Name_____

Student Number _____

SFWR TECH 4CC3E

EVENING CLASS — Dr. J. Fortuna
DURATION OF EXAMINATION: 3 Hours
McMaster University Final Examination — April 2015

This examination paper includes 9 pages and 6 questions. You are responsible for ensuring that your copy of the paper is complete. Bring any discrepancy to the attention of your invigilator.

SPECIAL INSTRUCTIONS:

Any type of calculator is permitted. When the question is subdivided into a number of parts, the marks allocated for each section are indicated. This is a **closed book** examination. No reference materials of any kind are permitted

The distribution of marks is as follows (do not write in this box!):

Question	Marks Available	Mark Obtained
1	**5**	
2	**20**	
3	**15**	
4	**10**	
5	**10**	
6	**10**	
Total	**70**	

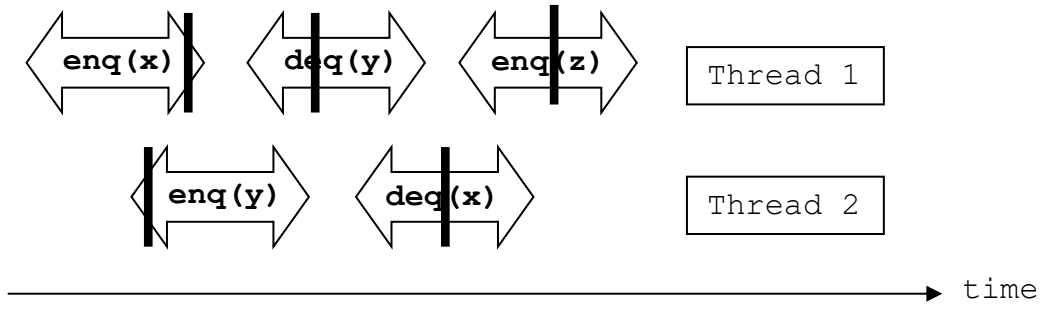
Name_____

Student Number _____

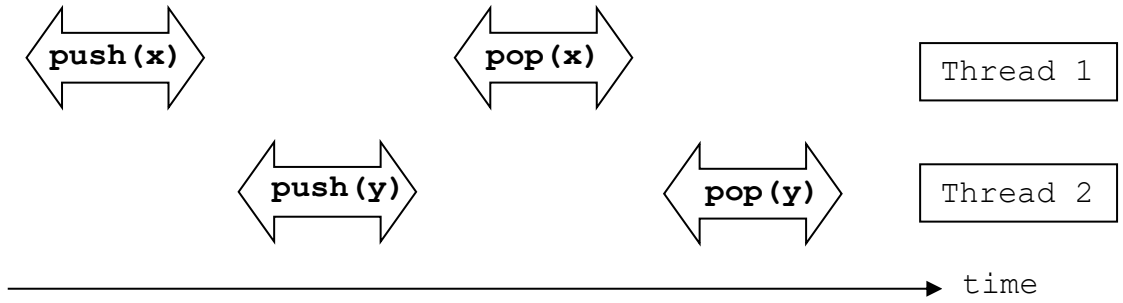
Question 1. Short Answer

(5 marks) State whether or not the following execution histories are linearizable and, if they are, show linearization points.

