Name: Jon Abrahamson
ID: 107084898

CSCI 3104, Algorithms **Profs. Grochow & Layer**
Problem Set 1 **Spring 2019, CU-Boulder**

Advice 1: For every problem in this class, you must justify your answer: show how you arrived at it and why it is correct. If there are assumptions you need to make along the way, state those clearly.

Advice 2: Verbal reasoning is typically insufficient for full credit. Instead, write a logical argument, in the style of a mathematical proof.

Hyperlinks for convenience: 1a 1b 1c 1d 1e 1f 1g 1h 1i 1j 2a 2b 2c 2d 2e 3a 3b 4a 4b

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

   True

   $\lim_{n\to\infty} f(n)/g(n) = L$
   $f(n) = n^2 + 2n - 4, \qquad g(n) = n^2$
   $\lim_{n\to\infty} \dfrac{n^2 + 2n - 4}{n^2} = L$ Divide by n^2
   $\lim_{n\to\infty} \dfrac{1 + \frac{2}{n^2} - \frac{4}{n^2}}{1} = \lim_{n\to\infty} 1 + \dfrac{2}{n^2} - \dfrac{4}{n^2} = 1 + 0 + 0 = 1$
   $L = 1$
   Thus the claim $n^2 + 2n - 4 = \Omega(n^2)$ is proven to be True

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

True

$\lim_{n\to\infty} f(n)/g(n) = L$
$f(n) = 10^{100}, \qquad g(n) = 1$
$\lim_{n\to\infty} \frac{10^{100}}{1} = L$
$L = 10^{100}$
Thus, $\quad 10^{100} = \Theta(1)$

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

True

$\lim_{n\to\infty} f(n)/g(n) = L$
$f(n) = ln^2 n \qquad g(n) = lg^2 n$
$\lim_{n\to\infty} \frac{ln^2 n}{lg^2 n} = \lim_{n\to\infty} \frac{ln^2 n}{\frac{ln^2 n}{ln^2(2)}} = \lim_{n\to\infty} ln^2(2)$
$L = ln^2(2)$
Thus the claim that $\ln^2 n = \Theta(\lg^2 n)$ is true

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

True

$\lim_{n\to\infty} f(n)/g(n) = L$
$f(n) = 2^n \qquad g(n) = 2^{n+7}$
$\lim_{n\to\infty} \frac{2^n}{2^{n+7}} = L$
Exponent Rule: $\frac{x^a}{x^b} = \frac{1}{x^{b-a}}$
$\lim_{n\to\infty} \frac{2^n}{2^{n+7}} = \lim_{n\to\infty} \frac{1}{2^{(n-7)-n}} = \lim_{n\to\infty} \frac{1}{2^7} = \lim_{n\to\infty} \frac{1}{128}$
$\lim_{n\to\infty} \frac{1}{128} = \frac{1}{128} = L$
Thus, $2^n = \Theta(2^{n+7})$

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

True

$\lim_{n\to\infty} f(n)/g(n) = L$
$f(n) = n + 1 \qquad g(n) = n^4$
$\lim_{n\to\infty} \frac{n+1}{n^4} = L$
$\lim_{n\to\infty} \frac{1+0}{4 * n^3} = \frac{1}{4 * n^3} = 0$
$L = 0$
Thus, $n + 1 = O(n^4)$

(f) $1 = O(1/n)$

False

$\lim_{n\to\infty} f(n)/g(n) = L$
$f(n) = 1 \qquad g(n) = 1/n$
$\lim_{n\to\infty} \frac{1}{1/n} = L$
$\lim_{n\to\infty} n = \infty$
$L = \infty$
Thus $1O(1/n)$ So the claim is false
Correct: $1 = \Omega(1/n)$

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

False

$\lim_{n\to\infty} f(n)/g(n) = L$
$f(n) = 3^{3n} \qquad g(n) = 9^n$
$lim_{n\to\infty} \frac{3^{3n}}{9^n} = L$
$lim_{n\to\infty} \frac{27^n}{9^n} = lim_{n\to\infty}(\frac{27}{9})^n = lim_{n\to\infty} 3^n = L$
$lim_{n\to\infty} 3^n = \infty$
$\infty = L$
Thus, $3^{3n}\Theta(9^n)$ so the claim is false
Correct Claim: $3^{3n} = \Omega(9^n)$

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

False

$\lim_{n\to\infty} f(n)/g(n) = L$
$f(n) = 2^{2n} \qquad g(n) = 2^n$
$lim_{n\to\infty} \frac{2^{2n}}{2^n} = lim_{n\to\infty} \frac{2^n * 2^n}{2^n} = lim_{n\to\infty} 2^n = L$
$lim_{n\to\infty} 2^n = \infty$
$L = \infty$
Thus $2^{2n} O(2^n)$ So the claim is false
Correct Claim: $2^{2n} = \Omega(2^n)$

