CSCI 3104 Spring 2017 Problem Set 1
Samuel Cuthbertson (06/16)

---

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) $n + 3 = O(n^3)$
   True:

$$\begin{aligned}\lim_{n\to\infty} \frac{n+3}{n^3} &= \lim_{n\to\infty} \frac{\frac{\partial}{\partial n}(n+3)}{\frac{\partial}{\partial n}(n^3)} \\ &= \lim_{n\to\infty} \frac{1}{3n^2} \\ &= 0 \\ &\Rightarrow \mathbf{n + 3 = O(n^3)}\end{aligned}$$

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

$$\begin{aligned}\lim_{n\to\infty} \frac{3^{2n}}{3^n} &= \lim_{n\to\infty} \left(\frac{3^2}{3}\right)^n \\ &= \lim_{n\to\infty} 3^n \\ &= +\infty \\ &\Rightarrow \mathbf{3^{2n} = \Omega(3^n)}\end{aligned}$$

   (c) $n^n = \Theta(n!)$ *Help from Mathematica, used for numerically solving the limit.*
   False:

$$\begin{aligned}\lim_{n\to\infty} \frac{n^n}{n!} &= \lim_{n\to\infty} \frac{n^n}{1 * 2 * 3 * 4 * ... * n} \\ &= +\infty \\ &\Rightarrow \mathbf{n^n = \Omega(n!)}\end{aligned}$$

(d) $\frac{1}{3n} = \Omega(1)$

False:

$$\begin{aligned}\lim_{n\to\infty} \frac{\frac{1}{3n}}{1} &= \lim_{n\to\infty} \frac{1}{3n} \\ &= 0 \\ &\Rightarrow \mathbf{\frac{1}{3n} = O(1)}\end{aligned}$$

(e) $\ln^3 n = \Theta(\log^3 n)$ *Help from Luke Mezlar, Matt Maierhofer and Grant Baker.*

True:

$$\begin{aligned}\lim_{n\to\infty} \frac{\ln^3 n}{\log^3 n} &= \lim_{n\to\infty} \frac{(\ln n)^3}{(\log n)^3} \\ &= \lim_{n\to\infty} \left(\frac{\frac{\log n}{\log e}}{\log n}\right)^3 \\ &= \lim_{n\to\infty} 1 \\ &= 1 \\ &\Rightarrow \mathbf{\ln^3 n = \Theta(\log^3 n)}\end{aligned}$$

2. *Simplify the following.*

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

$$\begin{aligned}\frac{d}{dt}(2t^4 + \frac{1}{2}t^2 - 7) &= \frac{d}{dt}(2t^4) + \frac{d}{dt}(\frac{1}{2}t^2) - \frac{d}{dt}(7) \\ &= \mathbf{8t^3 + t}\end{aligned}$$

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

The leap of logic below came from my digital logic class, where the maximum value that can be stored in n bits is the sum of the value of those bits, or $2^{n+1} - 1$. The proof is as follows. For $k = 0$, note:

$$\begin{aligned}\sum_{i=0}^{0} 2^i = 2^0 &= 2^{0+1} - 1 \\ &= 1\end{aligned}$$

Then for the inductive step:

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

Thus, $\sum_{\mathbf{i=0}}^{\mathbf{k}} \mathbf{2^i = 2^{k+1} - 1}$

(c) $\Theta(\sum_{k=1}^{n} \frac{1}{k})$ *Help from Appendix 1, A.7 which was mentioned in lecture.*

$$\begin{aligned}\sum_{k=1}^{n} \frac{1}{k} &= \ln(n) + O(1) \\ \Rightarrow \Theta(\sum_{k=1}^{n} \frac{1}{k}) &= \Theta(\ln(n) + O(1)) \\ \Rightarrow \mathbf{\Theta}(\sum_{\mathbf{k=1}}^{\mathbf{n}} \frac{\mathbf{1}}{\mathbf{k}}) &= \mathbf{\Theta}(\ln(\mathbf{n}))\end{aligned}$$

3. *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.*

   $\mathbf{O(n^2)}$ is an upper bound, and having a running time of "at least $O(n^2)$" implies that $O(n^2)$ is the lower bound for running time. Having a Big-O of 'at least' anything is a nonsensical statement.

