CSCI 3104 Spring 2018 Problem Set #1
Alison Ostlund (05/22)
Worked with: Kyle Ward, Denver Chernin, Hannah Haines, Matt Donavon

---

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

   $\lim_{n\to\infty} (\frac{n+1}{n^4}) = L'Hospital's \lim_{n\to\infty} (\frac{1}{4n^3}) = \frac{1}{\infty} = 0$
   True, $n + 1 = O(n^4)$

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

   $\lim_{n\to\infty} (\frac{2^{2n}}{2^n}) = \lim_{n\to\infty} (\frac{2^2}{2})^2 = \lim_{n\to\infty} (2^n) = \infty$
   False, $2^{2n} = \Omega(2^n)$

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

   $\lim_{n\to\infty} (\frac{2^n}{2^{n+7}}) = \lim_{n\to\infty} (\frac{2^n}{2^n \cdot 2^7}) = (2^7) = \frac{1}{128}$
   True, $2^{2n} = \Theta(2^n)$

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

   $\lim_{n\to\infty} (\frac{1}{\frac{1}{n}}) = \lim_{n\to\infty} (n) = \infty$
   False, $1 = \Omega(\frac{1}{n})$

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

$c_1 \cdot \log^2(n) \leq \ln^2(n) \leq c_2 \cdot \log^2(n) = 5 \cdot \log^2(n) \leq \ln^2(n) \leq 6 \cdot \log^2(n)$
True, by the definition of Big theta $\ln^2(n) = \Theta(\log^2(n))$

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

$$\lim_{n\to\infty} (\frac{n^2+2n-4}{n^2}) = L'Hospital's \lim_{n\to\infty} (\frac{2n+2}{2n}) = L'Hospital's \lim_{n\to\infty} (\frac{2}{2}) = 1$$

False, $n^2 + 2n - 4 = \Omega(n^2)$,but can also be True because Ω is included in Θ

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

$$\lim_{n\to\infty} (\frac{3^{3n}}{9^n}) = \lim_{n\to\infty} (\frac{3^3}{9})^n = \lim_{n\to\infty} (3)^n = \infty$$

False, $3^{3n} = \Omega(9^n)$

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

$$\lim_{n\to\infty} (\frac{2^{n+1}}{2^{n\log(n)}}) = 2 \cdot \lim_{n\to\infty} (\frac{2^n}{2^{n\log(n)}}) = 2 \cdot \lim_{n\to\infty} (\frac{2}{2^{\log(n)}})^n = 2 \cdot \lim_{n\to\infty} (\frac{2}{2^n})^n = 0$$
False, $3^{n+1} = O(2^{n\log(n)})$

(i) $\sqrt{n} = O(\log(n))$
$\sqrt{n} \geq c_1 \cdot \log(n)$ for all values $c_1 > 1$
False, $\sqrt{n} = \Omega(\log(n))$

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

$$\lim_{n\to\infty} (\frac{10^{100}}{1}) = 10^{100}$$
True, $10^{100} = \Theta(1)$

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.

(a) Consider the pseudocode below that takes as input an array A of size n.What is the running time complexity of the procedure above? Write your answer as a Θ bound in terms of n.

$T(n) \leq c_1 + c_2 + c_3 + c_4 + n \cdot n$

$T(n) \leq c_5 n^2 = O(n^2)$ $T(n) = \sum_{i=1}^{n} \sum_{j=i+1}^{n} C = C \cdot \sum_{i=1}^{n} \sum_{j=i}^{n-1} 1 = C(\frac{n(n-1}{2}) = \Theta(n^2)$

(b) Explain (12 sentences) under what conditions on the contents of A the $makeWizardMoney$ algorithm will return 0 coins.

The contents of the array A would need to be in descending order to have the algorithm return 0 coins. This will cause the contents in A able to return a negative number, for example, 8-19 = -9, therefore the algorithm will never hit the line if(coins > maxCoinsSoFar) causing 0 to be the return value.

