COS226 Week 2 Activity

1. *Generics, iterators, autoboxing, and mathematical analysis.*

```
Stack<Character> stack = new Stack<Character>();
stack.push('A');
stack.push('G');
stack.push('C');
stack.push('T');
for (char left : stack)
    for (char right : stack)
         StdOut.println(left + " " + right);
```

   (a) What does the above code fragment output to standard output?

   (b) If N characters are pushed onto the stack initially (instead of 4), how many lines of output does the above code fragment produce as a function of N?

2. *Stacks, queues, and amortized analysis.* Describe an implementation of a queue using two stacks, where each operation (*construct*, *push*, and *pop*) takes a constant *amortized* number of stack operations. Explain why, starting from an empty queue, any sequence of N queue operations takes proportional to N stack operations, in the worst case.

3. *Creating Iterators.* `ResizingArrayStack` includes a `ReverseArrayIterator`, which iterates through the stack in the normal LIFO order. Write an `ArrayIterator` that returns the items in FIFO order. You can assume the instance variables `N` and `a[]` exist, and that a[] is an array of type `Item`.

```
private class ArrayIterator implements ____________________ {

   private int i;
   public ArrayIterator()    { i = _____; }
   public boolean hasNext()  { return ____________;                          }
   public void remove()      { throw new UnsupportedOperationException();  }

   public _______ next() {
         if (!hasNext()) throw new NoSuchElementException();
         Item returnItem = ______;
         i = ______;
         return returnItem;
   }
```

4. *Unit testing.* Suppose that you are implementing a data type with the following API:

```
public class Stack<Item> implements Iterable<Item> {
    public void push(Item item)
    public Item pop()
    public boolean isEmpty()
    public int size()
}
```

Describe three ways to test the correctness of your implementation.

5. *Memory analysis.* Suppose you have a generic stack implemented using a linked list, as defined below:

```
public class Stack<Item> implements Iterable<Item> {
    private int N;          // size of the stack
    private Node first;     // top of stack

    private class Node {
        private Item item;
        private Node next;
    }
    ...
}
```

(a) How much memory (in bytes) does a single `Node` object use? Use the 64-bit memory cost model from Section 1.4 (starting on page 200). Do not include the memory for the item itself—this memory is allocated by the client and depends on the item type.

(b) How much memory (in bytes) does a `Stack` use to store N items? Do not include the memory for the items themselves.

(c) Repeat the previous question, but use tilde notation to simplify your answer.

-
-
-