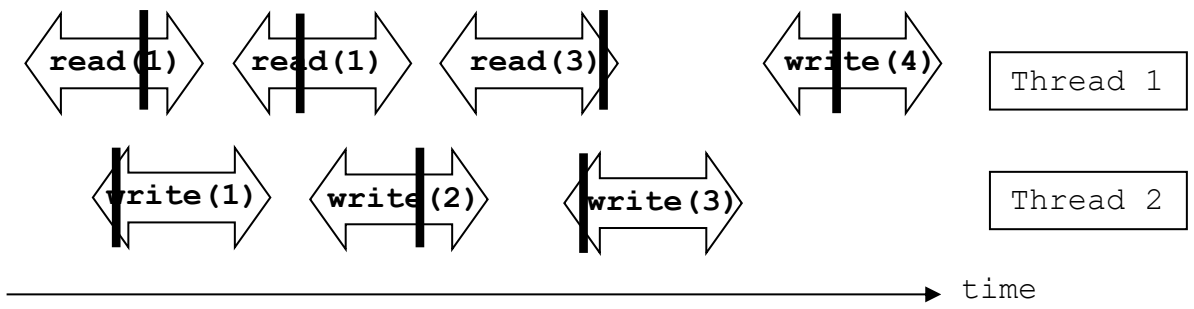
History 1: linearizable



History 2: not linearizable



History 3: linearizable



Name_____

Student Number _____

Question 2. Speedup

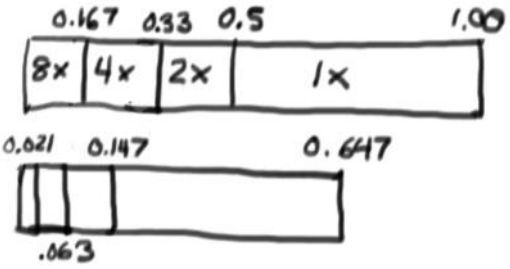
Assume that we have a program where **50 percent** of the program can be parallelized. Now assume that **if 8 cores are available**, 1/3 of the concurrent part of the program can receive a speedup of **8x**, 1/3 of the concurrent part of the program can receive a speedup of **4x** and the remaining 1/3 of the concurrent part of the program can receive a speedup of **2x**. Also assume that **if only 4 cores are available**, 2/3 of the concurrent part of the program can receive a speedup of **4x** and the remaining 1/3 of the concurrent part of the program can receive a speedup of **2x**.

(20 marks) Using Amdahl's law, determine whether the **8 core implementation** is more than **1.25x faster** than the **4 core implementation.**

Amdahl's Law

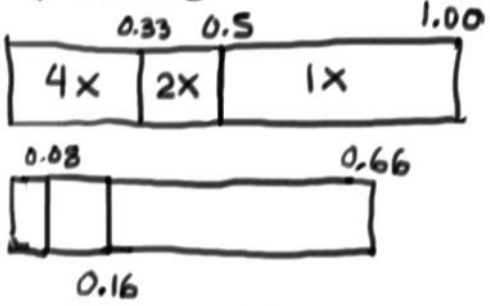
$$S(n) = \frac{t_s}{ft_s + (1-f)t_s/n} = \frac{n}{1 + (n-1)f}$$

a) 8 CORES



SPEEDUP = 1.54

b) 4 CORES



SPEEDUP = 1.51

THERE IS HARDLY ANY INCREASE FOR THE 8 CORE IMPLEMENTATION

Name_____

Student Number _____

Question 2 Continued (extra space provided for your solution if needed)

