CSCI 3104 PS2a

Luna Mcbride

TOTAL POINTS

11 / 12

QUESTION 1

6 pts

1.1 2 / 2

✓ + 1.5 pts **Sufficient reasoning showing how C and n_0 were obtained.**

✓ + 2 pts **Correct values of C and n_0**

+ 1 pts Correct value for one constant with reasoning

+ 0 pts Incorrect

1.2 2 / 2

✓ + 1 pts **Correct value for n0**

✓ + 1 pts **Correct value of c**

+ 0 pts Incorrect or not attempted

1.3 2 / 2

✓ + 2 pts **Correct values of c and n_0**

+ 0 pts Not attempted or Incorrect answer

QUESTION 2

2 2 / 2

✓ + 2 pts **Correct**

+ 0 pts Incorrect

+ 1 pts incomplete answer

+ 2 pts Correct but please rearrange c1 and c2

+ 1 pts Correct c_1 or c_2 (but not both)

+ 0 pts Click here to replace this description.

QUESTION 3

3 2 / 2

✓ + 2 pts **g(n) is correct.**

+ 1.5 pts g(n) is nearly correct, but some small mistakes made.

+ 0 pts Incorrect/Not attempted.

- 2 pts Plagiarism

+ 0.5 pts Correct asymptotic complexity of outer loop.

+ 0.5 pts Correct complexity of inner loop

QUESTION 4

4 1 / 2

+ 2 pts Correct Function mentioned. Good work !!!

✓ + 0 pts **Incorrect Function provided or Empty answer submitted. Please refer to the solution file.**

+ 1 pts In the correct direction, but you need to remove constants while providing theta function.

✓ + 0.5 pts **Correct asymptotic complexity for outer loop.**

✓ + 0.5 pts **Correct asymptotic complexity for inner loop**

+ 1 pts Correctly related complexities of inner and outer loops

Inner Loop - [(n + 1) / 2] times

Outer Loop - log(n - 1) times

Name: Luna McBride
ID: 107607144

CSCI 3104, Algorithms **Profs. Hoenigman & Agrawal**
Problem Set 2a (12 points) **Fall 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.

Instructions for submitting your solution:

- The solutions **should be typed** and we cannot accept hand-written solutions. Here's a short intro to Latex.
- You should submit your work through **Gradescope** only.
- If you don't have an account on it, sign up for one using your CU email. You should have gotten an email to sign up. If your name based CU email doesn't work, try the identikey@colorado.edu version.
- Gradescope will only accept **.pdf** files (except for code files that should be submitted separately on Gradescope if a problem set has them) and **try to fit your work in the box provided**.
- You cannot submit a pdf which has less pages than what we provided you as Gradescope won't allow it.

Name: Luna McBride
ID: 107607144

CSCI 3104, Algorithms **Profs. Hoenigman & Agrawal**
Problem Set 2a (12 points) **Fall 2019, CU-Boulder**

Logarithm-Rules-https://www.chilimath.com/lessons/advanced-algebra/logarithm-rules/

1. *(6 pts) For each of the following pairs of functions $f(n)$ and $g(n)$, we have that $f(n) \in \mathcal{O}(g(n))$. Find valid constants c and n_0 in accordance with the definition of Big-O. For the sake of this assignment, both c and n_0 should be strictly less than 10. You do **not** need to formally prove that $f(n) \in \mathcal{O}(g(n))$ (that is, no induction proof or use of limits is needed).*

   (a) $f(n) = n^3 \log(n)$ and $g(n) = n^4$.

   For Big O: $f(n) \leq cg(n)$ for $n \geq n_0, c > 0, n_0 \geq 0$
   $n^3 log(n) \leq cn^4$
   $\frac{log(n)}{n} \leq c$
   Let's try n_0=4, as with log_2 it becomes 2, a whole number
   $\frac{log(4)}{4} - - > \frac{1}{2}$
   $\frac{1}{2} \leq c$
   c=$\frac{1}{2}$

   (b) $f(n) = n2^n$ and $g(n) = 2^{n \log_2(n)}$.

