CSCI 3104 Spring 2014 — Problem Set 1

Cole Schlisner 01/24

---

1. *(10 pts total) For each of the following claims, determine whether they are true or false. Justify your determination (show your work). If the claim is false, state the correct asymptotic relationship as O, Θ, or Ω.*

   (a) $n+1 = O(n^4)$

$$\begin{aligned}
n+1 = O(n^4) &\Leftrightarrow \lim_{n\to\infty} \frac{n+1}{n^4} = 0 && (1)\\
\lim_{n\to\infty} \frac{n+1}{n^4} &\overset{\text{L'H}}{=} \lim_{n\to\infty} \frac{1}{4n^3} && (2)\\
&= 0 && (3)\\
\therefore n+1 &= O(n^4) && (4)
\end{aligned}$$

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

$$\begin{aligned}
2^{2n} = O(2^n) &\Leftrightarrow \lim_{n\to\infty} \frac{2^{2n}}{2^n} = 0 && (5)\\
\lim_{n\to\infty} \frac{2^{2n}}{2^n} &= \lim_{n\to\infty} \frac{2^n 2^n}{2^n} && (6)\\
&= \lim_{n\to\infty} 2^n && (7)\\
&= \infty && (8)\\
\therefore 2^{2n} &\neq O(2^n) && (9)\\
(8) \implies 2^{2n} &= \Omega(2^n) && (10)
\end{aligned}$$

(c) $2^n = \Theta(2^{n+7})$

$$2^n = \Theta(2^{n+7}) \Leftrightarrow \lim_{n\to\infty} \frac{2^n}{2^{n+7}} = C \tag{11}$$

$$\lim_{n\to\infty} \frac{2^n}{2^{n+7}} = \lim_{n\to\infty} \frac{2^n}{2^n 2^7} \tag{12}$$

$$= \lim_{n\to\infty} \frac{1}{2^7} \tag{13}$$

$$= \frac{1}{128} = C \tag{14}$$

$$\therefore 2^n = \Theta(2^{n+7}) \tag{15}$$

(d) $1 = O(\frac{1}{n})$

$$1 = O(\frac{1}{n}) \Leftrightarrow \lim_{n\to\infty} \frac{1}{(\frac{1}{n})} = 0 \tag{16}$$

$$\lim_{n\to\infty} \frac{1}{(\frac{1}{n})} = \lim_{n\to\infty} n \tag{17}$$

$$= \infty \tag{18}$$

$$\therefore 1 \neq O(\frac{1}{n}) \tag{19}$$

$$(19) \implies 1 = \Omega(\frac{1}{n}) \tag{20}$$

(e) $ln^2(n) = \Theta(lg^2(n))$

$$ln^2(n) = \Theta(lg^2(n)) \Leftrightarrow \lim_{n\to\infty} \frac{ln^2(n)}{lg^2(n)} = C \tag{21}$$

$$\lim_{n\to\infty} \frac{ln^2(n)}{lg^2(n)} \overset{\text{L'H}}{=} \lim_{n\to\infty} \frac{(\frac{2ln(n)}{n})}{(\frac{2ln(n)}{nln^2(2)})} \tag{22}$$

$$= \lim_{n\to\infty} \frac{ln^2(2)ln(n)2n}{ln(n)2n} \tag{23}$$

$$= \lim_{n\to\infty} ln^2(2) \tag{24}$$

$$= ln^2(2) = C \tag{25}$$

$$\therefore ln^2(n) = \Theta(lg^2(n)) \tag{26}$$

(f) $n^2 + 2n - 4 = \Theta(n^2)$

$$n^2 + 2n - 4 = \Omega(n^2) \Leftrightarrow \lim_{n\to\infty} \frac{n^2+2n-4}{n^2} = \infty \tag{27}$$

$$\lim_{n\to\infty} \frac{n^2+2n-4}{n^2} \overset{\text{L'H}}{=} \lim_{n\to\infty} \frac{2n+2}{2n} \tag{28}$$

$$= \lim_{n\to\infty} \frac{n+1}{n} \tag{29}$$

$$= \lim_{n\to\infty} 1 + \frac{1}{n} \tag{30}$$

$$= 1 \tag{31}$$

$$\therefore n^2 + 2n - 4 \neq \Omega(n^2) \tag{32}$$

$$(31) \implies n^2 + 2n - 4 = \Theta(n^2) \tag{33}$$

(g) $3^{3n} = \Theta(9^n)$

$$\begin{aligned}
3^{3n} = \Theta(9^n) &\Leftrightarrow \lim_{n\to\infty} \frac{3^{3n}}{9^n} = C && (34)\\
\lim_{n\to\infty} \frac{3^{3n}}{9^n} &= \lim_{n\to\infty} \frac{3^3 3^n}{9^n} && (35)\\
&= \lim_{n\to\infty} 9\frac{3^n}{9^n} && (36)\\
&= \lim_{n\to\infty} 9\frac{1}{3^n} && (37)\\
&= 0 && (38)\\
\therefore 3^{3n} &\neq \Theta(9^n) && (39)\\
(38) \implies 3^{3n} &= O(9^n) && (40)
\end{aligned}$$

