1) ArrayList Mystery
a) [2, 5, 2]
b) [5, 4, 9]
c) [4, 3, 1, 0]

2) Recursion Mystery
a) 1
b) 10
c) 10011
d) -1101

3) Collection

```
public static Map<String, List<Double>> studentGrades(Map<String, Map<String, Double>>
                                                                                classes)
{
      Map<String, List<Double>> result = new TreeMap<String, List<Double>>();
      for (String teacher : classes.keySet()) {
            Map<String, Double> grades = classes.get(teacher);
            for (String student : grades.ketSet()) {
                  if (result.containsKey(student)) {
                        result.put(student, new ArrayList<Double>()) ;
                  }
                  result.get(student).add(grades.get(student));
            }
      }
      return result;
```

4) Stacks and Queues

```
public boolean hasAlternatingParity(Stack<Integer> stack) {
      if (stack.isEmpty()) {
            return true;
      }
      int prev = stack.pop();
      Queue<Integer> queue = new LinkedList<Integer>();
      queue.add(prev);
      boolean test = false;
      while (!stack.isEmpty()) {
            int next = stack.pop();
            queue.add(next);
            if (next % 2 != prev % 2) {
                  test = false;
            }
            prev = next;
      }
      q2s(queue, stack);
      s2q(stack, queue);
      q2s(queue, stack);
      return test;
}
```

5) LinkedIntList implementor

```
public boolean removeFirstNegative() {
     if (front != null) {
          if (front.data < 0) {
               front = front.next;
               return true;
          }
          ListNode current = front;
          while (current.next != null) {
               if (current.next.data < 0) {
                    current.next = current.next.next;
                    return true;
               }
               current = current.next;
          }
     }
     return false;
}
```

6) LinkedIntList client

```
public static int removeAllNegatives(LinkedIntList list) {
     int count = 0;
     while (list.removeFirstNegative()){
          count++;
     }
     return count;
}
```

7) recursion programming

```
    public boolean sameDashes(String s1, String s2) {
          if (s1.length() != s2.length()) {
                throw new IllegalArgumentException();
           } else if (s1.length() == 0 && s2.length() == 0) {
                return true;
           } else if (s1.charAt(0) == '-' || s2.charAt(0) == '-') {
               return s1.charAt(0) == s2.charAt(0) &&
               sameDashes(s1.substring(1), s2.substring(1));
           } else {
               return sameDashes(s1.substring(1), s2.substring(1));
          }
    }
```