   For Big O: $f(n) \leq cg(n)$ for $n \geq n_0, c > 0, n_0 \geq 0$
   $n2^n \leq c2^{nlog_2(n)}$
   $\frac{n2^n}{2^{nlog_2(n)}} \leq c$
   $\frac{n2^n}{2^{log_2(n^n)}} \leq c$
   $\frac{n2^n}{n^n} \leq c$
   Let's use n_0=4 in this case, as the n^n function crosses over just before it and 4 is relatively small
   $\frac{4*2^4}{4^4} \leq c - - > \frac{4*16}{256}$
   $\frac{64}{256} \leq c - - > \frac{1}{4}$
   $c = \frac{1}{4}$

(c) $f(n) = 4^n$ and $g(n) = (2n)!$

For Big O: $f(n) \leq cg(n)$ for $n \geq n_0, c > 0, n_0 \geq 0$
$4^n \leq c(2n)!$
$\frac{4^n}{2n!} \leq c$
Let's try n_0=2, as it is a smaller number that is past the overlap period displayed in the graphs
$\frac{4^2}{2(2)!} \leq c$
$\frac{16}{24} \leq c - - > \frac{2}{3}$
c=$\frac{2}{3}$

Name: Luna McBride
ID: 107607144

CSCI 3104, Algorithms **Profs. Hoenigman & Agrawal**
Problem Set 2a (12 points) **Fall 2019, CU-Boulder**

2. *(2 pts) Let* $f(n) = 3n^3 + 6n^2 + 6000$. *So* $f(n) \in \Theta(n^3)$. *Find appropriate constants* c_1, c_2, *and* n_0 *in accordance with the definition of Big-Theta.*

Big Theta: $c_1 g(n) \leq f(n) \leq c_2 g(n)$ for all n $\geq n_0$
$c_1 n^3 \leq 3n^3 + 6n^2 + 6000 \leq c_2 n^3$ for all n $\geq n_0$
$c_1 \leq 3 + \frac{6}{n} + \frac{6000}{n^3} \leq c_2$

Left side: $c_1 \leq 3 + \frac{6}{n} + \frac{6000}{n^3}$
True when $c_1 \leq 3$, as the rest only adds to it.
$c_1 = 3$

Right side: $3 + \frac{6}{n} + \frac{6000}{n^3} \leq c_2$
Let us use a n_0 of 10 as to knock the 6000 down a notch
$3 + \frac{6}{10} + \frac{6000}{1000} \leq c_2$
$9.6 \leq c_2$
$c_2 = 9.6$

$3n^3 \leq 3n^3 + 6n^2 + 6000 \leq 9.6n^3$ for $n \geq 10$

Name: Luna McBride
ID: 107607144

3. *(2 pts) Consider the following algorithm. Find a suitable function $g(n)$, such that the algorithm's worst-case runtime complexity is $\Theta(g(n))$. You do **not** need to formally prove that $f(n) \in \Theta(g(n))$ (that is, no induction proof or use of limits is needed).*

```
count = 0
for(i = n; i >= 0; i = i - 1){
    for(j = i-1; j >= 0; j = j-1){
        count = count+1
    }
}
```

count = 0 –> 1 step
$for(i = n; i >= 0; i = i - 1) - - >$ n+1 steps (yes, even when backwards)
$for(j = i - 1; j >= 0; j = j - 1) - - > \sum_{i=0}^{n}(n - i)$
This sum is 0 to n-1, which goes to n-1. However, for notation, n-1 is essentially n.
count = count+1–> $n * n = n^2$

The two loops run to create an n for each loop, which would make for a good g(n) worse case scenario.
g(n)=n^2

Name: Luna McBride
ID: 107607144

4. *(2 pts) Consider the following algorithm. Find a suitable function $g(n)$, such that the algorithm's worst-case runtime complexity is $\Theta(g(n))$. You do **not** need to formally prove that $f(n) \in \Theta(g(n))$ (that is, no induction proof or use of limits is needed).*

```
count = 0
for(i = 1; i < n; i = i * 3){
    for(j = 0; j < n; j = j + 2){
        count = count + 1
    }
}
```

count = 0 –> 1 step
$for(i = 1; i < n; i = i * 3) - - > log_3(n) + 1$ steps (1,3,9,27... transfer to 1,2,3...)
$for(j = 0; j < n; j = j + 2) - - > \frac{n}{2} + 1$ steps (Cuts out half steps as every odd is ignored)
count = count + 1 –> $\frac{n}{2} log_3(n)$ (Count for every single iteration)

The function overall comes to a worst case of $\frac{n}{2} log_3(n)$, which follows the notation of n log_3(n).
Since we need something of the same complexity to create a big theta, the best g(n) to go with is n log_3 (n) itself.
g(n)=$nlog_3(n)$