This part for lecturer's grading use only

NATIONAL UNIVERSITY OF SINGAPORE

SCHOOL OF COMPUTING

AY2018/2019, SEMESTER 2

FINAL ASSESSMENT FOR

CS4231: Parallel and Distributed Algorithms

30 April 2019 **Time Allowed: 2 Hours**

INSTRUCTIONS TO CANDIDATES

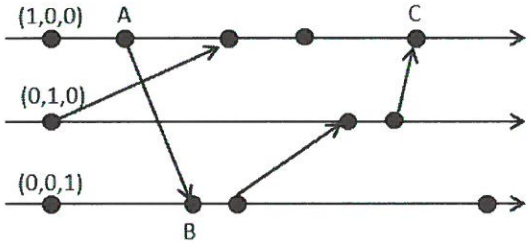
1. This assessment paper contains **12** problems and **12** printed pages, including this page.
2. Students are required to answer **ALL** questions.
3. All questions should be answered within the space provided in this booklet.
4. This is an **Open Book** assessment.
5. Read each question carefully before you answer. If you misunderstand the question and answer a different question, you get **zero credit** for that question.
6. It is your job to show the correctness of each statement you make. Otherwise points will be deducted **even if the lecturer cannot prove you are wrong**. (This is a general rule for all theory subjects – otherwise you can claim P=NP and nobody can prove you are wrong.)
7. Please write your Student Number only. Do not write your name.

STUDENT NO:

DO NOT WRITE BELOW THIS LINE

Problem 1,2	
Problem 3,4	
Problem 5,6	
Problem 7	
Problem 8	
Problem 9	
Problem 10	
Problem 11	
Problem 12	

Problem 1 (6 marks)



Consider the above execution of a distributed system, where the vector clock value of the first event on each process has already been indicated.

What should be the vector clock value for event A? (Just write the answer, no explanation needed.) _____

What should be the vector clock value for event B? (Just write the answer, no explanation needed.) _____

What should be the vector clock value for event C? (Just write the answer, no explanation needed.) _____

Problem 2 (4 marks)

In the context of distributed systems, for any two messages m1 and m2, let s1 and r1 be the send and receive event respectively for m1, and let s2 and r2 be the send and receive event respectively for m2. Consider the following two properties:

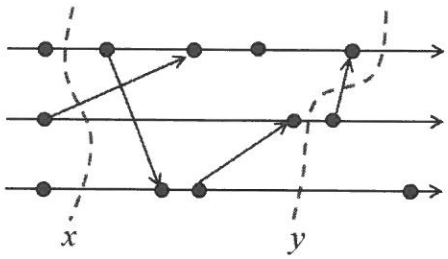
> Property#1: If i) s1 happened-before s2, ii) s1 and s2 are on the same process, and iii) r1 and r2 are on the same process, then r2 must not happened-before r1.
>
> Property#2: If i) s1 happened-before s2 and ii) r1 and r2 are on the same process, then r2 must not happened-before r1.

If a distributed system satisfies Property#1 for all m1 and m2, do you feel that it will always satisfy Property#2 for all m1 and m2?

Indicate yes/no first: _____

Give a rigorous proof or counter-example:

Problem 3 (4 marks)

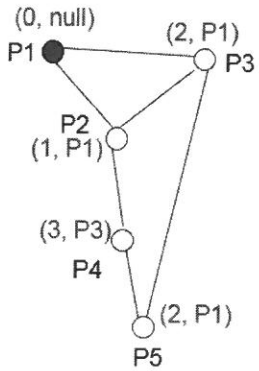


Consider the above execution of a distributed system.

Does the cut *x* correspond to a consistent global snapshot? (Just write yes/no, no explanation needed.) _____

Does the cut *y* correspond to a consistent global snapshot? (Just write yes/no, no explanation needed.) _____

Problem 4 (4 marks)



Consider the self-stabilizing spanning tree algorithm in distributed systems, with P1 being the root. Imagine that the algorithm continues its execution from the state depicted in the above figure. For each node, the figure indicates the current value of the (**dist**, **parent**) tuple on that node. Define an *action* to be one node executing its code (in the algorithm) once. Give a sequence of actions (or more precisely, a sequence of nodes who take actions one by one in that ordering), so that the **dist** value on P5 first increases and then decreases.

