NOTE TO STUDENTS: This file contains sample solutions to the term test together with the marking scheme and comments for each question. Please read the solutions and the marking schemes and comments carefully. Make sure that you understand why the solutions given here are correct, that you understand the mistakes that you made (if any), and that you understand *why* your mistakes were mistakes.

Remember that although you may not agree completely with the marking scheme given here it was followed the same way for all students. We will remark your test only if you clearly demonstrate that the marking scheme was not followed correctly.

For all remarking requests, please submit your request **in writing** directly to your instructor. For all other questions, please don't hesitate to ask your instructor during office hours or by e-mail.

Question 1. [4 MARKS]

Consider the following algorithm:

```
EXP($a, b, p$):        # Computes and returns $a^b \bmod p$.
1.  if $b = 0$:  return 1
2.  if $b \bmod 2 = 1$:      # i.e., b is odd
3.      $x = $ EXP($a, (b-1)/2, p$)
4.      return $(x \times x \times a) \bmod p$
    else:
5.      $x = $ EXP($a, b/2, p$)
6.      return $(x \times x) \bmod p$
```

Give a recurrence relation for the **exact** number of multiplication operations ("×") performed by this algorithm on inputs of size n, in the worst-case. Explain what you are doing, *i.e.*, state what n is equal to (in terms of the algorithm's variables), and what part of the algorithm is represented by each term in your recurrence (use the line numbers to the left of each line for easy reference). **Do *NOT* try to give a closed-form solution for your recurrence!** All we want is the recurrence itself.

CORRECTION: In the original solutions, the constant term in both recurrences below was incorrect (it read "$+3$" instead of "$+2$").

SAMPLE SOLUTION — VERSION 1:

Let $n = b$. Then

- $T(0) = 0$ — line 1 performs no multiplication;
- $T(n) = T(\lfloor n/2 \rfloor) + \mathbf{2}$, for $n > 0$ — in the worst-case, b is odd and the algorithm performs **2** multiplications and one recursive call on input $(b-1)/2 = \lfloor n/2 \rfloor$.

SAMPLE SOLUTION — VERSION 2:

Let n be the number of bits used to represent b. Then

- $T(0) = 0$ — line 1 performs no multiplication;
- $T(n) = T(n-1) + \mathbf{2}$, for $n > 0$ — in the worst-case, b is odd and the algorithm performs **2** multiplications and one recursive call on input $(b-1)/2$ of size $n-1$.

MARKING SCHEME:

- **Recurrence** [3 marks]: correct recurrence: base case [1 mark], recursive term "$T(\ldots)$" for general case [1 mark], and additive term "$+\ldots$" for general case [1 mark]
- **Justification** [1 mark]: reasonable justification

Expected Errors:

- **Asymptotic** [-2 marks]: recurrence is for total running time, using asymptotic notation ($\mathcal{O}/\Theta$)

Marker's Comments:

- **common error**: many students did not read (or understand) the question correctly and gave a recurrence for the general running time, instead of counting only the number of multiplication operations
- **common error**: some students had a recurrence with two recursive terms, so they clearly misunderstood how to write down a recurrence relation from a recursive algorithm — or they misread/ misunderstood the algorithm itself

Question 2. [8 marks]

Consider the following recurrence relation:

$$T(1) = 2,$$
$$T(n) = T(\lceil n/3 \rceil) + g(n) \qquad \text{for } n > 1.$$

Give a **tight** bound on the closed-form solution for $T(n)$, for each function $g(n)$ on the next page. Show your work, *i.e.*, describe clearly how to apply the Master Theorem when it is possible, and when it is not, explain how you obtained your closed-form — **but do *NOT* write a proof**!

Part (a) [2 marks]

$g(n) = 4n$

Sample Solution:

The Master Theorem applies with $a = 1$, $b = 3$, and $d = 1$, so $a < b^d$ and $T(n) \in \Theta(n^d) = \Theta(n)$.

