CSCI 3104-300 Summer 2019 Problem Set 2
Hoffman, Ryan
01/11

1. (10 points) 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. Unless otherwise specified, lg is log_2.

   (a) $n+1 = O(n^4)$ This is true.
   (b) $2^{2n} = O(2^n)$ This is false, $f(n) = \Omega(g(n))$
   (c) $2^n = \Theta(2^{n+7})$ This is true.
   (d) $1 = O(1/n)$ This is false, $f(n) = \Omega(g(n))$
   (e) $\ln^2 n = \Theta(\lg^2 n)$ This is true.
   (f) $n^2 + 2n - 4 = \Omega(n^2)$ This is false, $f(n) = \Theta(g(n))$
   (g) $3^{3n} = \Theta(9^n)$ This is false, $f(n) = \Omega(g(n))$
   (h) $2^{n+1} = \Theta(2^{n \lg n})$ This is false, $f(n) = \mathcal{O}(g(n))$
   (i) $\sqrt{n} = O(\lg n)$ This is false, $f(n) = \Omega(g(n))$
   (j) $10^{100} = \Theta(1)$ This is true.

2. (25 points) Asymptotic relations like O, Ω, and Θ represent relationships between *functions*, and these relationships are transitive. That is, if some $f(n) = \Omega(g(n))$, and $g(n) = \Omega(h(n))$, then it is also true that $f(n) = \Omega(h(n))$. This means that we can sort *functions* by their asymptotic growth.
Sort the following *functions* by order of asymptotic growth such that the final arrangement of functions $g_1, g_2, \ldots, g_{12}$ satisfies the ordering constraint $g_1 = \Omega(g_2)$, $g_2 = \Omega(g_3)$, $\ldots$, $g_{11} = \Omega(g_{12})$.

n	n^2	$(\sqrt{2})^{\lg n}$	$2^{\lg n}$	$n!$	$(\lg n)!$	$\left(\frac{3}{2}\right)^n$	$n^{1/\lg n}$	$n \lg n$	$\lg(n!)$	e^n	42

Give the final sorted list and identify which pair(s) functions $f(n), g(n)$, if any, are in the same equivalence class, i.e., $f(n) = \Theta(g(n))$.

Solution:

42	n	$n^{1/\lg n}$	$(\sqrt{2})^{\lg n}$	$2^{\lg n}$	$\lg(n!)$	$n \lg n$	n^2	$\left(\frac{3}{2}\right)^n$	$n!$	e^n	$(\lg n)!$

Name	Running Time
Constant time	$\mathcal{O}(1)$
Log-logarithmic	$\mathcal{O}(n \log n)$
Logarithmic time	$\mathcal{O}(\log n)$
Fractional power	$\mathcal{O}(n^c)$, where $0 < c < 1$
Linear time	$\mathcal{O}(n)$
"n log star n" time	$\mathcal{O}(n \log^* n)$
Quasilinear time	$\mathcal{O}(n \log n)$
Quadratic time	$\mathcal{O}(n^2)$
Polynomial time	$poly(n) = 2^{\mathcal{O}(\log n)}$
Exponential time(with linear exponent)	$2^{\mathcal{O}(n)}$
Exponential time	$2^{poly(n)}$
Factorial time	$\mathcal{O}(n!)$

3. (30 points) 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 to 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 one stock at time $i = 2$ (assuming 0 indexing) with $A[i] = 3$ and $j = 7$ with $A[j] = 19$, Hogwarts gets an income of $19 - 3 = 16$ coins.

   (a) Consider the pseudocode below that takes as input an array A of size n:

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

   What is the running time complexity of the procedure above? Write your answer as a Θ bound in terms of n.
   Solution:
   The running time complexity for the procedure above is $\Theta(n^2)$.

   (b) Explain (1–2 sentences) under what conditions on the contents of A the `makeWizardMoney` algorithm will return 0 gold.
   Solution:
   If the array, A, is sorted from greatest to least or if each element is the same value(i.e. A[i]=2 for i $\geq$ 0), the assignment of coins=A[j] - A[i] will always be 0 or negative and prevent maxProfitSoFar from incrementing. It will remain at 0.

(c) Professor Dumbledore knows you know that `makeWizardMoney` is wildly inefficient. He suggests you write 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 $A[0..i]$.
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.
Solution:

```
def makeWizardMoney(A)
M=[]
maxProfitSoFar = 0
minVal = A[0]
for i=0 to len(A)-1):
if A[i] > minVal:
M.append(minVal)
else:
M.append(A[i])
minVal=A[i]
for j=0 to len(A)-1:
if A[j] > M[j]:
maxProfitSoFar=A[j]-M[j]
return maxProfitSoFar
```

The running time complexity for the pseudocode to create M is $\Theta(n)$.

(d) Use the template code provided and implement the function described in (3c) to compute the maximum coin difference in time $\Theta(n)$.
Solution:
See python code appended to this solution file.

(e) Use the template code provided to determine and compare the runtimes for the functions in 2a and 2d. Explain your findings.
Solution:
Upon running the code, I recieved a value of 246. The runtime for the improved algorithm is more efficient and has $\Theta(n)$ (linear) in contrast with the original $\Theta(n^2)$ runtime.

4. (15 points) Consider the problem of finding the maximum element in a given array. The input is a sequence of n numbers $A = \langle a_1, a_2, \ldots, a_n \rangle$. The output is an index i such that $a_i \geq a_1 \geq a_2 \geq ... \geq a_n$.

   (a) Write pseudocode for a simple maximum element search algorithm, which will scan through the input sequence A, and return the index of the last occurrence of the maximum element.
   Solution:

```
set maxElement to A[0]
for i = 1 to len(A) - 1
if A[i] > maxElement
maxElement=A[i]
return maxElement
```

   (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.
   Solution:
   Loop Inavariant: At the start of the each iteration of the for loop, maxElement is the largest element in the given array.
   Initialization:(j=0)
   Prior to the first iteration of the loop, we have k=p. The subarray A[p..k-1] is empty. The initial value of i is 1, and this occurs when j = 0, (i.e., the loop has iterated 0 times), and we have i=j+1. maxElement is the largest element and our loop invariant holds.
   Maintenance:(i=j+1)
   Assume the loop invariant holds for (i=j-1). When i=j we check to see if the element at index i is greater than the current maxElement.
   If true, we set the element at that index to maxElement.
   Otherwise, we return maxElement.
   Termination:(i=n)
   If the maxElement exists in the subarray A[0..n-1], we return it. Hence the algorithm is correct.

5. (20 points) 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 an input array `A` containing n elements sorted in ascending order.

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

binarySearch(A, l, r, v) {
if l <= r then return -1
m = floor( (l + r)/2 )
if A[m] == 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 does not, state the bug(s), fix the line(s) of code that are incorrect, and then prove to Goyle that your fixed algorithm is correct.
Solution:
This function terminates the search unsuccessfully when the range is empty (i.e., high ¿ low) and terminates successfully if the value, v, is found.

(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 fournary search, which would divide the remaining array A into fourths at each step, would be *way more* efficient asymptotically. Explain who is correct and why.
Solution:
Further division of the subarray is not necessary and would not contribute to any gain in efficiency. It would actually decrease efficiency because you are doing more comparisons at that level.