Give your sequence of actions (i.e., the sequence of nods taking actions --- for example "P1 P2 P4 P1") first:

_____

For each action in your sequence, indicate the new value of the (**dist**, **parent**) tuple on the node taking the action, **after** that node takes that action, in the following. If your sequence contains more/less than 5 actions, feel free to add/delete entries. For example, for the first action in the above sequence, it should be "P1:(0, null)".

Action #1: _____

Action #2: _____

Action #3: _____

Action #4: _____

Action #5: _____

Problem 5 (4 marks)
In the context of parallel systems, construct a history that satisfies sequential consistency but not linearizability. In your execution, there should only be a single shared object, and the type of the object should be a stack. The initial value of the stack should be empty. Your history should have at most 3 processes.

Draw your history:

Explain why your history satisfies sequential consistency:

Explain why your history does not satisfy linearizability:

Problem 6 (4 marks)
Alice proposes the following design where she uses Java-style monitor to implement a standard critical section (CS) in parallel systems. She wants to ensure the Mutual-exclusion property and the Progress property. To enter the CS, a process should invoke RequestCS(). To exit the CS, a process should invoke ReleaseCS(). Her design also requires that the function Initialize() be executed once, for the purpose of global initialization at the very beginning, before any process invokes RequestCS().

```
shared object dummy = null;
void Initialize() { dummy = new object(); synchronized (dummy) { dummy.notify(); }}

void RequestCS() {  synchronized(dummy) { dummy.wait(); } }
void RequestCS() {  synchronized(dummy) { dummy.notify(); } }
```

Does her design satisfy Mutual-exclusion and Progress? Why?

Problem 7 (6 marks)
Consider any given execution of some distributed system, and let n be the total number of events in this execution. Alice claims that for any k where $0 \leq k \leq n$, there always exists some consistent global snapshot containing exactly k events.

Do you agree with Alice? Answer Yes/No: _____

If you answered "Yes", give a rigorous proof. (Your proof should hold for all n values.) If you answered "No", give a counter-example. (In your counter-example, you should first indicate the values of n and k, and then fully describe an execution with n events, such that there is no consistent global snapshot with exactly k events.)

Problem 8 (4 marks)
Consider the distributed consensus problem with 10 nodes, under the synchronous timing model. The communication channels are all reliable. During the execution of the algorithm, there are up to 1 crash failure and up to 3 byzantine failures, among these 10 nodes. (This means that in the worst case, there will be only 6 nodes that do not experience any failures.) The problem comes with the following requirements:

- Termination: All nodes that do not experience any failures eventually decide.
- Agreement: The decisions made by nodes that do not experience byzantine failures are the same.
- Validity: If all nodes that do not experience any failures have the same input value, then that value should be the (unique) decision value. Otherwise nodes are allowed to decide on anything (except that they still need to satisfy Agreement).

Do you feel this problem can be solved using a deterministic algorithm? Answer Yes/No: _____

If you answered "Yes", give an algorithm. (You should first present the high-level idea behind your algorithm, then provide complete pseudo-code, and finally prove Termination, Agreement, and Validity.) If you answered "No", give a rigorous impossibility proof.

Problem 9 (6 marks)
Consider the lead election problem on a ring with $3n$ nodes, where $n > 10$. Among these $3n$ nodes, n of them have the same id X, n of them have the same id Y, and the remaining n of them have the same id Z. It is known that X≠Y, X≠Z, and Y≠Z. We aim to elect a unique leader, regardless of how the nodes are ordered on the ring. **The value of n is known to your algorithm.**

Do you feel this problem can be solved using a deterministic algorithm? Answer Yes/No: _____

If you answered "Yes", give an algorithm. (You should first present the high-level idea behind your algorithm, then provide complete pseudo-code, and finally prove its correctness.) If you answered "No", give a rigorous impossibility proof.

