CSCI 3104 Spring 2017 Problem Set 1

Matthew Niemiec (06/24)

---

1. *For each of the following claims, determine whether they are true or false. Justify your determination. If the claim is false, state the correct asymptotic relationship as O, Θ, or Ω.*

   (a) *$n + 3 = O(n^2)$*

   True. n^2 will be greater than $n + 3$ as n approaches infinity for any constant. However, it is more accurate to say $n + 3 =$ O(n).

   (b) *$3^{2n} = O(3^n)$*

   False. After the following:

$$\lim_{n\to\infty} \frac{3^{2n}}{3^n} = \infty \tag{1}$$

   So there is no constant that you can use to change infinity.

   (c) *$n^n = \Theta(n!)$*

   False. We see that the limit

$$\lim_{n\to\infty} \frac{n^n}{n!} = \infty \tag{2}$$

   So there is no constant we can multiply by to bound n^n.

   (d) *$1/(3n) = \Omega(n!)$*

   False. Looking at another limit we see

$$\lim_{n\to\infty} \frac{1}{3n} = 0 \tag{3}$$

   Which means that the function will never do worse than constant

(e) $ln^3n = \Theta(lg^3n)$

True. Using the following limit and logarithm change of base formula, we get

$$\lim_{n\to\infty} \frac{ln^3n}{lg^3n} = \lim_{n\to\infty} (\frac{ln(n)}{lg(n)})^3 = \lim_{n\to\infty} (\frac{ln(n)}{\frac{ln(n)}{ln(2)}})^3 = (\frac{1}{ln(2)})^3 = C \tag{4}$$

Which means that the function can be bounded both above and below.

2. *(15 pts total) Simplify the following expressions. Show your work.*

(a) $\frac{d}{dt}(2t^4 + \frac{1}{2}t^2 - 7)$

$$\frac{d}{dt}(2t^4 + \frac{1}{2}t^2 - 7) = \frac{d}{dt}2t^4 + \frac{d}{dt}\frac{1}{2}t^2 - \frac{d}{dt}7 = 4 * 2t^{4-1} + 2 * \frac{1}{2}t^{2-1} - 0 = 8t^3 + t \tag{5}$$

(b) $\sum_{i=0}^{n} 2^i$

Using proof by induction, we prove that $\sum_{i=0}^{n} 2^i = 2^{n+1} - 1$. See the base case n=0:

$$\sum_{i=0}^{0} 2^i = 1 = 2^1 - 1 \tag{6}$$

Now it is proved up to k, so we show it works for k+1, so

$$\begin{aligned} \sum_{i=0}^{k+1} 2^i &= (\sum_{i=0}^{k} 2^i) + 2^{k+1} \\ &= (2^{k+1} - 1) + 2^{k+1} \\ &= 2 * 2^{k+1} - 1 \\ &= 2^{(k+1)+1} - 1 \end{aligned} \tag{7}$$

Thus it is proved for all n

(c) $\Theta(\sum_{k=1}^{n} 1/k)$

We know from calculus that the harmonic series approaches log(n), so

$$\lim_{n\to\infty} \sum_{l=1}^{n} = \lim_{n\to\infty} log(n) \tag{8}$$

So $\Theta(\sum_{k=1}^{n} 1/k) = \Theta(log(n))$.

3. *(5 pts) The young wizards Crabbe and Goyle are having an argument about an algorithm A. Crabbe claims, vehemently, that A has a running time of at least $O(n^2)$. Explain why Crabbe is spouting nonsense.*

   Big-OH is an upper bound on a function's run-time. Therefore, the function cannot run any faster than O(n^2), in this case. What would make sense is to say that A has a running time of at least $\Omega(n^2)$, because that is a lower bound and can describe functions that will never come close to running that fast.

4. *(10 pts total) Using the mathematical definition of Big-O, answer the following. Show your work.*

   (a) *Is* $2^{nk} = O(2^n)$ *for* $k > 1$*?*

   No. We see that the limit always goes to infinity, as shown by the following:

$$\lim_{n\to\infty} \frac{2^{nk}}{2^n} = \lim_{n\to\infty} \frac{(2^n)^k}{2^n} = \lim_{n\to\infty} (2^n)^{k-1} = \infty \tag{9}$$

   Because $k - 1 > 0$.

   (b) *Is* $2^{n+k} = O(2^n)$*, for* $k = O(1)$*?*

   Yes. We know that because k = O(1) it is a constant function, or just a constant. After following we get

$$\lim_{n\to\infty} \frac{2^{n+k}}{2^n} = \lim_{n\to\infty} \frac{2^k 2^n}{2^n} = \lim_{n\to\infty} 2^k = 2^k \tag{10}$$

   Which is a constant, so 2^{n+k} is O(2^n)

5. *(15 pts) Exercise 2.1-3 in CLRS: Consider the searching problem:*
Input: A sequence of numbers $A = \{a_1, a_2, ..., a_n\}$ *and a value* v
Output: An index i *such that* $v = A[i]$ *or the special value NIL if* v *does not appear in* A.

