CS 161 Winter 2021 Section 1

January 14, 2021

Asymptotic Analysis

Asymptotic Analysis Definitions

Definition 1 (Big-oh notation)**.** Let f, g be functions from the positive integers to the non-negative reals. Then we say that:

$f(n) = O(g(n))$ if there exist constants $c > 0$ and n_0 such that for all $n > n_0$,

$$f(n) \leq c \cdot g(n).$$

$f(n) = \Omega(g(n))$ if there exist constants $c > 0$ and n_0 such that for all $n > n_0$,

$$f(n) \geq c \cdot g(n).$$

$f(n) = \Theta(g(n))$ if $f = O(g)$ and $f = \Omega(g)$.

You will use "big-oh notation" A LOT in class. Additionally, you may occasionally run into "little-oh notation":

Definition 2 (Little-oh notation)**.** Let f, g be functions from the positive integers to the non-negative reals. Then we say that:

$f(n) = o(g(n))$ if **for every constant** $c > 0$ there exist a constant n_0 such that for all $n > n_0$,

$$f(n) < c \cdot g(n).$$

$f(n) = \omega(g(n))$ if **for every constant** $c > 0$ there exist a constant n_0 such that for all $n > n_0$,

$$f(n) > c \cdot g(n).$$

Asymptotic Analysis Problems

1. For each of the following functions, prove whether $f(n) = O(g(n))$, $f(n) = \Omega(g(n))$, or $f(n) = \Theta(g(n))$. For example, you can prove by specifying some explicit constants n_0 and $c > 0$ such that the definition of big-oh or big-omega is satisfied. *Bonus: prove little-oh and little-omega (where applicable).*

(a)	$f(n) = n \log(n^3)$	$g(n) = n \log n$
(b)	$f(n) = 2^{2n}$	$g(n) = 3^n$
(c)	$f(n) = \sum_{i=1}^{n} \log i$	$g(n) = n \log n$

2. Give an example of f, g such that $f(n)$ is not $O(g(n))$ and $g(n)$ is not $O(f(n))$.

Induction

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 $0 \leq n \leq k$, $2^n = 1$.

Base Case: For $n = 0$, $2^0 = 1$.

Inductive Step: Suppose the inductive hypothesis holds for k; we will show that it is also true $k+1$, i.e. $2^{k+1} = 1$. We have

$$\begin{aligned} 2^{k+1} &= \frac{2^{2k}}{2^{k-1}} \\ &= \frac{2^k \cdot 2^k}{2^{k-1}} \\ &= \frac{1 \cdot 1}{1} && \text{(by strong induction hypothesis)} \\ &= 1 \end{aligned}$$

Conclusion: By strong induction, the claim follows

2.

Fake Claim 2. For every positive integer n,

$$\underbrace{\frac{1}{1 \cdot 2} + \frac{1}{2 \cdot 3} + \ldots}_{n \text{ terms}} = \frac{3}{2} - \frac{1}{n}. \tag{1}$$

Inductive Hypothesis: (1) holds for $n = k$

Base Case: For $n = 1$,

$$\frac{1}{1 \cdot 2} = 1/2 = \frac{3}{2} - \frac{1}{1}.$$

Inductive Step: Suppose the inductive hypothesis holds for $n = k$; we will show that it is also

true $n = k + 1$. We have

$$\begin{aligned}\left(\frac{1}{1\cdot 2}+\frac{1}{2\cdot 3}+\cdots\frac{1}{(k-1)\cdot k}\right)+\frac{1}{k\cdot(k+1)} &= \frac{3}{2}-\frac{1}{k}+\frac{1}{k\cdot(k+1)} \quad \text{(by weak induction hypothesis)}\\ &= \frac{3}{2}-\frac{1}{k}+\frac{1}{k}-\frac{1}{k+1}\\ &= \frac{3}{2}-\frac{1}{k+1}.\end{aligned}$$

Conclusion: By weak induction, the claim follows

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).

```
def binarySearch(x, A, l, r):
    if l > r:
        return None
    m = (l + r) / 2
    if x == A[m]:
        return m
    if x < A[m]:
        return binarySearch(x, A, l, m-1)
    else:
        return binarySearch(x, A, m+1, r)
```

The first call to the algorithm is: `binarySearch(x, A, 0, len(A) - 1)`.