Marking Scheme:

- **Bound** [1 mark]: correct tight bound
- **Justification** [1 mark]: correct application of the Master Theorem (showing values of a, b, d, comparing a with b^d, and stating final result)

Part (b) [2 marks]

$g(n) = 5$

Sample Solution:

The Master Theorem applies with $a = 1$, $b = 3$, and $d = 0$, so $a = b^d$ and $T(n) \in \Theta(n^d \log n) = \Theta(\log n)$.

Marking Scheme:

- **Bound** [1 mark]: correct tight bound
- **Justification** [1 mark]: correct application of the Master Theorem (showing values of a, b, d, comparing a with b^d, and stating final result)

Part (c) [4 marks]

$g(n) = 6^n$ (Hint: $5 + 5 \cdot 6 + 5 \cdot 6^2 + \cdots + 5 \cdot 6^n = 6^{n+1} - 1$.)

Warning: This may be a bit long. Don't spend too much time on it at first — plan your time carefully!

Sample Solution:

The Master Theorem does not apply, so we perform repeated substitution on a simplified version of the recurrence (ignoring floors and ceilings).

$$\begin{aligned} T(n) &\approx T(n/3) + 6^n \\ &\approx T(n/9) + 6^{n/3} + 6^n \\ &\approx T(n/27) + 6^{n/9} + 6^{n/3} + 6^n \\ &\approx \dots \\ &\approx T(1) + 6^3 + \cdots + 6^{n/3} + 6^n \\ &< 6^{n+1} \qquad \text{(from the hint)} \end{aligned}$$

Since $T(n) \geqslant 6^n$ (directly from the recurrence), we get that $T(n) \in \Theta(6^n)$.

Marking Scheme:

- **Bound** [1 mark]: correct bound
- **Closed Form** [2 marks]: correct approach and correct application of repeated substitution
- **Justification** [1 mark]: correct explanation of how to reach a tight bound from the closed form

Marker's Comments:

- **common error** [−1]: no closed form reached
- **common error** [−1]: missing repeated substitution or other explanation for the closed form

Question 3. [6 marks]

Suppose that we want to find, at the same time, both the minimum *and* the maximum elements in an unsorted list. This can be done with two loops: one to find the minimum (using exactly $n-1$ comparisons between list elements), and the other to find the maximum (using $n-2$ comparisons between list elements, because we can skip the comparison with the minimum element). This "naive algorithm" makes a total of $2n-3$ comparisons between list elements, and we would like to do better.

Write a divide-and-conquer algorithm MinMax that returns, as a pair, the minimum *and* maximum elements in a list. *You do **NOT** have to show that your algorithm makes fewer than $2n-3$ comparisons between list elements — for this question, just write a correct algorithm.*

Hint: This is simple: don't overthink it! Include two base cases: one for $n = 1$ and one for $n = 2$.

Sample Solution:

```
MinMax($A$):
    if len($A$) == 1: return $(A[0], A[0])$
    else if len($A$) == 2:
        if $A[0] \leqslant A[1]$:   return $(A[0], A[1])$
        else:                      return $(A[1], A[0])$
    else:
        $(min_0, max_0)$ = MinMax($A[0 \dots \frac{\text{len}(A)}{2} - 1]$)
        $(min_1, max_1)$ = MinMax($A[\frac{\text{len}(A)}{2} \dots \text{len}(A) - 1]$)
        return $\big(\min(min_0, min_1), \max(max_0, max_1)\big)$
```

MARKING SCHEME:

- **Divide-and-Conquer** [1 mark]: algorithm clearly follows the divide-and-conquer paradigm (even if it is incorrect)
- **Base Cases** [2 marks]: correct base cases
- **Recursive Case** [3 marks]: correct recursive case

MARKER'S COMMENTS:

