A
B
C
D
E
F
G
H
I
J
K
L
M
N
O
P
Q
R
S
T
U
V
W
X
Y
Z
-

COMP2054 2023-24 ADE Coursework ONE (12.5%)
Mon. 26-FEB-2024

Time: 30 minutes.
Do not turn over page until instructed.
Answer ALL questions for a potential total of 25 marks.
Calculators are not permitted.
Write your answers on these sheets within the spaces provided.
Please write clearly.
Write your name & ID in the box below CLEARLY AND IN UPPER CASE LETTERS.
Circle the first initial of your family name on the left.

PARTIAL ANSWERS AND FEEDBACK

Definitions (reminders): "iff" = "if and only if", "s.t." = "such that"

Big-Oh: f(n) is O(g(n)) iff there exist c, n0 s.t.
f(n) <= c g(n) forall n >= n0

Big-Omega: f(n) is Ω(g(n)) iff there exist c>0, n0 s.t.
f(n) >= c g(n) forall n >= n0

Big-Theta: f(n) is Θ(g(n)) iff there exist c', c''>0, n0 s.t.
f(n) <= c' g(n) and f(n) >= c'' g(n) forall n >= n0

little-oh: f(n) is o(g(n)) iff forall c > 0, there exists n0 s.t.
f(n) < c g(n) forall n >= n0

To help distinguish "O" and "o", a note "[[little-oh]]" is added whenever it is used, otherwise it is a Big-Oh.

Question 1. “Primitive Operation counting” [8 marks]

For each of the cases of Java fragments below, give a reasonable estimate of the count of the number of primitive operations they correspond to, and give a **BRIEF 1-line** justification.

a) **int c = 0;**

count= **1 (or small)**

justification: **just need to set the memory to 0 – the declaration itself is an instruction to the compiler, and does not have a direct runtime cost**

b) **h = h/2; // where h is an int**

count= **1-4**

justification: **Get h, get 2, do “/”, write h back. Or might use right shift. Or might just count the /**

c) **k = k * 4; // where k is an int**

count= **1-4**

justification: **Get k, get 4, do “*”, write k back. Or might use double left shift. Or might just count the ***

d) **int[] A = new int[n]; // where n is an int**

count= **100 + n or similar with a larger-than-usual constant and a linear term**

justification: **The constant is ‘large as ‘new’ is expensive as it needs to do a lot of background work to allocate space in the heap. The linear term is because Java initialises all elements to a default value of ‘0’.**
COMMON ERROR: putting a small constant number here

e) **int[] B = A; // using the A from part d**

count= **1 or small**

justification: **This is cheap as it is just copying a pointer (or ‘object reference’). It is NOT copying the entire array! Hence, having a linear term would be wrong.**

Question 2. Big-Oh family with simple f(n) [8 marks]

In the following, you must use $f(n) = n^2 + 2n$

(a.) **From the definitions** (e.g. see front page), prove or disprove the following statements. Show your working. If you claim the statement is true, then be clear about the values of c and n0 that you use. If you claim it is false, then justify your claim, and leave c and n0 blank.

i. f(n) is $O(n^2)$ **TRUE** ~~/ FALSE~~

c =	n0 =
2, or >= 3	**2, or 1**

Need to find c,n0 s.t. n^2 + 2 n <= c n^2 forall n >=n0
c=1 would fail as would then need 2 n <= 0
c=2 needs n^2 + 2 n <= 2 n^2 hence 2 n <= n^2 hence 2 <= n and n0=2 is good
c=3 needs n^2 + 2 n <= 3 n^2 hence 2 n <= 2 n^2 hence 1 <= n and n0=1 is good.
Hence TRUE

ii. f(n) is $\Omega(n^2)$ **TRUE** ~~/ FALSE~~

c =	n0 =
>=1	**1**

Need to find c,n0 s.t. n^2 + 2 n >= c n^2 forall n >=n0
c=1just needs n^2 + 2 n >= n^2 hence 2 n >= 0 and so n0=1
Hence TRUE

(b.) What do you conclude, if anything, about the Big-Theta behaviour of f(n)?

Since it is both big-Oh(n^2) and Big-Omega(n^2), then it is also Big-Theta(n^2)

(c.) Is it true that f(n) is $o(n^2)$ [[little-oh]] ? Circle One: ~~YES /~~ NO

You do not need to prove/justify your answer.

(Explanation – not needed in answers)
Quick reasoning is that little-oh is like "strictly less than", and Big-Theta is like "equal" , and from (b) it "equal", so cannot be "strictly less than".
From the definition would need,
forall c >0 n^2 + 2 n <= c n^2 forall n >=n0 which already fails at c=1
COMMON ERROR: many people selected YES.

Question 3. Big-Oh family with harder f(n) [9 marks]

In the following, you must use $f(n) = \begin{cases} n^2 + 2n & \text{if } n \text{ is even, else} \\ n & \text{if } n \text{ is odd} \end{cases}$

(a.) **From the definitions** (e.g. see front page), prove or disprove the following statements. Show your working. If you claim the statement is true, then be clear about the values of c and n0 that you use. If you claim it is false, then justify your claim, and leave c and n0 blank.

i. f(n) is $O(n^2)$ **TRUE** ~~/ FALSE~~

c =	n0 =
2, or >= 3	**2, or 1**

c=2 works for the even case just as in Q2.a.i.
then the odd case just needs n <= 2 n^2, hence n >= 1/2, and so n0=2 is fine.

c=3 works for the even case just as in Q2.a.i.
then the odd case just needs n <= 3 n^2, hence n >= 1/3, and so n0=1 is fine.

Note: a single value of (c,n0) must be given that works for both the even and odd cases. It is incorrect to give different values for the even and odd cases

OCCASIONAL ERROR: only doing the even case as it is the "best case", or similar.
COMMON ERROR (in many similar questions) trying some value of c, finding it fails, and so concluding it fails overall, instead of trying a different value of c

ii. f(n) is $\Omega(n^2)$ ~~TRUE /~~ **FALSE**

c =	n0 =
"BLANK"	**"BLANK"**

The odd case would need: exists c>0,n0 s.t. n >= c n^2, hence 1/c >= n, and this is clearly not possible forall n >= n0.

Hence FALSE

COMMON ERROR: showing the of case fails at some value of c, but not showing that it fails at all values c > 0.

(b.) What do you conclude, if anything, about the Big-Theta behaviour of f(n)?

It does not have a (useful) Big-Theta.
(Strictly, one could say that f is Theta(f) – but this is vacuous and useless.)