(i) $2^{n+1} = \Theta(2^{n \lg n})$

False

$\lim_{n\to\infty} f(n)/g(n) = L$
$f(n) = 2^{n+1} \qquad g(n) = 2^{nlg(n)}$
$lim_{n\to\infty} \frac{2^{n+1}}{2^{nlg(n)}} = L$
Apply Exponent Rule: $\frac{n^a}{n^b} = n^{a-b}$
$\lim_{n\to\infty} \frac{2^{n+1}}{2^{nlg(n)}} = \lim_{n\to\infty} 2^{(n+1)-nlg(n)}$
$\lim_{n\to\infty} 2^{(n+1)-nlg(n)} = \lim_{n\to\infty} 2^{n+1} * 2^{-nlg(m)} = \lim_{n\to\infty} (2^{lg(n)})^{-n} * 2^{n+1} =$
$\lim_{n\to\infty} n^{-n} * 2^{n+1} = \lim_{n\to\infty} 2^n * n^{-n} = 2 * \lim_{n\to\infty} \frac{2^n}{n^n}$
$n^n \geq 2^n, n \geq 2$
Thus, $2 * \lim_{n\to\infty} \frac{2^n}{n^n} = 0$
Thus $2^{n+1} \neq \Theta(2^{nlg(n)})$
Correct claim: $2^{n+1} = O(2^{nlg(n)})$

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

False

$\lim_{n\to\infty} f(n)/g(n) = L$
$f(n) = \sqrt{n} \qquad g(n) = lg(n)$
$lim_{n\to\infty}\frac{\sqrt{n}}{lg(n)} = L$
Apply L'Hospital's Rule:
$lim_{n\to\infty}\frac{\frac{1}{2\sqrt{n}}}{\frac{1}{nln(2)}} = lim_{n\to\infty}\frac{1}{2}ln(2)\sqrt{n} = lim_{n\to\infty}\frac{ln(2)\sqrt{n}}{2}$
$\frac{ln(2)}{2}lim_{n\to\infty}\sqrt{n} = lim_{n\to\infty}\frac{ln(2)}{2} * \sqrt{\infty} = \infty$
L=∞ Thus $\sqrt{n}O(lg(n))$
Correct Claim: $\sqrt{n} = \Omega(lg(n))$

2. *(15 pts) Professor Dumbledore needs your help optimizing the Hogwarts budget. You'll be given an array A of exchange rates for muggle money and wizard coins, expressed as 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) *Consider the pseudocode below that takes as input an array A of size n:*

```
makeWizardMoney(A) :
    maxCoinsSoFar = 0
    for i = 0 to length(A)-1 {
        for j = i+1 to length(A) {
            coins = A[j] - A[i]
            if ($coins > maxCoinsSoFar$) { maxCoinsSoFar = coins }
    }}
    return maxCoinsSoFar
```

   What is the running time complexity of the procedure above? Write your answer as a Θ bound in terms of n.

   makeWizardMoney(a) : $\Theta(1)$
   maxCoinsSoFar = 0 $\Theta(1)$
   for i=0 to length(A)-1 $\Theta(n^2)$
   for j= i+1 to length(A) $\Theta(n^2)$

   ...

   Each action performed in the code is of $\Theta(1)$ except for the two nested for loops of $\Theta(n)$ so $4 * \Theta(1) * \Theta(n) * \Theta(n) = \Theta(n^2)$
   so $A = \Theta(n^2)$, n = size of A

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

The makeWizardMoney algorithm will return 0 if the list is not sorted by increasing amount. Because of $A[j] < A[i]$ then the coins value will be negative despite having different sized coins.

(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*

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

That is, $M[i]$ gives the minimum value in the subarray of $A[0..i]$.

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

Compute-M
“‘M[0] = A[0]
“‘for $i = 1$ to length(A)-1
“““M[i] = MinimumOf(M[i-1], A[i])
“‘return M

All actions of complexity $\Theta(1)$ except a for loop equal to $\Theta(n)$. Thus the complexity for Compute-M is $\Theta(n)$

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

Psuedo Code:
ComputeMax(M, A)
“maxA = 0
“minM = M[0]
“for i=0 to length(a)-1
““““if $A[i] > maxA$ then maxA=A[i]
““““elseif $M[i] < minM$ then minM=M[i]
“max=maxA-minM
“return max

This also has time complexity $\Theta(n)$

(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): “‘max=0
“‘min=0
“‘for i=0 to length(A)-1:
““““if $A[i] < min$ then min=i
““““if $A[i] > max$ then max=i
“‘MaxcoinsSoFar = A[max]-A[min]
“‘return MaxcoinsSoFar

Name: Jon Abrahamson
ID: 107084898

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

```
VinA(A, n, v):
“‘index = NIL
“‘for int i=0 to length n:
“““if A[i] == v:
“““““index = i
“““““break
“‘return index
```

(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.*

the variable index is the return value and will return either its initialized state or whatever the for loop sets it equal to if the correct conditions are met.