Write speudocode for linear search, which scans through the sequence, looking for v. *Using a loop invariant, prove that your algorithm is correct. Make sure that your loop invariant fulfills the three necessary properties.*

Here is the pseudocode for the algorithm:

```
linearSearch(A, v) {
    for i = 1 to A.length{
        if A[i] == v { return i }
    }
    return NIL
}
```

The Loop Invariant is: Each time the loop begins iterating the value v does not exist in the array at an index less than i, so it is not missed.

The Initialization is: The first time the loop iterates, i=1, and there are no values before A[1].

The Maintenance is: if v were at the previous value, then the function would have returned. Therefore, v still is not in the previous part of the array.

The Termination is: When the loop terminates, either v is the last index, in which case it returns i=A.length, or v is not in the array, in which case it returns NIL.

6. *(5 pts) Crabbe and Goyle are at it again. This time, Crabbe is claiming, vehemently, that when n real-valued numbers are arranged in sorted order in an input array A, it is always more efficient to use binary search to find a target v in the array. Goyle claims, loudly and just as vehemently, that sometimes linear search, i.e., one that scans from A[1] to A[n] in order, can be faster. Explain who is correct.*

   Crabbe is correct. In order to prove this I only have to provide one counter-example. For example, if you're searching fo the value 1 in the array 1, 2, 3, 4, 5, 6, 7, 8, then the binary search will look search 4, 2, and then 1, taking 3 times as many search operations. In general, in an array of length greater than three, linear search will find the first item faster than binary search. There are other cases, but it only takes one because Crabbe claimed, vehemently, that "it is always more efficient to use binary search."

7. *(5 pts) Crabbe has written some code to apply a wizard function w() to some numbers, stored in an input array A[i, j] for 1 i, j n. Assume w() takes O(1) time to compute. Determine the asymptotic running time of Crabbes function.*

Here is the pseudocode for the algorithm:

```
wizardAllTheThings(A) {  // Assigning variable takes O(1) time
    for i = 1 to n {  // Runs n times
        for j = i/2 to n { w(A[i,j]) }  // Runs 2 times sum from 1 to n/2 of n-i
    }
    return
}
```

So, ignoring the constant array initialization, the inner loop runs $2 * \sum_{i=1}^{n}(n-i)$ times. Thus we can expand this to $2 * (n + (n-1) + (n-2) + ... + (n/2)) = 2 * ((n/2) * n - (1 + 2 + ... + n/2))$. Taking into account only the largest term, $(n/2)n$, we get $2((n/2)n = n^2$. Thus the function has O(n^2) run time.

8. *(10 pts) Exercise 3.1-5 in CLRS.*
3.1-5: Prove Theorem 3.1.
Theorem 3.1: For any two function $f(n)$ and $g(n)$, we have $f(n) = \Theta(g(n))$ if and only if $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$.

False. After the following:

$$\lim_{n\to\infty} \frac{3^{2n}}{3^n} = \infty \tag{11}$$

So there is no constant that you can use to change infinity.

9. *(20 pts total) Crabbe and Goyle are now arguing about binary search. Goyle writes the following pseudocode on the board, which he claims implements a binary search for a target value v within input array A containing n elements.*

```
bSearch(A, v) {
    return binarySearch(A, 0, n, v)
}

binarySearch(A, l, r, v) {
    if l >= r then return -1
    p = floor( (l + r)/2 )
    if A[p] == v then return m
    if A[m] < v then
        return binarySearch(A, m+1, r, v)
    else return binarySearch(A, l, m-1, v)
}
```

(a) *Help Crabbe determine whether this code performs a correct binary search. If it does, prove to Goyle that the algorithm is correct. If it is not, state the bug(s), give line(s) of code that are correct, and then prove to Goyle that your fixed algorithm is correct.*

The error in the code is that m is uninitialized. If the middle of $ifA[p] == vthenreturnm$, we switch the name of the variable p to m, which is previously undeclared. Instead the naming should be consistent and keep using p. Also, the first line of the function should be $ifl > rthenreturn - 1$. l can equal r in a number of cases. Also, the algorithm never returns the index. Instead of the else, if should say else if A[p] > v then return binarySea..., and then followed by else return p.

(b) *Goyle tells Crabbe that binary search is efficient because, at worst, it divides the remaining problem size in half at each step. In response Crabbe claims that tri-nary search, which would divide the remaining array A into thirds at each step, would be even more efficient. Explain who is correct and why.*

First, a tri-nary search would need two middle variables, as well as the left and right variables. If v were less than m1 = (l+r)/3, then call the function again and pass in l for the new l and m1-1 for right. If it is greater than m1 and less than m2 = 2*(l+r)/3, then call the function again and pass in m1+1 for l and m2-1 for r. If v is greater than m2, then call the function again and pass in m1+1 for l and r for r. Then check both m1 and m2. So, tri-nary runs in $log_3(n)$ while the binary search runs in $log_2(n)$, so they both run in O(logn), so they both have the same upper bound. If one were to be picky the tri-nary runs in fewer iterations, but uses more variables and has more comparisons per recursion.