(h) $2^{n+1} = \Theta(2^{nlg(n)})$

$$\begin{aligned}
2^{n+1} = \Theta(2^{nlg(n)}) &\Leftrightarrow \lim_{n\to\infty} \frac{2^{n+1}}{2^{nlg(n)}} = C && (41)\\
\lim_{n\to\infty} \frac{2^{n+1}}{2^{nlg(n)}} &= \lim_{n\to\infty} \frac{2^n 2}{2^{lg(n)^n}} && (42)\\
&= \lim_{n\to\infty} 2\frac{2^n}{n^n} && (43)\\
&= 0 && (44)\\
\therefore 2^{n+1} &\neq \Theta(2^{nlg(n)}) && (45)\\
(44) \implies 2^{n+1} &= O(2^{nlg(n)}) && (46)
\end{aligned}$$

(i) $\sqrt{n} = O(lg(n))$

$$\sqrt{n} = O(lg(n)) \Leftrightarrow \lim_{n\to\infty} \frac{\sqrt{n}}{lg(n)} = 0 \tag{47}$$

$$\lim_{n\to\infty} \frac{\sqrt{n}}{lg(n)} \overset{\text{L'H}}{=} \lim_{n\to\infty} \frac{(\frac{1}{2\sqrt{n}})}{(\frac{1}{nln(2)})} \tag{48}$$

$$= \lim_{n\to\infty} \frac{nln(2)}{2\sqrt{n}} \tag{49}$$

$$= \frac{ln(2)}{2} \lim_{n\to\infty} \frac{n}{\sqrt{n}} \tag{50}$$

$$= \frac{ln(2)}{2} \lim_{n\to\infty} \sqrt{n} \tag{51}$$

$$= \infty \tag{52}$$

$$\therefore \sqrt{n} \neq O(lg(n)) \tag{53}$$

$$(52) \implies \sqrt{n} = \Omega(lg(n)) \tag{54}$$

(j) $10^{100} = \Theta(1)$

$$10^{100} = \Theta(1) \Leftrightarrow \lim_{n\to\infty} \frac{10^{100}}{1} = C \tag{55}$$

$$\lim_{n\to\infty} \frac{10^{100}}{1} = 10^{100} = C \tag{56}$$

$$\therefore 10^{100} = \Theta(1) \tag{57}$$

2. *(15 pts) Professor Dumbledore needs your help optimizing the Hogwarts budget. Youll be given an array A of exchange rates for muggle money and wizard coins, expressed at integers. Your task is help Dumbledore maximize the payoff by buying at some time i and selling at a future time j ¿ i, such that both A[j] ¿ A[i] and the corresponding difference of A[j] A[i] is as large as possible. For example, let A = [8, 9, 3, 4, 14, 12, 15, 19, 7, 8, 12, 11]. If we buy stock at time i = 2 with A[i] = 3 and sell at time j = 7 with A[j] = 19, Hogwarts gets in income of 19 - 3 = 16 coins.*

   (a) *What is the running time complexity of the [given pseudocode]? Write your answer as a bound in terms of n.*

   The procedure exhausts both loops without breaking $\implies$ it makes (n-1) comparisons for the first value, (n-2) comparisons for the second, etc... for a total of

   $$\sum_{i=1}^{n-1} i = \frac{n(n-1+1)}{2} = \frac{n^2}{2}$$

   comparisons. Additionally, each comparison takes $\Theta(1)$ time. Therefore the whole operation takes $\Theta(n^2)$ time.

   (b) *Explain (1 - 2 sentences) under what conditions on the contents of A the makeWizardMoney algorithm will return 0 coins.*

   Either:

   - There is only 1 element in array A, in which case no comparisons can be made (in fact the second loop will never be entered). The initial value (0) of maxCoinsSoFar will be returned.
   - A is in strictly non-increasing order. The value of *coins* will always be less than 0, in which case the initial value (0) of maxCoinsSoFar will be returned.

   (c) *Dumbledore knows you know that makeWizardMoney is wildly inefficient. With a wink, he suggests writing a function to make a new array M of size n such that:*

   $$B[i] = \min_{0 \le j \le i} A[j].$$

   That is, M[i] gives the minimum value in the subarray of A[0..i]. What is the running time complexity of the pseudocode to create the array M? Write your answer as a bound in terms of n.