Name_____

Student Number _____

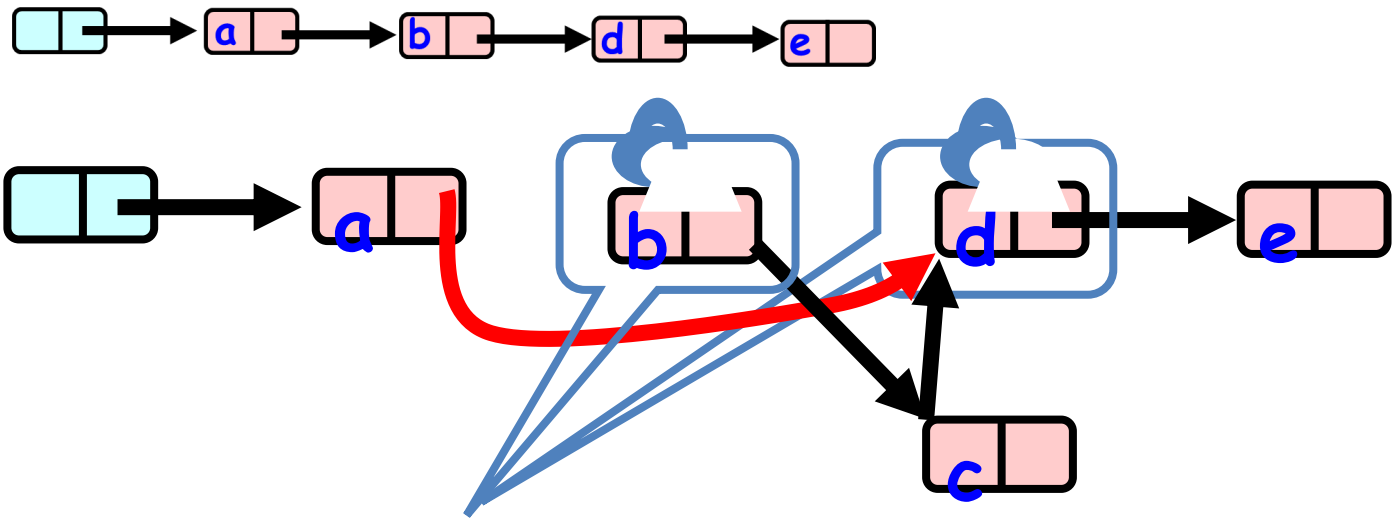
Question 3. – Concurrent Linked List

Consider the following Java implementation the **add** and remove operations of a **concurrent linked list**:

```
1   public boolean add(T item) {
2     int key = item.hashCode();
3     while (true) {
4       Node pred = head;
5       Node curr = pred.next;
6       while (curr.key <= key) {
7         pred = curr; curr = curr.next;
8       }
9       pred.lock(); curr.lock();
10      try {
11        if (validate(pred, curr)) {
12          if (curr.key == key) {
13            return false;
14          } else {
15            Node node = new Node(item);
16            node.next = curr;
17            pred.next = node;
18            return true;
19          }
20        }
21      } finally {
22        pred.unlock(); curr.unlock();
23      }
24    }
25  }
```

```
26  public boolean remove(T item) {
27    int key = item.hashCode();
28    while (true) {
29      Node pred = head;
30      Node curr = pred.next;
31      while (curr.key < key) {
32        pred = curr; curr = curr.next;
33      }
34      pred.lock(); curr.lock();
35      try {
36        if (validate(pred, curr)) {
37          if (curr.key == key) {
38            pred.next = curr.next;
39            return true;
40          } else {
41            return false;
42          }
43        }
44      } finally {
45        pred.unlock(); curr.unlock();
46      }
47    }
48  }
```

(a) (5 marks) Describe clearly, with the aid of diagrams, what can go **wrong** if, for the list shown below, with **respect to the code above**, **Thread Q** is trying to add **node c** and **Thread R** removes **node b** before **Thread Q** locks the nodes.

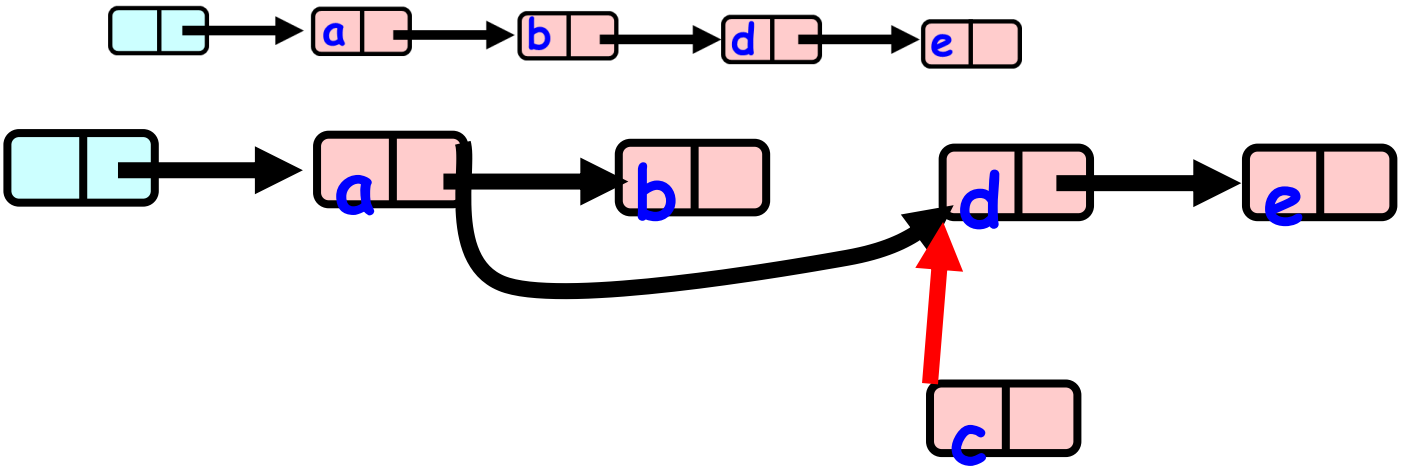


B was removed and c was not added.

