INSTRUCTIONS:

- You have **50 minutes** to complete the exam.

- You will receive a deduction if you keep working after the instructor calls for papers.

- This exam is closed-book and closed-notes. You may not use any computing devices including calculators.

- Code will be graded on proper behavior/output and not on style, unless otherwise indicated.

- Do not abbreviate code, such as “ditto” marks or dot-dot-dot (“...”) marks. The only abbreviations that are allowed for this exam are: **S.o.p** for `System.out.print` and **S.o.pln** for `System.out.println`.

- You do not need to write import statements in your code.

- You may not use scratch paper on this exam. If you need extra space, use the back of a page.

- If you enter the room, you must turn in an exam before leaving the room.

- You must show your Student ID to a TA or instructor for your exam to be accepted.

- If you get stuck on a problem, move on and come back to it later.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Points</th>
<th>Score</th>
<th>Problem</th>
<th>Points</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>14</td>
<td></td>
<td>4</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>16</td>
<td></td>
<td>5</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>14</td>
<td></td>
<td>6</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>Σ</td>
<td></td>
<td>100</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Mechanical Missions.
This section tests whether you are able to trace through code of various types in the same way a computer would.

1. Recursive Tracing [14 points]

```java
public static void mystery(int n) {
    if (n == 1) {
        System.out.print(n);
    } else {
        System.out.print(n + " , ");
        if (n % 3 == 0) {
            mystery(n / 3);
        } else {
            mystery(2 * n - 1);
        }
    }
}
```

For each of the following, fill in the String printed when the given mystery method call is run.

<table>
<thead>
<tr>
<th>mystery Call</th>
<th>String Printed</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a) mystery(1)</td>
<td>1</td>
</tr>
<tr>
<td>(b) mystery(2)</td>
<td>2, 3, 1</td>
</tr>
<tr>
<td>(c) mystery(9)</td>
<td>9, 3, 1</td>
</tr>
<tr>
<td>(d) mystery(14)</td>
<td>14, 27, 9, 3, 1</td>
</tr>
<tr>
<td>(e) mystery(15)</td>
<td>15, 5, 9, 3, 1</td>
</tr>
</tbody>
</table>
2. ListNode Before & After [16 points]
Your task in this question is to write code to turn the “before” picture into the “after” picture. You may not change any existing node’s data value. You may not construct any new nodes. In each part, you may declare up to two new variables of type ListNode. Recall the ListNode class from lecture:

```java
public class ListNode {
    public final int data;
    public ListNode next;

    public ListNode(int data) {
        // constructor...
    }

    public ListNode(int data, ListNode next) {
        // constructor...
    }
}
```

<table>
<thead>
<tr>
<th>Before &amp; After</th>
<th>Your Code</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="a" alt="Diagram" /></td>
<td>list2 = list1.next.next; list1.next.next = null;</td>
</tr>
<tr>
<td><img src="b" alt="Diagram" /></td>
<td>list2.next = list1.next; list1.next = list2; list2 = null;</td>
</tr>
<tr>
<td><img src="c" alt="Diagram" /></td>
<td>list2.next.next = list1; list1 = list2.next; list2.next = list1.next.next; list1.next.next = list1.next.next.next; list2.next.next = null;</td>
</tr>
<tr>
<td><img src="d" alt="Diagram" /></td>
<td>LN three = list1.next.next; list2 = three.next; three.next = list1; three.next.next.next = null; list1 = three;</td>
</tr>
</tbody>
</table>
Programming Pursuits.
This section tests whether you synthesized various topics well enough to write novel programs using those topics.

3. Replacing the Min-utes [14 points]
Add a method replaceMin to the ArrayIntList class that removes the minimum element from the list and replaces the hole with the element from the end of the list (if such an element exists). If the list contains the minimum element multiple times, you should remove only the first copy. If the list is empty, it should remain unchanged.

Example Output

<table>
<thead>
<tr>
<th>Before list.replaceMin()</th>
<th>After list.replaceMin()</th>
</tr>
</thead>
<tbody>
<tr>
<td>[1, 3, 2, 7]</td>
<td>[7, 3, 2]</td>
</tr>
<tr>
<td>[-2, 5, 5, -5]</td>
<td>[-2, 5, 5]</td>
</tr>
<tr>
<td>[42, 5, 5, 99]</td>
<td>[42, 99, 5]</td>
</tr>
</tbody>
</table>

Implementation Restrictions
- You may not call any other methods on the ArrayIntList object (e.g., add, remove)
- You may not use a foreach loop on the ArrayIntList.
- You may not use any other data structures such as arrays, lists, queues, etc.
- Your solution should run in $O(n)$ time, where $n$ is the number of elements in the list.

```java
public class ArrayIntList {
    private int size;
    private int[] data;

    // Write Your Solution Here
}
```
Question #3: Extra Solution Space

Solution:

```java
public void replaceMin() {
    if (this.size > 0) { // Do nothing in case where size == 0
        // Find Min Part
        int minIdx = 0; // Use first element
        for (int i = 0; i < this.size; i++) {
            if (this.data[minIdx] > this.data[i]) {
                minIdx = i;
            }
        }
        // Replace Min Part
        if (this.size > 1) {
            this.data[minIdx] = this.data[this.size - 1];
        }
        this.size--;
    }
}
```
4. Write a recursive method called `bracketXX` that takes a `String` as a parameter and returns a new string with brackets around all occurrences of identical adjacent characters.

