Assignment 2

Algorithms, Spring 2024

Honor code: *Work on this assignment alone or with one partner. Between different teams, collaboration is at level 1 (verbal collaboration only). It is not allowed to search online for the specific problems in this assignment—doing so violates academic honesty for the class.*

Preamble

The first problem asks you to prove a certain property related to big-oh. We'll start with an example.

Claim: For any functions $f, g : N \to N$, prove that $f(n))$ is $O(g(n))$ implies that $g(n)$ is $\Omega(f(n))$.

Note that this is a *universal statement*, about *any* functions $f(n)$ and $g(n)$. To prove that it is true, we must show that it works in *all* cases. This is hard because we cannot try *every single* function f and *every single* function g.

Rule of thumb: To prove a universal statement, you must show that it works in all cases. Note that if we want to *disprove* a universal statement, we only need to find one counterexample.

We always start by writing down what we know and what we need to show.

- What we know: Let $f(n), g(n)$ be two functions such that $f(n) = O(g(n))$.
- What we want to prove: We want to show that $g(n) = \Omega(f(n))$.

Now let's plug in the definitions of $O()$ and $\Omega()$:

- $f(n) = O(g(n))$ by definition means that there exists a constant $c > 0$ such that $f(n) \leq c \cdot g(n)$, for any $n \geq n_0$.
- $g(n) = \Omega(f(n))$ by definition means that there exists a constant $c > 0$ such that $g(n) \geq c \cdot f(n)$, for any $n \geq n_0$. Note that the constant c and the n_0 here will be different than the ones above, so to make that clear, we can denote them with different names, for e.g. c' and n_0': $g(n) = \Omega(f(n))$ means that there exists a constant $c' > 0$ such that $g(n) \geq c \cdot f(n)$, for any $n \geq n_0'$.

To prove the claim, we need to show that $f = O(g))$ impliesthat $g = \Omega(f)$, which, by plugging in the definition of $O()$ and $\Omega()$, means that we need to show that:

For any functions $f, g : N \to N$: If there exists a constant $c > 0$ such that $f(n) \leq c \cdot g(n)$, for any $n \geq n_0$, this means that there exists a constant $c' > 0$ such that $g(n) \geq c' \cdot f(n)$, for any $n \geq n_0'$.

Now that we have written down in terms of the definition what we know and what we need to show, let's show it. Here is the actual proof:

Proof of the claim: Let $f(n), g(n)$ be any two functions such that $f(n) = O(g(n))$. By definition we know that there exists a constant $c > 0$ such that $f(n) \leq c \cdot g(n)$, for any $n \geq n_0$. This means that $g(n) \geq \frac{1}{c} \cdot g(n)$, for any $n \geq n_0$. Since c is a constant and $c > 0$, then $1/c$ is also a constant; let $c' = 1/c$. Then it follows that $g(n) \geq c' \cdot g(n)$, for any $n \geq n_0$. By definition, this means that $g(n) = \Omega(f(n))$. This is what we had to prove.

Since this logic works for *any* functions $f(n), g(n)$ such that $f(n) = O(g(n))$, we have shown that $f(n) = O(g(n))$ implies that $g(n) = \Omega(f(n))$.

An alternate proof

In general there can be more than one way to prove things. Here's a different way to prove the Claim above, using the definition of $O()$ and $\Omega()$ in terms of limits.

Again, we start by writing the definition of what we have and what we want to prove:

- What we have: $f(n) = O(g(n))$, By definition means that the limit $\lim \frac{f(n)}{g(n)}$ is either 0 or a constant $c > 0$.
- What we want to show: $g(n) = \Omega(f(n))$ by definition means that the limit $\lim \frac{g(n)}{f(n)}$ is either ∞ or a constant $c > 0$.

Let's show that the first implies the second.

Alternate proof: Let $f(n), g(n)$ be any two functions such that $f(n) = O(g(n))$. By definition we know that the limit $\lim \frac{f(n)}{g(n)}$ is either 0 or a constant $c > 0$. From here it means that $\lim \frac{g(n)}{f(n)}$ is either ∞ or a constant $1/c$. By definition, this means that $g(n) = \Omega(f(n))$. This is what we had to prove.

Again, since this logic works for *any* functions $f(n), g(n)$ such that $f(n) = O(g(n))$, we have shown that $f(n) = O(g(n))$ implies that $g(n) = \Omega(f(n))$.

The problems

1. (5 points) Prove that big-O is transitive, that is: For any functions $f(n), g(n), h(n)$ such that $f(n)$ is $O(g(n))$ and $g(n)$ is $O(h(n))$, show that this implies that $f(n)$ is $O(h(n))$.