4. *Using the mathematical definition of Big-O, answer the following.*

   (a) *Is* $2^{nk} = O(2^n)$ *for* $k > 1$*? Help from Matt Maierhofer.*
   To prove this, we must use induction on the following statement.

$$2^{nk} \leq c_2 2^n \qquad \exists c_2 > 0, \forall n \geq n_0 > 0, \forall k > 1$$

   Starting with a base case of $k = 2$:

$$\begin{aligned} \lim_{n\to\infty} \frac{2^{2n}}{2^n} &= \lim_{n\to\infty} \left(\frac{2^2}{2}\right)^n \\ &= \lim_{n\to\infty} 2^n \\ &= +\infty \\ \Rightarrow 2^{nk} &\neq O(2^n) \qquad k = 2 \end{aligned}$$

   We find that $\mathbf{2^{nk} \neq O(2^n)}$ for $k = 2$, and therefor is also not equal $\forall k > 1$.

   (b) *Is* $2^{n+k} = O(2^n)$ *for* $k = O(1)$*?*
   Following a similar approach to above, we use induction on the following. It's also worth noting that $k = O(1)$ holds true $\forall k > 0$.

$$2^{n+k} \leq c_2 2^n \qquad \exists c_2 > 0, \forall n \geq n_0 > 0, \forall k > 0$$

   Starting with a base case of $k = 1$:

$$\begin{aligned} \lim_{n\to\infty} \frac{2^{n+1}}{2^n} &= \lim_{n\to\infty} \frac{2^n * 2^1}{2^n} \\ &= \lim_{n\to\infty} 2 \\ &= 2 \\ \Rightarrow 2^{n+k} &= \Theta(2^n) = O(2^n) \qquad k = 1 \end{aligned}$$

   Then examining if the inductive step:

$$\begin{aligned} \lim_{n\to\infty} \frac{2^{n+k}}{2^n} * 2^1 &\stackrel{?}{=} \lim_{n\to\infty} \frac{2^{n+k+1}}{2^n} \\ 2 * \lim_{n\to\infty} \frac{2^n * 2^k}{2^n} &\stackrel{?}{=} \lim_{n\to\infty} \frac{2^n * 2^k * 2^1}{2^n} \\ 2 * \lim_{n\to\infty} 2^k &\stackrel{?}{=} \lim_{n\to\infty} 2^k * 2^1 \\ 2 * 2^k &= 2 * 2^k \end{aligned}$$

   Therefor, $\mathbf{2^{n+k}}$ **is bounded by** $\mathbf{O(2^n)}$ and also $\Theta(2^n)$ for positive constant k.

5. *Pseudocode for Linear Search*

```
linearSearch(A, v) {
    for(item from items in A, index of item i) {
        if item == v { return i }
    }
    return NIL
}
```

This solutions satisfies all three parts of a loop invariant:

Initialization: When the function is called, v is somewhere in $A[0...n]$ where n is the highest index in A.
Maintenance: On every iteration of the loop, v is either the current item or is in $A[i...n]$. If it is the current value, then the loop terminates. If it is not, then the loop continues until all items in A have been checked.
Termination: The function either terminates from finding v, as mentioned above, or from exhausting all the items in A and returning NIL.

6. *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.*

   Goyle is correct. The keywords here are 'always' and 'sometimes'. When searching for the lowest value in the array, linear search will always be faster as it completes in $O(1)$. However, when searching for the highest value, binary search is clearly faster as it completes in $O(\log(n))$ rather than $O(n)$. Therefor, Goyle is correct that *sometimes* linear search is faster.

