**Question 1**. [5 points] State a big-O upper bound on the worst case running time of the given method, where the problem size $N$ is the number of elements in the `list` parameter. You can assume that the call to `equals` is $O(1)$. Explain your answer briefly.

```
public static<E> int find(ArrayList<E> list, E elt) {
  for (int i = 0; i < list.size(); i++) {
    if (list.get(i).equals(elt)) {
      return i;
    }
  }
  return -1;
}
```

**Question 2**. [5 points] State a big-O upper bound on the worst case running time of the given method, where the problem size $N$ is the number of elements in the `list` parameter. You can assume that the call to `equals` is $O(1)$. Explain your answer briefly.

```
public static<E> int find(LinkedList<E> list, E elt) {
  for (int i = 0; i < list.size(); i++) {
    if (list.get(i).equals(elt)) {
      return i;
    }
  }
  return -1;
}
```

**Question 3**. [5 points] Complete the following generic method. It takes two values of type `E`, and a `Comparator` that can compare values of type `E`, and returns the smaller of the two values.

Hint: use the comparator's `compare` method to compare the two values.

```
public static<E> E min(E val1, E val2, Comparator<E> comp) {
```

**Question 4**. [5 points] Indicate whether a stack, or a queue would be best for implementing each of the following:

(a) A keyboard buffer:

(b) A text editor "undo" operation:

(c) Buffering events in a video game:

(d) Capturing the most recent 30 seconds of security video:

(e) Reversing the contents of a list:

**Question 5**. [5 points] For a certain card game, the value of each card in a Suit is multipied by its Suit's multiplier factor (an integer). Using the given Suit enum, declare and create a Map that allows the user to retrieve a Suit's multiplier factor, where SPADES = 8X, HEARTS = 4X, DIAMONDS = 2X and CLUBS = 1X. Be sure to show how the map is populated.

```
public enum Suit {
  SPADES,
  HEARTS,
  DIAMONDS,
  CLUBS
}
```

**Question 6**. [5 points] Given the following two enums, declare and create a Map that cross-references a Suit with its Color. Be sure to show how the map is populated.

```
public enum Suit {
  SPADES,
  HEARTS,
  DIAMONDS,
  CLUBS
}

public enum SuitColor {
  RED,
  BLACK
}
```

**Question 7**. [10 points] Consider the following method to compute an iteration count to test whether a complex number is in the Mandelbrot set:

```
public int computeIterCount(Complex c){
  Complex z = new Complex(0, 0);
  int count = 0;

  while (z.getMagnitude() < 2 && count < MAX_COUNT){
    z = z.multiply(z).add(c);
    count++;
  }
  return count;
}
```

(a) Implement a recursive version of this computation:

```
// Recursive version
public int computeRecursiveIterCount(Complex c, Complex z, int count){









}
```

Make sure you check a base case or base cases, that the recursive call(s) work towards a base case, and that the result of the recursive call or calls is extended to be a solution for the overall problem.

(b) Is it a good idea to implement this computation recursively? Briefly explain why or why not.

**Question 8**. [5 points] Consider the following methods, which attempt to compute the $n$'th member of the Fibonacci sequence using memoization:

```
public static int fib(int n) {
  return fibMemo(n, new int[n+1]);
}

private static int fibMemo(int n, int[] memo) {
  if (n == 0 || n == 1) {
    return 1;
  } else {
    if (memo[n] == 0) {
      memo[n] = fib(n-2) + fib(n-1);
    }
    return memo[n];
  }
}
```

Breifly explain the error in this code, and how to fix it.

**Question 9**. [10 points] Given the constructor for the `CalculateTask` (which implements `Runnable`)

```
public CalculateTask(int[] arr, int start, int end)
```

where `start` and `end` specify the portion of array `arr` to process for each thread, complete the following fork/join code to process the array in parallel, using `numThreads` threads.

Hint: think about how you can divide up the elements of the array to split the work equally between tasks.

```
Thread[] threads = new Thread[nummThreads];
CalculateTask[] tasks = new CalculateTask[numThreads];

// create CalculateTasks and specify range to be processed
for (int i = 0; i < numThreads; i++) {









}

// create Threads, initialize from tasks array,
// and start the threads
for (int i = 0; i < numThreads; i++) {









}

// wait for the threads to complete
try {
  for (int i = 0; i < numThreads; i++) {









  }
} catch (InterruptedException e) {
  System.out.println("Error waiting for thread to complete: " + e);
}
```

# Programming Questions

To get started, use a web browser to download the zipfile as specified by your instructor. Import it as an Eclipse project using File → Import... → General → Existing Projects into Workspace → Archive file.

**Important**: You may use the following resources:

- The textbook
- The lecture notes posted on the course web page
- Your previous labs and assignments
- Your work on CloudCoder and Codingbat exercises

Do not open any other files, web pages, etc.

**Question 10**. [25 points] Complete the `countOccurrences` method in the `Q10` class. This method takes a `String` and a `char` value, and returns the number of times that the specified character occurs in the string. For example, the call

```
Q10.countOccurrences("Hello, world!", 'o')
```

should return 2, since the string has 2 occurrences of the character '`o`'.

**Important**: Your implementation *must* be recursive. Do not use a loop.

There are JUnit tests in `Q10Test` class. Make sure that when you finish all of the tests pass. The tests will probably also be useful to help you understand how the `countOccurrences` method is intended to work.

Hints:

- The method should count the number of characters in the the string that are *exactly* equal to the specified character; you can use the `==` operator to compare `char` values for equality
- Think about an appropriate base case or cases
- Think about how to find a subproblem or subproblems which, if solved recursively, will help you solve the overall problem
- Make sure that the subproblem or subproblems work towards a base case
- The `length` method returns the number of characters in a string
- The `charAt` method returns the character at a specified index in a string (0 being the index of the first character)
- The `substring` method returns a string containing all characters from a specified start index (inclusive) to the specified end index (exclusive); if the end index is omitted, the string returned contains all characters from the start index to the end of the string

**Question 11**. [20 points] Complete the `countSuits` method in the `Q11` class. It should return a `Map` in which the keys are `Suit` values and the values corresponding to keys are `Integer`s, such that for each key (suit), the corresponding integer value is the number of `Card`s in the `hand` parameter (a list of `Card`s) which had that particular suit.

There are JUnit tests in `Q11Test` class. Make sure that when you finish all of the tests pass. The tests will probably also be useful to help you understand how the `countSuits` method is intended to work.

Hints:

- You can call the `getSuit` method on a `Card` object to get the card's `Suit` value
- Can can create either a `HashMap` or `TreeMap` to return as the result of the method
- You can use the `containsKey` method to check a map to determine whether or not a key is present
- You can use the `get` method to retrieve the value associated with a specified key
- You can use the `put` method to create or update an association between a specified key and value

---

When you are ready to submit your code, export the **CS201_Exam03** project as a zip file and upload it to the Marmoset server as **exam03**:

`https://cs.ycp.edu/marmoset`