CSE 331 21au Section 5 Worksheet

Consider the following class `IntStack`:

```
/* IntStack is a LIFO stack of integers with a fixed capacity.
 * The most recently added integer is the first to be removed.
*/
public class IntStack {
  private int[] vals; // stack
  private int top;    // top

  public IntStack(int capacity) {
    vals = new int[capacity];
    top = 0;
  }

  public boolean push(int x) {
    if (top == vals.length)
      return false;
    vals[top] = x;
    top++;
    return true;
  }

  public int pop() {
    if (top == 0)
      throw new NoSuchElementException();
    top--;
    return vals[top];
  }

  public int size() {
    return top;
  }
}
```

1. What might the rep invariant for this class be? What about the abstraction function?

vals is not null, 0 <= top <= vals.length, and for 0 <= k < top, vals[k] has been initialized with stack elements. Note: it would be incorrect to say that vals[k] is not null, since int values cannot be null – they are not references.

vals[0..top-1] represents the stack of values, where vals[0] is the bottom element of the stack (s0), vals[top-1] is the top element on the stack (sn), and top is the number of items on the stack. If top=0, the stack is empty.

2. Consider the following two classes. Write an RI and AF for the Sequence class.

```
class IntPair {
  int num;
  int count;
  public IntPair(int n, int c) { num = n; count = c; }
}
/* Sequence represents a non-decreasing sequence of ints. Numbers
* can only be added to the end of the sequence.
*/
public class Sequence {
  private List<IntPair> list;
  private int length;

  public Sequence(int i) {
    list = new ArrayList<IntPair>();
    list.add(new IntPair(i, 1);
    length = 1;
  }

  public boolean add(int x) {
    int max = list.get(list.size() - 1).num;
    if (x < max){
      return false;
    } else if (x == max) {
      list.get(list.size() - 1).count += 1;
    } else {
      list.add(new IntPair(x, 1));
    }
    length++;
    return true;
  }

  public int get(int i) { // exclude bound checks
    int j = -1;
    while (i >= 0) {
      j++;
      i -= list.get(j).count;
    }
    return list.get(j).num;
  }
  public int getCount(int n) {
    for (int i = 0; i < list.size(); i++) {
       if (list.get(i).num == n) {
          return list.get(i);
        } else if (list.get(i).num > n) {
          return 0;
        }
    }
    return 0;
  }
}
```

CSE 331 21au Section 5 Worksheet

list is sorted by IntPair.num, each element of list has a unique IntPair.num
list is not null
for each element list.get(i), list.get(i) != null and list.get(i).count > 0
length = sum of list.get(i).count for all 0 <= i < list.size()

list.get(i).num represents the ith distinct integer appearing in the sequence, where
list.get(i).count is the number of times it appears in the sequence
length is the number of integers in the non-decreasing sequence