*Answer question 1 and any other two questions.*

1.

a) Write and explain the operation of a storage allocation monitor which allocates contiguous space from an array of size [1:N], and which has the following basic structure:

```
  _monitor(ResourceAllocator)
    op request(size: int) returns location: int
    op free(location: int)
  _body(ResourceAllocator)
    const N = 10000
    var array[1:N]

    ...

    _proc(request(size) returns location)
      ## Should allocate 'size' contiguous elements of the array,
      ## and return the index of the first element allocated.
      ## If there is currently no free space large enough, then
      ## wait for there to be one.
    _proc_end

    _proc(free(location))
      ## Should free up the elements of the array previously
      ## allocated by a request.
    _proc_end
_monitor_end
```

You may use any of the semantics for condition variables, rather than being restricted to those provided by m2sr, provided you say which you are using. [17]

b) Assume that there are two clients and an odd number of servers. Show how *livelock* may occur in the two-phase commit protocol and suggest a solution to this problem. [8]

c)

i) How could a non-deterministic algorithm be implemented efficiently given that you have available as many processors as you may need?

ii) Assuming that you have only a single processor available, how would you implement a non-deterministic algorithm? [8]

[TURN OVER]

2. *Consider a system with three chef processes and one agent process. Each chef continuously cooks a meal then eats it. But to cook and eat a meal, the chef needs three ingredients: rice, chicken and black bean sauce. One of the chef processes has rice, one has chicken and the last has black bean sauce. The agent has an infinite supply of all three materials. The agent places two ingredients on the table, and the chef with the remaining ingredient cooks a meal and eats it, signalling the agent on completion. The agent then puts out another two of the three ingredients and the cycle repeats.*

   Explain your general approach to the synchronisation of the agent and the chefs, using semaphores. Next, write and comment an SR program which implements this approach. What are the safety and liveness properties of your solution? [33]

3. Assume that N processes are arranged in a unidirectional ring network, where each process has a process identifier.

   a) Present the pseudo-code of a deterministic algorithm to *elect* a single leader out of the N processes, assuming the process identifiers are unique and are linearly ordered. [11]

   b) Present the pseudo-code of a randomized algorithm to *elect* a single leader out of the N, assuming the process identifiers are *not* linearly ordered. [11]

   c) Discuss the use of a randomized algorithm to solve a concurrency problem as opposed to using a deterministic algorithm, with reference to the leader election problem. [11]

4.
   a) Write a maximum of 50 words on each of the following:

      i) The common memory arbiter

      ii) Critical sections

      iii) Safety properties

      iv) Liveness properties

      v) Deadlock

      vi) Starvation

      vii) Livelock [14]

[Question 4 cont. over page]
[CONTINUED]

b) The *bakery algorithm* is so named because of its similarity to the way that bakeries or delicatessens work. On entering the store, each customer receives a number, and the customer with the lowest outstanding number is served next.

In the following algorithm it is not possible to ensure that two processes (customers) do not obtain the same number. Thus, in the case of a tie, the lowest numbered process is served first. The following is the two process bakery algorithm:

```
resource Bakery
  var turn[1:2] := ([2] 0)

  process customer1
    do true ->
      turn[1] := 1                    ##### Sic
      turn[1] := turn[2] + 1

      do turn[2] != 0 and turn[1] > turn[2] ->
        skip
      od

      CRITICAL SECTION

      turn[1] := 0

      NON-CRITICAL SECTION
    od
  end customer1

  process customer2
    do true ->
      turn[2] := 1                    ##### Sic
      turn[2] := turn[1] + 1

      do turn[1] != 0 and turn[2] >= turn[1] ->
        skip
      od

      CRITICAL SECTION

      turn[2] := 0

      NON-CRITICAL SECTION
    od
  end customer2
end Bakery
```

[Question 4(b) cont. over page]
[TURN OVER]

[Question 4(b) cont.]

i) Explain the working of this algorithm, paying particular attention to the odd looking double assignments at lines 6-7 and 23-24. Would the code work with only the second of the pair? [10]

ii) What liveness and safety properties can you show for the algorithm? [9]

5.

a) Consider the following pseudo-code for solving the dining philosophers problems, where `Nphil` represents the number of philosophers:

```
sem fork_free[1:Nphil] := ([Nphil] 1)
process philosopher (i := 1 to Nphil)
  do true ->
    THINK
    P(fork_free[i])
    P(fork_free[i+1])
    EAT
    V(fork_free[i])
    V(fork_free[i+1])
  od
end philosopher
```

What is wrong with the above algorithm and how can it be corrected ? [11]

b) Define Nick's parallel complexity class and discuss its significance. [11]

c)

i) Devise the pseudo-code of a Monte Carlo algorithm to evaluate pi.

ii) How can parallelism be introduced into your algorithm to speed it up ?

(You can assume that a function `random()` is available which returns a uniformly distributed random real number between zero and one, and that the usual arithmetic operators are built-in.) [11]

[END OF PAPER]