order within a given stack (largest at the bottom, smallest at the top), but may be spread out across the three stacks however you like. You are allowed to move one block at a time, and may never place a bigger block on a smaller block. The algorithm below takes the three stacks (labeled, `from`, `to`, and `spare`) and moves the topmost N blocks from the `from` stack, to the `to` stack. The `spare` stack is used as temporary storage. If the `to` and `spare` stack are empty (or their smallest block is larger than the Nth smallest `from` block), the algorithm is guaranteed to never put a larger block on a smaller one[1]

```
public int moveBlocks<T>(int N, Stack<T> from,
                              Stack<T> to, Stack<T> spare) {
  if(N <= 1){
    // Move the block from [from], to [to]
    T block = stacks[from].pop();
    stacks[to].push(block);
  } else {
    // Move N-1 blocks from [from], to [spare]
    moveBlocks(N-1, from, spare, to);
    // Move the last block from [from], to [to]
    T block = stacks[from].pop
    stacks[to].push(block);
    // Move N-1 blocks from [spare] to [to]
    moveBlocks(N-1, spare, to, from);
  }
}
```

Question 1 (20 pt): Setup and Hypothesis

Before we prove anything about the runtime of the code above, we have to determine what the code's runtime growth function is, and come up with a hypothesis for its runtime bounds. For the runtime, you can ignore the `from`, `to`, and `spare` parameters. Assume that Stack pushes and pops are always $O(1)$ (e.g., the stack is backed by a Linked List).

- **(10 pt)** Write out the recursive form of the growth function, $T(n)$ for `moveBlocks(n)`.

[1]The details of this guarantee are not immediately relevant, but if curious, stop by office hours and ask.

- **(5 pt)** Draw the recursion tree for `moveBlocks`. Label each node with the runtime of a call to `moveBlocks`, excluding the cost of recursive calls. Label the height of your tree in terms of n.
- **(5 pt)** Based on your recursion tree, give a hypothesis for the closed-form, tight upper bound of your growth function (i.e., a hypothesis of the form $T(n) = O(f(n))$). In at most two sentences, explain how you used your recursion tree to come up with the hypothesis.

Question 2 (10 pt): Base Case

Prove that your hypothesis holds true for an appropriate number of base cases. Make sure to consider how many base cases you will need to prove, based on your growth function.

Question 3 (20 pt): Inductive Case

In order to prove that our hypothesis is true for **all** values of n, we must use induction. Remember that this involves showing how to prove the hypothesis for a specific value of n by 'chaining' a proof from a smaller value of $f(n)$.

- **(5 pt)** Make an appropriate inductive assumption (i.e., to prove the hypothesis for a specific value n, what $f(n)$ do you plan to chain a proof from).
- **(15 pt)** Prove that if your inductive assumption is true, then your hypothesis must be true for $T(n)$ (i.e., show how to chain a proof for $T(f(n))$ to $T(n)$).