Linear Structures

1. Exam Data
   a. Mean: 87/100. Max = 101/100
   b. Interpreting your grade:

<table>
<thead>
<tr>
<th>D</th>
<th>C</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>65</td>
<td>85</td>
<td>95</td>
</tr>
</tbody>
</table>

2. Most common misunderstandings:
   a. It is legal to **cast** any expression to any types in Java. A runtime error will occur if the actual objects do not have the right type.
   b. The **lower bound** is the shortest amount of time (fastest) an algorithm could possibly run. O(1) is a lower bound on everything, but it is not a very tight bound (that’s like saying the algorithm is infinitely fast, regardless of the size of the input).
   c. The **best and worst** case bounds occur based on the actual data. Whether you are in the best or worst case must be independent of the bound variables (e.g., \(n\)). Bounds must hold for some arbitrarily large \(n\). The best case cannot be “when \(n = 1\)”; the point is that for some large \(n\), what input data of that size would create good luck and let the algorithm terminate as soon as possible.
   d. For an **inductive proof** you must:
      i. Explain why the base case should have some value
      ii. Demonstrate that the base case does have that value
      iii. Assume that the theorem holds for \(n-1\)
      iv. Explain why the incremental change from \(n\) to \(n-1\) should have some value
      v. Demonstrate that the incremental change does in fact have that value
   e. Any iterative method can trivially be rewritten as recursive by rearranging the parts of the FOR loop into the arguments to a recursive call

3. Be lazy

4. **Stack** (all methods \(O(1)\))

   ```java
   public interface Stack<E> {
     public void push(E v); // addLast
     public E pop(); // removeLast
     public boolean isEmpty();
     public int size();
   }
   ```

5. **Tail recursion**
6. Example of a non-tail recursive method made iterative:

<table>
<thead>
<tr>
<th>Recursive</th>
<th>Iterative with Explicit Stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>public static void quickSort(int[] data) {</td>
<td>public static void quickSort2(int[] data) {</td>
</tr>
<tr>
<td>quickSort(data, 0, data.length - 1);</td>
<td>Stack&lt;Bounds&gt; stack =</td>
</tr>
<tr>
<td>}</td>
<td>new ArrayStack&lt;Bounds&gt;();</td>
</tr>
<tr>
<td>/** Sorts elements from L to R, inclusive */</td>
<td>stack.push(new Bounds(0, data.length - 1));</td>
</tr>
<tr>
<td>private static void quickSort(int[] data, int L, int R) {</td>
<td>while (! stack.isEmpty()) {</td>
</tr>
<tr>
<td>if (L &lt; R) {</td>
<td>// Get the next piece of work to do</td>
</tr>
<tr>
<td>int p = partition(data, L, R);</td>
<td>Bounds bounds = stack.pop();</td>
</tr>
<tr>
<td>quickSort(data, L, p - 1);</td>
<td>int L = bounds.L, R = bounds.R;</td>
</tr>
<tr>
<td>quickSort(data, p + 1, R);</td>
<td>// Push the two sub-arrays onto the stack</td>
</tr>
<tr>
<td>}</td>
<td>stack.push(new Bounds(L, p - 1));</td>
</tr>
<tr>
<td>}</td>
<td>stack.push(new Bounds(p + 1, R));</td>
</tr>
</tbody>
</table>

7. **Queue** (all O(1))

```java
public interface Queue<E> {
    public void pushLast(E v); // addLast
    public E popFirst();       // removeFirst
    public int size();
    public boolean isEmpty();
}
```

8. **Dequeue** (all O(1))

```java
public interface Dequeue<E> {
    public void pushFirst(E v);  // addFirst
    public void pushLast(E v);   // addLast
    public E popFirst();         // removeFirst
    public E popLast();          // removeLast
    public int size();
    public boolean isEmpty();
}
```

*How would you implement this using an array?*