Concurrency Worksheet

Consider the following code in a NotConcurrentStack class:

```
int[] array;
int size;
NotConcurrentStack(){...}
int pop() throws Exception
{
      if(size==0) throw new Exception("Can't pop on an empty stack!");
      return array[--size];
}
void push(int element) { array[size++]=element; }
```

This code contains data races; 'size' and the array cells can be simultaneously read/written or written/written.

a. Show an interleaving of calls that will result in a negative 'size' for the stack and a subsequent 'array out of bounds'-type exception.

b. Show an interleaving of calls that will result in a lost 'pushed' element.

c. How can these methods be fixed using 'synchronize'?

d. How can these methods be fixed using a ReentrantLock?

e. Why is the phenomenon of the 'deadlock' not relevant to the locked version of this class in part (d)? What sort of functionality would need to be added so that a deadlock may come up?

f. Assume we have fixed the above stack with one of the techniques in (c) or (d). Imagine now we had a scenario where several 'producer' threads were continually pushing elements onto a stack, and several 'consumer' threads were popping elements from the same stack and processing them. On a conceptual level, what would be a reasonable alternative to 'pop' throwing an exception when the stack is empty?