Name_____

Student Number _____

(b) (5 marks) Describe clearly, with the aid of diagrams, what can go **wrong** if, for the list shown below, with **respect to the code on the previous page**, Thread Q is trying to remove **node b** and **Thread R** adds **node c before Thread Q** locks the nodes.



Instead b and c were deleted!

(c) (5 marks) What is the validate function used for and what must be accomplished in the **validate** function? Your answer must describe how **pred** and **curr** are used to perform the validation.

Validate must check that pred is accessible from the head. This ensures that we detect the case that a node was removed before we locked the nodes for adding.

Validate must also check that pred is connect to curr. This ensures that we detect the case that a node was added before we locked the nodes for removing.

Name_____

Student Number _____

Question 4. Lock Free Stack

(a) (5 marks) Explain in **words** how to implement a **lock free stack** using an **elimination-backoff array**. You don't need to supply all the details, but you must explain how the elimination array is used and how values end up being pushed to and popped from the stack.

- Access Lock-free stack FIRST!,
  - If **uncontended**, apply operation
  - if **contended**, back off to elimination array and attempt elimination
  - Repeat until eventually one of these succeeds

- An exchanger is an object that allows exactly two threads to rendezvous and exchange values.
  - The first process that arrives writes its value and spins until a second process arrives.
  - The second process detects that the first is waiting, reads its value, and "signals" the exchange.
  - Both processes have now read the other's value, so they can return.
  - The call of the first process may timeout, allowing it to leave the exchanger.
- The Elimination Array is an array of exchangers.
- Processes should spin rather than block in the exchanger, since we expect them to wait only for very short durations

(b) (3 marks) Explain the **two ways** to control the likelihood of a collision in the **elimination-backoff array** and describe how each of them impact the **behavior** of the concurrent stack.

The array can be made resized. This has the effect of changing the probability of an elimination. A larger array creates a smaller probability of an elimination. A larger number of threads needs a larger array if the probability of elimination is to remain the same.

The time window for a collision can be changed. This has the effect of changing the time granularity over which accesses to the stack are considered "concurrent".

(c) (2 marks) Give 1 reason why a stack is inherently **sequential** and 1 reason why it is not.

A stack is inherently sequential if one considers that exactly simultaneous accesses to the top of the stack must result in indeterminate results.

A stack is not sequential if one can relax the notion of simultaneity slightly to allow for cancellations of "almost concurrent" pushes and pops. In the case, the stack is only modified for pushes without a concurrent pop or a pop without a concurrent push.

Name_____

Student Number _____

Question 5. Concurrent Queue

Consider the following Java code for the **enqueue** operation in a concurrent queue:

```
 9    public void enq(T value) {
10      Node node = new Node(value);
11      while (true) {
12        Node last = tail.get();
13        Node next = last.next.get();
14        if (last == tail.get()) {
15          if (next == null) {
16            if (last.next.compareAndSet(next, node)) {
17              tail.compareAndSet(last, node);
18              return;
19            }
20          } else {
21            tail.compareAndSet(last, next);
22          }
23        }
24      }
```

(a) (2 marks) Explain the purpose of the **if** statement on **line 14**

If another thread modified the tail before this point, abandon the enqueue.

(b) (2 marks) Explain the purpose of the **if** statement on **line 15**

If another thread modified last.next before this point, the other thread was attempting a logical enqueue. Therefore we will finish the enqueue on line 21

(c) (2 marks) Explain the purpose of the **compareAndSet** instruction on **line 16**

This instruction attempts the logical enqueue. If it fails, abandon the enqueue.

(d) (2 marks) Explain the purpose of the **compareAndSet** instruction on line **17**

This instruction performs the physical enqueue. It will succeed as long as the tail is the last node. If it fails, another thread modified the tail instead (on line 21)

(e) (2 marks) Explain the purpose of the **compareAndSet** instruction on **line 21**

This instruction performs the physical enqueue for threads that were partway through the sequence when they were interrupted by another thread.

Name_____

Student Number _____

Question 6. Parallel Sort

(a) (5 marks) Provide **pseudocode** for any parallel sorting algorithm that you would like to.

(b) (5 marks) Briefly describe how your algorithm works

********THE END********