Full Name: _____

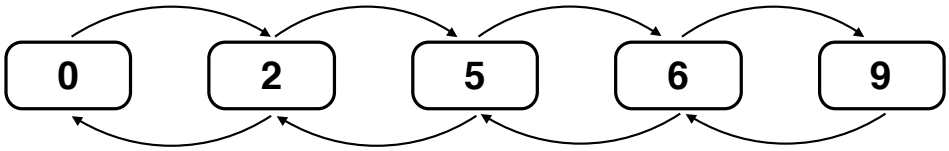
Andrew Id: _____

CMU 15-418/618 Practice Exercise 5

Problem 1: Concurrent Linked Lists

Consider a **SORTED doubly-linked list** that supports the following operations.

- `insert_head`, which traverses the list from the head. The implementation uses hand-over-hand locking just like in class.
- `delete_head`, which deletes a node by traversing from the head, using hand-over-hand locking just like in class.
- `insert_tail`, which traverses the list **backwards from the tail** to insert a node using hand-over-hand locking in the opposite order as `insert_head`.



A. (2 pts) Your friend writes three unit tests that each execute a pair of operations concurrently on the list shown above.

- Test 1: `insert_head(1)`, `delete_head(9)`
- Test 2: `insert_head(8)`, `delete_head(2)`
- Test 3: `insert_head(8)`, `insert_tail(1)`

The first two unit tests complete without error, but the third test goes badly and it does not terminate with the right answer. Describe what behavior is observed and why the problem occurs. (All unit tests start with the list in the state shown above.)

B. (4 pts) Imagine that locks in this system supported not only `lock()` and `unlock()`, but the ability to query the state of the lock via the call `trylock()` (this call takes the lock if the lock is free, but immediately returns false if the lock is currently locked – it does not block). Given this functionality, describe a fix to the problem you identified in part A? **Warning, your answer should avoid livelock, but it is acceptable in this problem to allow for the possibility of starvation.**

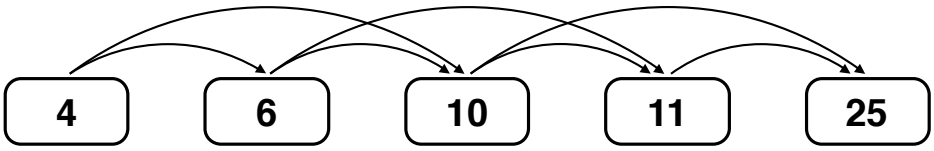
C. (1 pt) Imagine you removed all locks and implemented `insert_head`, `delete_head`, and `insert_tail` by placing the entire body of these functions in an atomic block for execution on a system **supporting optimistic transactional memory**. Does this fix the correctness problem described in part A? Why or why not?

D. (1 pt) Consider two transactions simultaneously performing `insert_head(3)` and `delete_head(9)`. Assume both transactions start at the same time on different cores and the transaction for `insert_head(3)` proceeds to commit while the `delete_head(9)` transaction has just iterated to the node with value 6. Must either of the two transactions abort in this situation? Why? **(Remember this is an optimistic transactional memory system!)**

E. (1 pt) Must either transaction abort if the transaction for `delete_head(9)` proceeds to commit before the transaction for `insert_head(3)` does? Why? **Please assume that at the time of the attempted commit, `insert_head(3)` has iterated to node 2, but has not begun to modify the list.**

F. (1 pt) Must either transaction abort if the situation in part E is changed so that `delete_head(9)` attempts to commit first, but by this time `insert_head(3)` has made updates to the list (although not yet initiated its commit)? Why?

Another Fine-Grained Locking Question



Consider the semi-skip list structure pictured above. Each node maintains a pointer to the next node and the next-next node in the list. **The list must be kept in sorted order.** A node struct is given below.

```
struct Node {
   int value;
   Node* next;
   Node* skip;   // note that skip == next->next
};
```

(10 pts) Please describe a thread-safe implementation of **node deletion** from this data structure. You may assume that deletion is the only operation the data structure supports. Please write C-like pseudocode.

To keep things simple, you can ignore edge-cases near the front and end of the list (assume that you're not deleting the first two or last two nodes in the list, and the node to delete is in the list). If you define local variables like `curNode`, `prevNode`, etc. just state your assumptions about them. **However, please clearly state what per-node locks are held at the start of your process. E.g., "I start by holding locks on the first two nodes.".** Full credit will only be given for solutions that maximize concurrency.

```
// delete node containing value
void delete_node(Node* head, int value) {
```