Initialization condition: index remains constant unless altered by the for loop. It is set to NIL to begin and will remain constant unless v is found within array A. NIL is a possible outcome result and is assumed before the primary emphasis of the search algorithm.

Maintenance Condition: index will contain the value NIL until v is found within array A. It is a valid output before the loop is ran and can still be valid after all the iterations of the loop if the loops conditions do not succeed.

Termination Condition: The only possibilities for index to return are NIL if the conditions of the loop are not met or the index of where v is located first within array A if the conditions of the loop are met.

VinA(A, n, v):
“‘index = NIL ***index is initialized to a true and possible outcome
“‘for int i=0 to length n:
“““‘if A[i] == v: *** loop must meet condition or else value is maintained
“““““‘index = i
“““““‘break *** if condition is met, index updates and loop is terminated
“‘return index *** index value is returned and function terminates

4. *(20 pts) Ron and Hermione are arguing about binary search. Hermione writes the following pseudocode on the board, which she claims implements a binary search for a target value* `v` *within input array* `A` *containing* n *elements.*

```
bSearch(A, v) {
   return binarySearch(A, 1, n-1, v)
}

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

   (a) *Help Ron determine whether this code performs a correct binary search. If it does, prove to Hermione 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 Hermione that your fixed algorithm is correct.*

   The binary search algorithm is incorrect. The first issue arises on the first line of the function. The algorithm first checks its base cases to see if $l \leq r$ and if this is true than the function is terminated and -1 is returned. However this is incorrect because this is exactly what we want to being the search algorithm. We want the left (l) to be less than or equal to the right side (r) value. Otherwise the code as written suggests that the left pointer has moved beyond the right pointer and the value v has not been found in the array. Additionally there is an issue in the bSearch function, more specifically in the parameters of the return binarySearch line. Assuming that the code is written in one of the languages taught to us thus far, the algorithm should be 0-indexed. In the search algorithm, the parameters give the starting index to be 1 instead of 0. This would then skip the first value in array A which could possibly be the value the algorithm is searching for.

   Assuming all issues are fixed, We can now prove to Hermione that the algorithm is correct:
   - This is a recursion algorithm
   - if the algorithm makes it past the base case $l \leq r$, than it will return the answer

if found.
- the middle of the array is found first represented by $p = floor((1 + r)/2)$ which find the median value of using r as the right largest bound.
- after finding the mid point we check to see if we should look first to see if the value at the midpoint is equal to the value v we are looking for. If not then we must look to the left or the right of the mid point for our value v.
- this is determined by the line $if A[p] < v$. if this is true than we know that the value v is farther right than the midpoint index. The function is then called again but now the array A is restricted to the right half of the data.
- If it is not on the right side of the midpoint than the algorithm determines that it must be on the left side of the midpoint index. Thus, similarly, the algorithm is called recurrsively buty array A is restricted to the data on the left side of the midpoint index.
- The algorithm continues to minimize the sample size until $A[p] == v$ and thus the value is found.
- If the value is never found then evenutally $l \geq r$ and if we correctly fixed the first line of the search algorithm, thius should return -1, the value not found.

(b) *Hermione tells Ron that binary search is efficient because, at worst, it divides the remaining problem size in half at each step. In response Ron 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.*

First we will find the time complexity of the binary search algorithm:
essentially we are looking for $1 = \frac{N}{2^x}$ or how many timnes you can divide N before getting 1. This simplifies to: $2^x = N$.
now we take the lg of both sides of the equation: $lg(2^x) = lg(N)$
$x * lg(2) = lg(N)$
$x * 1 = lg(N)$
thus you have a time complexity of lg(N) + O(1)
If you have, like Ron said, a four-nary search than you have complexity of $2 * \log_4(n) + O(1)$
It can then be seen that $2 * \log_4(n) + O(1) = 2 * \frac{log(2)}{log(4)} lg(n) + O(1)$
we know that $2 * \frac{log(2)}{log(4)} > 1$, so we are getting more comparisons than from a binary search
This can be generalized into the form $(k-1) * \frac{log(2)}{log(k)}$ where as k continues to increase, the time complexity continues to increase as well. So, Hermione is right in this case in saying that a binary search algorithm will be more efficient than a four-ary search algorithm.