- **common error** [-4]: answer does not follow divide-and-conquer paradigm (*i.e.*, no recursion)
- **common error** [-1]: forgetting to "conquer" (*i.e.*, recursive calls are made, but nothing is done with the values they return)
- **common error** [-1]: infinite recursive loop

Question 4. [8 MARKS]

State precise preconditions and postconditions for your algorithm MINMAX from Question 3, then write a detailed proof that your algorithm is correct.
NOTE: You may complete this part and get most of the marks, even if your algorithm is not actually correct, as long as it is clear that you understand how to prove the correctness of your algorithm.
SAMPLE SOLUTION:

Precondition: A is a non-empty array of numbers (or other element types that can be ordered).

Postcondition: The call MINMAX(A) terminates and returns a pair (x, y) where x is the minimum element in A and y is the maximum element in A.

Proof: By complete induction on $n = \text{len}(A)$, we prove that for every input of size n, if the precondition is satisfied and the algorithm is executed, then the postcondition will be satisfied.

Base Case 1: Assume A is an arbitrary input of size $n = \text{len}(A) = 1$. Then MINMAX(A) returns $(A[0], A[0])$ on the first line, which satisfies the postcondition.

Base Case 2: Assume A is an arbitrary input of size $n = \text{len}(A) = 2$. Then MINMAX(A) compares $A[0]$ with $A[1]$ on the third line, and returns either $(A[0], A[1])$ or $(A[1], A[0])$, whichever satisfies the postcondition.

Ind. Hyp.: Assume A is an arbitrary input of size $n = \text{len}(A) \geqslant 3$ and that the algorithm is correct for every input whose size is strictly smaller than n.

Ind. Step: Since $n \geqslant 3$, $1 \leqslant \lfloor n/2 \rfloor \leqslant \lceil n/2 \rceil < n$, so the input for each recursive call is a non-empty array that satisfies the precondition and whose size is strictly less than the size of A. By the I.H., this means that after both recursive calls, min_0 is the smallest element in the first half of A, max_0 is the largest element in the first half of A, min_1 is the smallest element in the second half of A, max_1 is the largest element in the second half of A. But then, $\min(min_0, min_1)$ is the smallest element in A, $\max(max_0, max_1)$ is the largest element in A, and the algorithm returns exactly the right value to satisfy the postcondition.

Marking Scheme:

- **<u>P</u>re/Post-Conditions** [2 marks]: reasonable preconditions and postconditions
- **<u>S</u>tructure of Proof** [3 marks]: clear attempt to prove that the algorithm satisfies the postcondition for every input that meets the precondition, by induction on the size of the input (including correct inductive structure)
- **<u>C</u>ontent of Proof** [3 marks]: reasonable proof, in terms of the student's algorithm (this will be strongly dependent on the way that each student wrote their algorithm; please mark accordingly, *i.e.*, give some marks for reasonable attempts to prove correctness even if the algorithm is, in fact, incorrect)

Marker's Comments:

- **common error** [-4]: proving something about the running time instead of correctness [*Instructor's Comment*: A deduction of only -4 is actually quite generous here. . .]
- **common error** [-1]: postcondition does not specify that the return value contains both the minimum and the maximum value
- **common error** [-3 to -4]: postcondition does not mention the purpose of the algorithm
- **common error** [-3 to -5]: writing an argument that does not use induction (in which case it is not a proof)
- **common error** [-1]: using simple induction instead of complete induction
- **common error** [-1]: incomplete or missing induction hypothesis

Bonus. [3 marks]

Prove that your algorithm MinMax from Question 3 makes *fewer* comparisons between list elements than the naive algorithm described in Question 3. Explain what you are doing and show your work.

Sample Solution: (Not provided for bonus. . .)

Marking Scheme: *Be particularly picky when marking the bonus!*

- **<u>R</u>ecurrence** [1 mark]: correct recurrence relation for the student's algorithm
- **<u>S</u>olution** [2 marks]: correct solution for the recurrence relation, including a suitable proof that the solution is correct