7. *Crabbe has written some code to apply a wizard function* w() *to some numbers, stored in an input array* $A[i, j]$ *for* $1 \leq i, j \leq n$. *Assume* w() *takes* $O(1)$ *time to compute. Determine the asymptotic running time of Crabbe's function. Help from Luke Mezlar.*

```
0   wizardAllTheThings(A) {
1       for i = 1 to n {
2           for j = i/2 to n {
3               w(A[i,j])
4           }
5       }
6       return
7   }
```

The loop on lines 1-5 will run everything inside it n times, and the loop on lines 2-4 will run line 3 somewhere in between $\frac{1}{2}n$ and n times. However, the $\frac{1}{2}$ for this loop has no bearing on the asymptotic behavior of the function, as mentioned in lecture. Finally, since line 3 runs in $O(1)$, we have:

$$wizardAllTheThings(n) = O(n) * O(n) * O(1)$$
$$wizardAllTheThings(n) = \mathbf{O(n^2)}$$

8. *Prove that for any two functions $f(n)$ and $g(n)$, we have $f(n) = \Theta(g(n)) \iff f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$.*

   Note the definition of $f(n) = \Theta(g(n))$:

$$\exists\big((c_1, c_2, n_0) > 0\big) \mid 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n), \forall n \geq n_0$$

   Here I've highlighted the Ω parts and the O parts, and left the parts involved in both as black. The definitions can then clearly be found to ensure the if and only if statement in the question, as shown below:

$$\Omega \to \exists\big((c_1, n_0) > 0\big) \mid 0 \leq c_1 g(n) \leq f(n), \forall n \geq n_0$$
$$O \to \exists\big((c_2, n_0) > 0\big) \mid 0 \leq f(n) \leq c_2 g(n), \forall n \geq n_0$$

9. *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.*

```
0    bSearch(A, v) {
1        return binarySearch(A, 0, n, v)
2    }
3
4    binarySearch(A, l, r, v) {
5        if l >= r then return -1
6        p = floor( (l + r)/2 )
7        if A[p] == v then return m
8        if A[m] < v then
9            return binarySearch(A, m+1, r, v)
10           else return binarySearch(A, l, m-1, v)
11   }
```

(a) *Help Crabbe determine whether this code performs as 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. Help from Luke Mezlar.*

Crabbe's code is not correct. The biggest bug is every mention of 'm' should be replaced with 'p', as m is never initialized. Additionally, on line 5 the $\geq$ should simply be an equals, as there is no situation where $l > r$. Finally, since A is 0 indexed and has n-elements, the highest index is $n - 1$, not n as is on line 1. A correct version is as follows:

```
0    bSearch(A, v) {
1        return binarySearch(A, 0, n-1, v)
2    }
3
4    binarySearch(A, l, r, v) {
5        if l = r then return -1
6        p = floor( (l + r)/2 )
7        if A[p] == v then return p
8        if A[p] < v then
9            return binarySearch(A, p+1, r, v)
```

```
10            else return binarySearch(A, l, p-1, v)
11    }
```

We can see that this version fulfills all three requirements of a loop invariant:
Initialization: Every time binarySearch is called, v is somewhere in $A[l...r]$ or does not exist in A.
Maintenance: On every call of binarySearch, v is either the item under consideration, to the left of the item under consideration, or to the right of the item under consideration. In the first case, v is simply returned. In the second case, binarySearch is called on the portion of A to the left of p, with the initialization cases still holding true. Likewise for the the third case, only the portion of A to the right instead of to the left of p is searched.
Termination: The function either terminates from finding v, as mentioned above, or from exhausting all the items in A (when $l = r$, and the portion of the array being searched is only the item at p) and returning `-1`.

(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 trinary search, which would divide the remaining array A into thirds at each step, would be even more efficient. Explain who is correct and why*

Goyle is correct, because of how binary search is bounded by $O(\log_2 n)$. The base two in that bound is famously given because of how binary search divides whatever it is searching by two at each step, which means that trinary search would be bounded by $O(\log_3 n)$. As shown in Problem 1e, the base of a logarithm has no affect on it's asymtotic behavior, which means trinary search is asymptotically the same as binary search.