1. Consider the following recursive algorithm:
   
   ```java
   /**
   * require: d1, d2 > 0
   * ensure: ??
   */
   public int mystery(int d1, int d2){
       if (d2 == 0)
           return d1;
       else
           return mystery(d2, d1 % d2);
   }
   
   a. First, get the idea of what this algorithm is by computing the result to the call mystery(28, 24). You must show each of the steps that algorithm takes to get the answer. (more help, mystery(12, 40) = 4, and mystery(12, 30) = 6). (8 points)
   mystery(28, 24) = mystery(24, 4) = mystery(4, 0) = 4.
   
   b. What is the general case of this algorithm (i.e. what does the algorithm compute)? (4 points)
   Using modulo operation between the two arguments the algorithm computes the largest common divisor.
   
   c. What is the base case for this algorithm? (4 points)
   d2 is zero. In which case the answer is d1.
   
   d. Does the general case reduces to the base case? why? (4 points)
   The recursive call happens with the the call mystery(d2, d1%d2); the value of d1%d2 can be zero when d2 is a multiple of d1, if d2 is not a multiple of d1 there will be a remainder > 0. Either the remainder is a multiple of d2 or its successive remainders will reduce until 1 or 0. when d2 is one the next call with have d2 0.
   
2a. Name the two search algorithms for lists studied and choose one to give a description of its algorithm and provide a recursive implementation of the other. (20 points)
   Linear search and binary search. (check your notes or book for the linear search algorithm, found in List as indexOf() method; and a handout I gave for recursive binary search.)
   
3a. Define Exception. (5 points)
   A mechanism used by Java to detect, and handle unexpected errors or failures.
   
   b. Name the two types of Exceptions in Java and clearly state their difference. (6 points)
   Runtime exceptions and checked exceptions. Runtime exceptions do not have to be caught by clients receiving it. Checked exceptions must be caught or client method must specifies that it throws it.
   
   c. Give 3 examples of built-in exceptions classes already declared in Java and qualify them using b. (6 points)
   Runtime exceptions: NullPointerException, ArithmeticException.
   Checked exception: IOException, FileNotFoundException.
   
   d. Write an exception class to model a failure due to “sum out of bounds exception”, used to report on an error due to a sum falling outside a given integer range where the two bounds (low and high) are integers. (8 points)
   ```java
   public class SumOutOfBoundsException {
       private int low;
       private int high;
       public SumOutOfBoundsException(String message, int low bound, int highBound){
           super(message);
           low = lowBound;
           high = highBound;
       }

       public int lowBound(){ return low; }
       public int highBound(){ return high; }
   }
   ```
e. Assume you have a class that has the following instance variables

```java
private TUI tui;
private ControllerError error;
private String number;
```

Write a method, called `smallInt` that uses TUI's `update` method to get a string from input to compute the addition of its digits. TUI's `update` sets model's instance variable `number`. If number does not represent a legal positive integer or addition falls outside the bounds (1..10) you must report it as an error to a `ControllerError` via the method "update(String message)" and you must give user a new try via TUI's `update`. The `smallInt` method you are asked to write uses `digitSum` below. It also uses a method, call it `validNumber`, that checks that a string contains only digits. Specify `validNumber` but do not implement it, only use it in the method `smallInt`. (20 points)

```java
/**
 * Require: digits is a string representing a positive integer.
 * Ensure: digitSum(digits) > 0 and digitSum(digits) < 10.
 */
public int digitSum(String digits) throws SumOfBoundsException {... // assume implementation}
```

**solution:**

```java
public int smallInt() {
    int done = false;
    int sum;
    while (!done){
        try{
            tui.update();
            if (!isValid(number))
                throw new ArithmeticException();
            sum = digitSum(number);
            done = true;
        }catch(ArithmeticException e){
            error.update("Invalid number. Not all characters are digits." + number);
        }
        catch(SumOfBoundsException e){
            error.update(e + "Addition must be between the followign bounds: " + low bound >> " + high bound>> " + e.highBound());
        }
    }
    private boolean isValid (String number){ ...
}
```

4. Given a list and an order on elements of the list, specify and implement the most generic version of the algorithm that computes the largest element in the list. **Require**: list must have at least one element. (12 points)

```java
public <Element> Element largest(List<Element> list, Order<Element> order){
    Element largest = list.get(0);
    int length = list.size();
    int i = 1;
    while( i < length){
        if (order.inOrder(largest, list.get(i))
            largest = list.get(i);
        i = i + 1;
    }
    return largest;
}
```
5a. Provide a brief narrative description of bubble sort algorithm. (3 points)

Bubble sort is based on making \( n - 1 \) passes through the list. Each pass moves large elements towards the end of the list. The first pass finds the largest element which ends up in the end of the list; for the second finds the second largest and places it as next to last element; for the third pass it finds the third largest and places it next to the second largest, etc. During a pass through the list it compares adjacent elements. If \( (\text{list.get(i)}, \text{list.get(i+1)}) \) it swaps them. This way large elements get found and placed to the end of the list.

b. Provide a brief narrative description of the quicksort algorithm. (3 points)

Algorithm is a recursive one that chooses a pivot element and places it in the list in such a way that elements less than it are to the left of it and the others are to the right of it. Thus the pivot elements ends up in its correct place and then proceeds to sort the same way the sublist made up those elements to the left of it and those elements to the right of it.

c. Via an interface provide the most general specification of a sorting algorithm. Name the interface `Sorter`, and name the method `sort`. (4 points)

```java
public interface Sorter<Element> {
    public void sort(List<Element>, Order<Element> order);
}
```

d. Given a list of books produce two lists; first, the list of books in ascending order by ISBN number (publisher’s number that identifies books) and the second in descending order by publication date. Assume the following: The class `Book` has two queries one being `isbn()` returning a string representing the ISBN number; the other `date()` which returns a string of the form `month/day/year` (4/12/2005). Also assume that the interface `Sorter` has been implemented by different sorts using the common names seen in lectures. You need only to write the code necessary to invoke the sort. (Not to write the sort method; you may write on back of this page if needed). (12 points)

Assume `BubbleSorter` is an implementation of the interface.

Assume list is a list of books:

```java
List<Book> list = new list.copy();
Sorter sorter = new BubbleSorter();
sorter.sort(list, new Order<Book>{
    public boolean inOrder (Book b1, Book b2){
        b1.isbn().compareTo( b2.isbn()) < 0; // b1 < b2, wrt isbn()
    }
});

sorter.sort(list2, new Order<Book>{
    public boolean inOrder (Book b1, Book b2){
        b1.date().compareTo( b2.date()) > 0; // b1 > b2, wrt date()
    }
});
```

The order is being supplied using inner classes; you could also just implement the Order classes directly.