CS 161 W22: Section 1 Problems

January 2022

Exercise 0

For each of the following functions, prove whether $f = O(g)$, $f = \Omega(g)$, or $f = \Theta(g)$. For example, by specifying some explicit constants n_0 and $c > 0$ such that the definition of Big-Oh, Big-Omega, or Big-Theta is satisfied.

(1)	$f(n) = 4n^2 + 20$	$g(n) = n^2$
(2)	$f(n) = 4n^2 + 20$	$g(n) = n^3$
(3)	$f(n) = n\log(n^3)$	$g(n) = n\log n$
(4)	$f(n) = 2^{2n}$	$g(n) = 3^n$
(5)	$f(n) = \sum_{i=1}^{n} \log i$	$g(n) = n\log n$

Exercise 1

1 How NOT to prove claims by induction

In this class you'll prove a lot of claims, many of them by induction. You will also prove some wrong claims, and catching those mistakes will be an important skill!
The following are examples of a false proof where an obviously untrue claim has been 'proven' using strong induction with some minor error or detail left out. Your task is to investigate the 'proofs' and identify the mistakes made.

1. **Fake Claim 1:** For every nonnegative integer n, $2^n = 1$.

   Inductive Hypothesis: For all integers n such that $0 \leq n \leq k$, it must be true that $2^n = 1$.

   Base Case: For $n = 0$, we know from basic arithmetic that $2^0 = 1$.

   Inductive Step: Let's assume our inductive hypothesis holds for all values between zero and some non-negative integer k. We will prove by strong induction that our inductive hypothesis must also be true for the value $k + 1$. Namely, we will prove that if $2^k = 1$, then it must follow that $2^{k+1} = 1$.

   To prove this claim, we conduct the following algebra:

$$
\begin{aligned}
2^{k+1} &= \frac{2^{k+1}}{1} \\
&= \frac{2^{k+1}}{1}\frac{2^{k-1}}{2^{k-1}} \quad \text{(Multiplying the equation by one.)} \\
&= \frac{2^{k+1}2^{k-1}}{2^{k-1}} \\
&= \frac{2^{2k}}{2^{k-1}} \quad \text{(We sum the powers of terms with like bases.)} \\
&= \frac{2^k 2^k}{2^{k-1}}
\end{aligned}
$$

We're assuming from our inductive hypothesis that $2^n = 1$ for all values for n between 0 and k. (Note that assumptions like these are standard for strong inductive proofs.) Therefore, we can start plugging in values:

$$
\begin{aligned}
&= \frac{2^k 2^k}{2^{k-1}} \\
&= \frac{1 \cdot 1}{1} \\
&= 1
\end{aligned}
$$

2. **Fake Claim 2:** For every positive integer n,

$$\frac{1}{1 \cdot 2} + \frac{1}{2 \cdot 3} + ... + \frac{1}{n \cdot (n+1)} = \frac{3}{2} - \frac{1}{n}$$

Inductive Hypothesis: The equation above holds for $n = k$.

Base Case: For $n = 1$, We see that

$$\frac{1}{1 \cdot 2} = \frac{1}{2} = (\frac{3}{2} - \frac{1}{1})$$

Inductive Step: As usual, let's assume our inductive hypothesis holds for $n = k$. We will use this assumption to prove that our inductive hypothesis must hold for the value $k + 1$.

Once again, let's use algebra to carry out the following calculations:

$$\frac{1}{1 \cdot 2} + \frac{1}{2 \cdot 3} + ... + \frac{1}{(k-1) \cdot k} = \frac{3}{2} - \frac{1}{k} \quad \text{(By weak induction hypothesis).}$$

$$\mathbf{(}\frac{1}{1 \cdot 2} + \frac{1}{2 \cdot 3} + ... + \frac{1}{(k-1) \cdot k}\mathbf{)} + \frac{1}{k \cdot (k+1)} = \mathbf{(}\frac{3}{2} - \frac{1}{k}\mathbf{)} + \frac{1}{k \cdot (k+1)} \quad \text{(Add the same term to both sides).}$$

Let's now focus on this term on the right. We'll algebraically simplify it. (Note: Don't worry about the algebra for separating a fraction into two fractions. That part is correct.)

$$
\begin{aligned}
&\mathbf{(}\frac{3}{2} - \frac{1}{k}\mathbf{)} + \frac{1}{k \cdot (k+1)} \\
&= (\frac{3}{2} - \frac{1}{k}) + \frac{1}{k} - \frac{1}{(k+1)} \quad \text{(Separating a fraction into two fractions.)} \\
&= \frac{3}{2} - \frac{1}{(k+1)}
\end{aligned}
$$

Conclusion: The summation $\frac{1}{1 \cdot 2} + \frac{1}{2 \cdot 3} + ... + \frac{1}{(k-1) \cdot k} + \frac{1}{k \cdot (k+1)}$ is equivalent to $\frac{3}{2} - \frac{1}{(k+1)}$. Thus, by weak induction, our claim follows.

Exercise 2

2 Binary Search

Given a sorted array A, prove that binary search as defined below correctly returns the index of the element x in A if it appears in A (or None otherwise). Use the following code as a guide:

```
def binarySearch(array, target, left, right):
    #base case - if target isn't in array
    if left > right:
        return None

    #recursive case(s)
    mid = (left + right) // 2
    if array[mid] == target:
        return mid
    if array[mid] < target:
        return binarySearch(array, target, mid+1, right)
    else:
        #if array[mid] > target
        return binarySearch(array, target, left, mid-1)
```

Hint: While solving this problem, remember that recursion and induction often go hand-in-hand. They both have a base case and a "repeating subproblem" case. In recursion, we call that "repeating" case the "recursive step". Meanwhile, in induction, we call the repeating case the "inductive step". Use this structural similarity to aid you in transforming the recursive code above into an inductive proof. Wink Wink.