Problem 10 (4 marks)
Consider the distributed consensus problem with 2 nodes x and y, under the synchronous timing model. As we know, the synchronous timing model enables us to use the notion of rounds. The nodes are reliable, but the communication channel may drop messages arbitrarily, subject to the following constraint: In each round, the communication channel either "does not drop any message" or "drops both the message from x to y and the message from y to x". The problem comes with the following requirements:

- Termination: All nodes eventually decide.
- Agreement: All nodes decide on the same value.
- Validity: If all nodes start with 0, decision should be 0. If all nodes start with 1 and no message is lost throughout the execution, decision should be 1. Otherwise nodes are allowed to decide on anything (except that they still need to satisfy Agreement).

Do you feel this problem can be solved using a deterministic algorithm? Answer Yes/No: _____

If you answered "Yes", give an algorithm. (You should first present the high-level idea behind your algorithm, then provide complete pseudo-code, and finally prove Termination, Agreement, and Validity.) If you answered "No", give a rigorous impossibility proof.

Problem 11 (8 marks)

Consider the distributed consensus problem under the asynchronous timing model, where nodes may experience crash failures and where channels are reliable. There are total $n = k \times g$ nodes, for some integers $k > 10$ and $g > 10$. Both k and g are known to your algorithm. The nodes are categorized into g groups, with each group having exactly k nodes. All nodes in the same group always have the same input value for the distributed consensus problem. **Your algorithm, however, does not know which nodes belong to which group.** The total number of crash failures in any given execution will be at most f. The nodes that fail may belong to the same group, or may belong to different groups. Certain groups may contain more failed nodes that other groups. Under the above setting, you are asked to design a deterministic algorithm to solve the distributed consensus problem with the following standard requirements:

- Termination: All nodes (that have not failed) eventually decide.
- Agreement: All nodes that decide (including those who later failed) should decide on the same value.
- Validity: If all nodes have the same initial input value, that value should be the only possible decision value. Otherwise the nodes can decide on anything (except that they still need to satisfy the Agreement requirement).

Your algorithm should tolerate as large an f value as possible.

What is the maximum f that your algorithm can tolerate, as a function of k and g? If you feel that the problem is not solvable even for a single crash failure, please write 0 as your answer. Your answer: _____

Rigorously prove that it is not possible to tolerate any f value that is larger than your answer above:

(This problem continues on the next page.)

(Problem 11 continues.)
If your previous answer on the maximum f value was larger than zero, describe and then prove your algorithm below. (You should first present the high-level idea behind your algorithm, then provide complete pseudo-code, and finally prove correctness.) If your previous answer on the maximum f value was zero, you do not need to write anything below.

Problem 12 (6 marks)
Consider a distributed system with n nodes, where $n > 7$. The topology among the n nodes is some arbitrary undirected graph G (which may or may not be connected). Putting it another way, the vertices of G are the n nodes, and a node x can send messages to anther node y if and only if there is an edge between x and y in G. Every communication channel (between two nodes) guarantees FIFO message delivery (namely, if x sends two messages to y, then the message sent first will always be received first). A node never sends messages to itself. Whenever a node receives a message, it delivers the message (to the application) immediately without any delay.

Alice wants to define some function $f()$ that takes G as input and returns a boolean value. Alice wants to ensure all of the following:

1. If $f(G)$=true, then all possible executions of this distributed system will always satisfy causal order message delivery.

2. $f(G)$ should return true for as many G as possible. (Namely, $f()$ should not be unnecessarily pessimistic.)

3. The function $f(G)$ can be computed in polynomial time complexity with respective to n. (Namely, there should exist some centralized algorithm that takes G as an input, and outputs $f(G)$ within polynomial time.)

You are asked to define the function $f()$, while satisfying all three requirements above. (Hint: What if we define $f(G)$=true if and only if G is not a clique? What if we define $f(G)$=true if and only if G contains no edges at all?)

Write your answer on the next page, do not write on this page.

(For Problem 12.)
Rigorously define your function *f()*:

Rigorously prove that your *f()* satisfies all three requirements. (**You should do <u>three separate proofs</u>, one for each requirement.**)

END-OF-PAPER