(c) What is the running time complexity of the pseudocode to create the array M? Write your answer as a Θ bound in terms of n.

The array A contains [8,9,3,4,14,12,15,19....] where 3 is the minimum in the array. Therefore the array M would compare two position from each sub array for the minimum number so for M[0] = min A[0] then M[1] = min (A[0],A[1]), which would cause the array to contain [8,8,3,3,3...]. This would be an $\Theta(n)$ complexity because the algorithm is a single loop that takes the minimum atomic operations.

(d) Use the array M computed from C(n) to compute the maximum coin return in time $\Theta(n)$.

The maximum number in the array A[] is 19. So that implies that in array M there is an iteration i, M[i] that includes 19 from array A that will cause the maxCoinsSoFar to return 16, 19-3 = 16.

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

```
makeWizardMoney(A):
  maxCoinsSoFar = 0
```

```
    minValue = A[0]
    for i = 1 to length(A):
      if( A[i] ≥ minValue):
      coins = A[i] - minValue
      if( coins > maxCoinsSoFar){maxCoinsSoFar = coins}
  return maxCoinsSoFar
```

3. (15 pts) Consider the problem of linear search. The input is a sequence of n numbers A = $(a_1, a_2, ..., a_n)$ 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.

```
linearSearch(A,v):    for ( i = 0 to length(A)):
      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: if A[i] != v then subarray A[0,1..,i] does not contain v.
   Initialization: i = 0 and A[i] != v then array A does not have v in it.
   Maintenance: i = i+1 check if A[i] = v, if it not, then array A doesn't contain v.
   Termination: Return i if A[i] = v else return NIL.

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.

incorrect line: return binarySearch(A, 1, n-1, v)
fix: return binarySearch(A, 0, n-1, v)
incorrect line: if l $\leq$ r then return -1
fix: if l > r then return -1

Initialization: if A[p] != v and l $\leq$ r then the if(A[p] < v) statement will recurse with v+1 as l. Else the algorithm will recurse with p-1 as r.
Maintenance: update p and check if A[p] != v, if A[p] < v then recurse with p + 1 as l or recurse with p -1 as l.
Termination: if A[p] = v then return p, if l > r return -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 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.

Binary Search = $O(\log_2(n)) = \log(n)$
Four-nary Search = $O(\log_4(n))\log(n)$
$\lim_{n\to\infty} (\frac{\log(n)}{\log_2(n)}) = \ln(2) \therefore \log_2(n) = \Theta(\log(n))$
$\lim_{n\to\infty} (\frac{\log(n)}{\log_4(n)}) = 2\ln(2) \therefore \log_4(n) = \Theta(\log(n))$
Since both $\log_2(n)$ and $\log_4(n)$ both equal $\Theta(\log(n))$ then both binary search and four-nary search are equally efficient.

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 Θ bound.
$T(n) = n \cdot n + c_1$
$T(n) = c_1 + n^2$
$T(n) = \Theta(n^2)$

(b) For n = 5, describe input arrays A1, B1 that will be the best case, and arrays A2,

B2 that will be the worst case for findCommonElement.

The best case for A_1 and B_1 would be for both A[0] and B[0] to be equal at the first position. There would be a common element in the first iteration, causing a complexity of O(1). The worst case would be having both A[4] and B[4] be the position that the element is found. This is because the algorithm will have to run through the code multiple times and over each element in the arrays until it found a common element.The worst case could also be if there was no common element in neither A[i] or B[j]. This will also cause the algorithm to run through multiple iterations and go through each position of the array, causing a complexity of O(n^2).

(c) Write pseudocode for an algorithm that runs in $\Theta(n)$ 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):
    i,j = 0
    while ( i < n and j < n):
      if( A[i] < B[j]): {i++}
      else if( A[i] > B[j]): {j++}
      else: {return True}
    return False
```