   Hint: Use one of the definitions of $O()$, as in the preamble.

2. (5 points) Prove or disprove: $n^2 \log^{10} n$ is $O(n^{2.1})$

   Hint: You can manipulate this expression and reduce to one of the two basic rules that were discussed in class: (1) any polylogarithm is upper bounded by any polynomial; and (2) any polynomial is upper bounded by any exponential. Or, you can show this using the definition of $O()$ in terms of limits.

3. (5 points) Prove or disprove: 2^{2n} is $O(2^n)$

4. (5 points) Prove or disprove: 4^n is $\Theta(2^n)$

5. (6 points) For each of the following functions, prove whether $f = O(g), f = \Omega(g)$ or both i.e. $f = \Theta(g)$.

   (a) $f(n) = n \lg(n^3), g(n) = n \lg n$

   (b) $f(n) = 2^{2n}, g(n) = 3^n$

   (c) $f(n) = \sum_{i=1}^{n} \lg i, g(n) = n \lg n$

6. (5 points) Consider the following algorithm:

   Algorithm A(n):

   1. For $i = 0, 1, 2, ...\lceil \lg n \rceil - 1$:

   2. (do something)

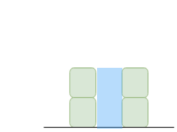
   Suppose that step 2 takes $\frac{n}{2^i}$ steps of computation in the i-th iteration. What is the total running time of algorithm A, as a function of n, using $\Theta()$ notation? Show your calculation.

7. (5 points) **Flooding 1D-terrain** You are given an array and assume its values represent heights and that the distance between these heights is 1m[1]. Imagine an arbitrarily large amount of water falling from the sky and flooding the terrain, and also imagine that the terrain is surrounded by a giant sink/ocean. Water will accumulate on top of a cell in the terrain unless it can find a path to the ocean. At some point flooding will reach a point (called: steady state) when further rain will no longer increase the flooding.

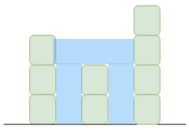
   The goal is to compute the total amount of water that is accumulated in the terrain (aka volume of flood) when steady state is reached. You may consider that the elevations are all non-negative (i.e. ≥ 0), and the width of each "pixel" (ie the distance between two consecutive heights in the array) is 1 unit.

[1] A different way to say this is that the array represents the heights of a one-dimensional terrain sampled at 1m resolution

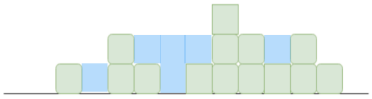
Examples:



Input: A = [0, 2, 0, 2, 0]
Output: 2

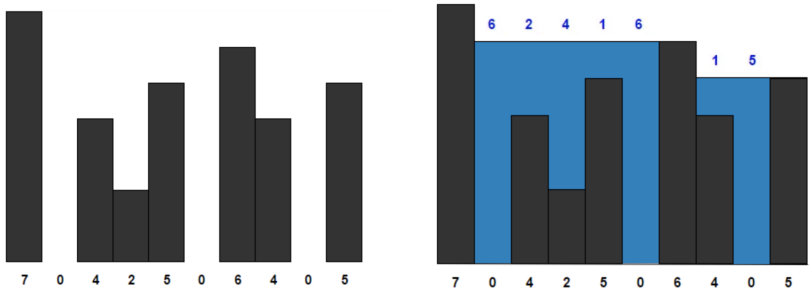


Input: A = [0, 3, 0, 2, 0, 4, 0]
Output: 7



Input: A = [0, 1, 0, 2, 1, 0, 1, 3, 2, 1, 2, 1, 0]
Output: 6

The maximum amount of water that can be trapped is 25, as shown below (blue).



Aim for a linear time algorithm.

(a) Formulate a claim on what is the height of water accumulated on $A[i]$, in terms of the elevation of the pixels to its left and right. Hint: consider min and max values.

(b) The rationale of your algorithm (English), and pseudocode

(c) Analysis of the running time.

Evaluation

The assignment will be evaluated along three criteria: correctness, justification and style.
Style guidelines:

- Space: Write each problem on a separate page or leave plenty of space between problems so that we can write comments.
- Neat and legible: We expect your answers to be clearly written and easily human-readable, with complete sentences and well-organized logic, and should definitely not be your first draft. Ideally answers should be typed.
- Clear: Try to put yourself in the position of the reader. If you hadn't been thinking of this problem for 3 hours, would your answers make sense to you? Try to finish the assignment early, then step away for a day or two, and then come back to it and read it again. Chances are you'll find something you can write more clearly.