

Solution to CSE143X Section #5 Problems

1. One possible solution appears below.

```
public void splitStack(Stack<Integer> s) {
    Queue<Integer> q = new LinkedList<>();
    // transfer all elements from stack to queue
    int oldLength = s.size();
    while (!s.isEmpty()) {
        q.add(s.pop());
    }

    // transfer negatives from queue to stack
    for (int i = 1; i <= oldLength; i++) {
        int n = q.remove();
        if (n < 0) {
            s.push(n);
        } else {
            q.add(n);
        }
    }

    // transfer nonnegatives from queue to stack
    while (!q.isEmpty()) {
        s.push(q.remove());
    }
}
```

2. One possible solution appears below.

```
public void stutter(Stack<Integer> s) {
    Queue<Integer> q = new LinkedList<>();
    while (!s.isEmpty()) {
        q.add(s.pop());
    }
    while (!q.isEmpty()) {
        s.push(q.remove());
    }
    while (!s.isEmpty()) {
        q.add(s.pop());
    }
    while (!q.isEmpty()) {
        int n = q.remove();
        s.push(n);
        s.push(n);
    }
}
```

3. One possible solution appears below.

```
public boolean equals(Stack<Integer> s1, Stack<Integer> s2) {
    if (s1.size() != s2.size()) {
        return false;
    } else {
        Stack<Integer> s3 = new Stack<>();
        boolean same = true;
        while (same && !s1.isEmpty()) {
            int num1 = s1.pop();
            int num2 = s2.pop();
            if (num1 != num2) {
                same = false;
            }
        }
    }
}
```

```

        s3.push(num1);
        s3.push(num2);
    }
    while (!s3.isEmpty()) {
        s2.push(s3.pop());
        s1.push(s3.pop());
    }
    return same;
}
}

```

4. One possible solution appears below.

```

public void reverseHalf(Queue<Integer> q) {
    Stack<Integer> s = new Stack<>();
    int oldLength = q.size();
    // transfer elements in odd spots to stack
    for (int i = 0; i < oldLength; i++) {
        if (i % 2 == 0) {
            q.add(q.remove());
        } else {
            s.push(q.remove());
        }
    }
    // reconstruct list, taking alternately from queue and stack
    for (int i = 0; i < oldLength; i++) {
        if (i % 2 == 0) {
            q.add(q.remove());
        } else {
            q.add(s.pop());
        }
    }
}
}

```

5. One possible solution appears below.

```

public boolean isPalindrome(Queue<Integer> q) {
    Stack<Integer> s = new Stack<>();
    for (int i = 0; i < q.size(); i++) {
        int n = q.remove();
        q.add(n);
        s.push(n);
    }

    boolean ok = true;
    for (int i = 0; i < q.size(); i++) {
        int n1 = q.remove();
        int n2 = s.pop();
        if (n1 != n2) {
            ok = false;
        }
        q.add(n1);
    }
    return ok;
}
}

```

6. One possible solution appears below.

```

public boolean isConsecutive(Stack<Integer> s) {
    if (s.size() <= 1) {
        return true;
    } else {
        Queue<Integer> q = new LinkedList<>();
        int prev = s.pop();

```

```

        q.add(prev);
        boolean ok = true;
        while (!s.isEmpty()) {
            int next = s.pop();
            if (prev - next != 1) {
                ok = false;
            }
            q.add(next);
            prev = next;
        }
        while (!q.isEmpty()) {
            s.push(q.remove());
        }
        while (!s.isEmpty()) {
            q.add(s.pop());
        }
        while (!q.isEmpty()) {
            s.push(q.remove());
        }
        return ok;
    }
}

```

7. One possible solution appears below.

```

public void reverseByN(Queue<Integer> q, int n) {
    Stack<Integer> s = new Stack<>();
    int times = q.size() / n;
    int extra = q.size() % n;
    for (int i = 0; i < times; i++) {
        for (int j = 0; j < n; j++) {
            s.push(q.remove());
        }
        while (!s.isEmpty()) {
            q.add(s.pop());
        }
    }
    for (int i = 0; i < extra; i++) {
        s.push(q.remove());
    }
    while (!s.isEmpty()) {
        q.add(s.pop());
    }
}

```