   No more than n comparisons are needed for this operation (as it boils down to comparing A[i] to M[i-1]) $\implies$ the complexity of this algorithm is $\Theta(n)$

(d) *Use the array M computed from (2c) to compute the maximum coin return in time (n).*

```
maxReturn = 0
for i in [1 .. A.length]
if (A[i] - M[i]) > maxReturn
maxReturn = A[i]-M[i]
return maxReturn
```

(e) *Use the array M computed from (2c) to compute the maximum coin return in time (n).*

```
maxRet = 0
for i in [1 .. A.length]
    if (A[i] - M[i]) > maxRet
        maxRet = A[i]-M[i]
return maxRet
```

(f) *Give Dumbledore what he wants: rewrite the original algorithm in a way that combines parts (2b)-(2d) to avoid creating a new array M.*

```
min = A[1]
maxRet = 0
for i in [1 .. A.length]
    if (A[i] < min)
        min = A[i]
    elseif (A[i] - min) > maxRet
        maxRet = (A[i] - min)
return maxRet
```

3. *(15 pts) Consider the problem of linear search. The input is a sequence of n numbers A = ha1, a2, . . . , ani and a target value v. The output is an index i such that v = A[i] or the special value NIL if v does not appear in A.*

   (a) *Write pseudocode for a simple linear search algorithm, which will scan through the input sequence A, looking for v.*

   ```
   for i in [1 .. A.length]
       if (A[i] = v)
           return i
   return NIL
   ```

   (b) *Using a loop invariant, prove that your algorithm is correct. Be sure that your loop invariant and proof covers the initialization, maintenance, and termination conditions.*

   Loop invariant: A[i] = v $\lor$ v $\notin$ A[1 .. i-1]

   Initialization: before the loop begins, $i = 1$. If A[1] = v, then both parts of the loop invariant hold. If A[1] $\neq$ v, then the loop invariant holds as $i - 1 = 0$ and the subset A[1 .. 0] is the empty set, which cannot contain v by definition.

   Maintenance: during each loop, we check if A[i] = v. In such case, the loop invariant holds and the function returns i. If A[i] $\neq$ v, we increment i for the next iteration and the second half of the loop invariant holds - A[1 .. i-1] does not contain v (i-1 now being the index of the value we last checked).

   Termination: when the loop exits, etiher:

   - an integer value was returned - in which case A[i] = v, satisfying the loop invariant
   - the loop condition was not met, indicating i = A.length + 1, meaning v was not equal to any value in A[1 .. i-1] = A. This satisfies the loop invariant and consequently NIL will (correctly) be returned.

4. *(15 pts) Crabbe and Goyle are 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.*

   (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.*

   Bugs:

   The line: *binarySearch(A, 1, n-1, v)* should be: *binarySearch(A, 1, n, v)*

   The line: *if* $l <= r$ should be: *if* $r <= l$

   Loop invariant: $r - l > 0 \lor v \notin A[l..r]$

   Initialization:
   When the program enters binarySearch(): $r - l = n - 1 \geq 0$

   Maintenance:
   If $r - l \leq 0 \implies$ return -1 $\implies v \notin$ A[l .. r] as A[l .. r] is the empty set. The invariant holds.
   If v $\in$ A[l .. r], $r - l > 0 \implies$ the invariant holds, return index of v
   These two exit points are where the 'loop' actually ends.
   The quantity r-l is then decreased by 1 and the function returns the value of a call to itself with the updated parameters.

   Termination:
   If -1 was returned, $v \notin$ (A[l .. r] = the empty set) $\implies$ the invariant holds
   If an integer i was returned, v $\in$ A[l .. r] $\implies r - l > 0 \implies$ the invariant holds.

   (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 four-nary search, which would divide the remaining array A into fourths at each step, would be way more efficient. Explain who is correct and why.*

   Goyle is correct. Four-nary search would need to check twice as many values/boundaries, even though it would be cutting the problem size in half.

5. *(10 pts extra credit) You are given two arrays of integers A and B, both of which are sorted in ascending order. Consider the following algorithm for checking whether or not A and B have an element in common.*

   (a) *If arrays A and B have size n, what is the worst case running time of the procedure findCommonElement? Provide a bound.*

   In the worst case, the arrays do not have an element in common, and both loops are exhausted. This means n^2 comparisons are made $\implies$ the procedure has a running time of $\Theta(n^2)$.

   (b) *For n = 5, describe input arrays A_1, B_1 that will be the best case, and arrays A_2, B_2 that will be the worst case for findCommonElement.*

   A_1 = [0,1,2,3,4]
   B_1 = [0,1,2,3,4]

   A_2 = [0,1,2,3,4]
   B_2 = [5,6,7,8,9]

   (c) *Write pseudocode for an algorithm that runs in (n) time for solving the problem. Your algorithm should use the fact that A and B are sorted arrays. (Hint: repurpose the merge procedure from MergeSort.)*

```
findCommonElement(A, B):
    ai = bi = 0
    while (bi < len(B) and ai < len(A)):
        if (A[ai]==B[bi]):
        return True
        if (A[ai] < B[bi]):
            ai += 1
        elif (B[bi] < A[ai]):
            bi += 1
    return False
```

References Used

1. CLRS
2. https://en.wikipedia.org/wiki/Merge_sort