**Example Output**

<table>
<thead>
<tr>
<th>Method Call</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>bracketXX(&quot;odegaard&quot;)</code></td>
<td>&quot;odeg[aa]rd&quot;</td>
</tr>
<tr>
<td><code>bracketXX(&quot;mississippi&quot;)</code></td>
<td>&quot;mi[ss]i[ss]i[pp]i&quot;</td>
</tr>
<tr>
<td><code>bracketXX(&quot;who am I??&quot;)</code></td>
<td>&quot;who am I[??]&quot;</td>
</tr>
<tr>
<td><code>bracketXX(&quot;....-.-.--..&quot;)</code></td>
<td>&quot;[..] ..[-[..].-[..]]&quot;</td>
</tr>
<tr>
<td><code>bracketXX(&quot;oopsies&quot;)</code></td>
<td>&quot;[oo]psies&quot;</td>
</tr>
<tr>
<td><code>bracketXX(&quot;foobar&quot;)</code></td>
<td>&quot;f[oo]bar&quot;</td>
</tr>
<tr>
<td><code>bracketXX(&quot;&quot;)</code></td>
<td>&quot;&quot;</td>
</tr>
</tbody>
</table>

**Implementation Restrictions**

- You must use recursion.
- You may assume that the input `String` is not `null`.
- You may assume that no letter appears more than two times in a row in the input `String`.
- You may only use the `String` methods on the cheatsheet.
Question #4: Extra Solution Space

Solution:

```java
public static String bracketXX(String s) {
    // Base Cases
    if (s.length() < 2) { return s; }
    // Found Case
    else if (s.charAt(0) == s.charAt(1)) {
        return '[' + s.substring(0, 2) + ']' + bracketXX(s.substring(2));
    } else { return s.charAt(0) + bracketXX(s.substring(1)); }
}
```
5. I’m The Map! [16 points]
Write a method called `indexMap` that takes a `List` of `Strings` as a parameter and returns a map that associates each `String` from the `List` to a `Set` of indices that string occurs at. The map returned by your method should be ordered alphabetically, and the sets that each string maps to should be as fast as possible.

**Example Output**

<table>
<thead>
<tr>
<th>Input List</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>[]</td>
<td>{}</td>
</tr>
<tr>
<td>[to, be, or, not, to, be]</td>
<td>{be=[1, 5], not=[3], or=[2], to=[0, 4]}</td>
</tr>
</tbody>
</table>

**Implementation Restrictions**
- You may assume that the list passed is not null (and that no value in the list is null).
- Your method should not modify the list of words passed as a parameter.
- You may not create and structures other than the ones described above.
- Your method should be reasonably efficient.
Question #5: Extra Solution Space

Solution:

```java
public static Map<String, Set<Integer>> indexMap(List<String> list) {
    // Return Value
    Map<String, Set<Integer>> map = new TreeMap<String, Set<Integer>>();
    // Looping through the List
    for (int i = 0; i < list.size(); i++) {
        String s = list.get(i);
        if (!map.containsKey(s)) {
            map.put(s, new HashSet<Integer>());
        }
        map.get(s).add(i);
    }
    return map;
}
```
6. Number Me Once... [20 points]

Write a method `numberify` that takes a Stack of digits (ints 0 through 9) separated by -1's as a parameter and returns a Queue of Strings with the numbers made up of those digits with leading zeroes removed. You should remove all leading zeroes from each number. If all digits in the number are zero, you should output zero.

**Example Output**

```
stack1
   ↓
  1
  9
  0
-1
 0
-1
 1
-1

numberify(stack1) → output ← "390" "0" "1"

stack2
   ↓
  1
  2
  3
  4
-1
 0
 0
-1
 5
1

numberify(stack2) → output ← "1234" "1" "5"
```

**Implementation Restrictions**

- You may assume that the only numbers in the input stack are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, and -1.
- You may assume the input stack has no adjacent -1's.
- You may assume the input stack is not null or empty.
- You may assume the input stack has a -1 at the bottom, and it does not have a -1 at the top.
- When your method returns, the input stack must be unchanged.
- You may create at most two queues and no other data structures.
- You may not use recursion to solve this problem.
- Your solution must run in $O(n)$ time, where $n$ is the size of the stack.
Question #6: Solution Space

Solution:

```java
public static Queue<String> numberify(Stack<Integer> digits) {
    Queue<String> output = new FIFOQueue<String>();
    Queue<Integer> temp = new FIFOQueue<Integer>();

    while (!digits.isEmpty()) {
        while (digits.peek() == 0) {
            temp.enqueue(digits.pop());
        }
        String num = "";
        while (digits.peek() != -1) {
            num += digits.peek();
            temp.enqueue(digits.pop());
        }
        digits.pop();
        if (num.length() == 0) {
            output.enqueue("0");
        } else {
            output.enqueue(num);
        }
    }

    while (!temp.isEmpty()) {
        digits.push(temp.dequeue());
    }

    while (!digits.isEmpty()) {
        temp.enqueue(digits.pop());
    }

    while (!temp.isEmpty()) {
        digits.push(temp.dequeue());
    }

    return output;
}
```