CS125 MT3 Fall 2011 Solution

1. Recursive Concepts – 14 points (2 points each)

```
1 public static int mystery(int a) {
2  if(a == 256) return 3;
3  return 1 + 2 * mystery(a*4);
4 }
```

a. Which line in the above code implements a Recursive Case? Line 3

b. Which line in the above code implements a Base Case? Line 2

c. Circle one italicized correct word within the {}'s to best describe the structure of the recursion.

"`mystery(1)` creates a {***chain***} of activations"

d. Carefully explain why the `mystery` method is forward recursive not tail recursive. Your answer should highlight which parts of the method cause the method to be forward recursive.

The mystery function is forward recursive because computation (doubling and add one) will be performed after the recursive call completes

e. `mystery(255)` does not return an integer result. Complete the following sentence to explain why.

"`mystery(255) is an example of infinite recursion`"

f. Refactor line 3 so that `mystery` uses a tree of activations:

New Line 3: `return 1 + mystery(a*4)+ mystery(a*4);`

g. Which one of the following best describes the refactored mystery function, that has a tree of activations, when compared to the original implementation?

C. `mystery(1)` will now take *more* time to calculate the same result. **Your answer: C**

2. Tracing code – 11 points

```
1   public static int foo(int a, int b) {
2     if (a==b) {
3          return b;
4     }
5     int min = Math.min(a,b), max = Math.max(a,b);
6     return foo(min,min) + foo( max - min , min);
7  }
```

a. Which one of the following statements best describes the method 'foo' in the above code?

C. Each activation will have its own local, temporary variable *min*. Your answer: **C**

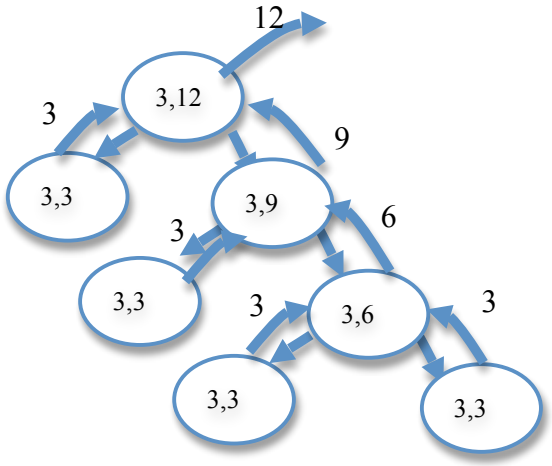
b. Which one of the following statements best describes the execution of `foo(1,0)`?

D. Example of infinite recursion. In practice an exception will be thrown. **D**

c. Create an activation diagram below for the execution of `foo(3,12)`. For full marks ensure your activation diagram includes:

d. Use your diagram to determine the returned value of `foo(3,12)` ? **12**

e. How many times is **`foo`** activated (called), including the first "`foo(3,12)`" ? **7**



3. Linked Lists – 15 points (5 points each)

```
public class Link {
   private int value; // Always non-null
   private Link next; // null for the last link
   public Link(int v,Link n) {this.value = v; next = n;}
   public Link insert(int v) {
       if( v < this.value) return new Link( v, this );
       if( next != null) next = next.insert(v);
       else next = new Link( v, null );
       return this; //we don't need to move.
    }
public int toSum() {
  if(next == null) return v;
  return v+next.getSum();
}
public int countEven(int acc) {
   if(v%2 ==0) acc++;
   if(next == null) return acc;
   return next.countEven(acc);
}
}
```

4. Twenty Questions. Tree Recursion – 15 points

The class below models a tree of Yes-No Questions for the game "20 Questions". Each question object is referenced just once in this network (i.e. it's a tree).

```
public class Question {
 private String text;
 private Question yes; // possibly null
 private Question no; // possibly null
 // assume a constructor is written to initialize the above instance variables.
```

a. Write a recursive instance method *count* that takes no parameters and returns an integer. Your method will recursively visit every question in the tree. Return the total number of question objects that have both *yes* and *no* set to `null`.

```
public int count() {
  if(yes == null && no == null ) return 1;
  int sum = 0;
  if(yes != null) sum += yes.count();
  if(no != null) sum += no.count();
  return sum;
}
```

b. Write an instance method *max* that takes no parameters and returns a reference to a question object: Return the question with the longest text.

```
public Question max() {
  Question best = this;
  if(yes != null) {
    Question temp = yes.max();
    if(temp.text.length() > best.text.length()) best = temp;
   }
   if(no != null) {
     Question temp = no.max();
     if(temp.text.length() > best.text.length()) best = temp;
   }
   return best;
  }
```

5. Binary Search – 15 points

By analyzing tagged images and web pages, you create a simple database - an array of *Pair* objects (see below). Each *Pair* object contains a unique Facebook user name in lowercase and the likely UIUC email of the user:

```
public class Pair {
       public String name;
       public String uiuc;
}
```

a. Complete the following recursive binary search method to quickly find the relevant *Pair* object in an array. Use a 'divide and conquer' approach: Assume the given array is already sorted alphabetically by the *name* variable. Search the array only between loth and hith indices for the *Facebook user* that matches the search parameter '*key*' . Return `null` if no *name* matches the search key. All values in the array are valid and non-null.

```
class Lookup {
   public static Pair search(Pair[] data, String key, int lo, int hi){
     int mid  = (lo+hi)/2;
     if(lo>hi )return null;
     String n = data[mid].name;
     if(n.equals(key)) return data[mid];
     if( n.compareTo(key) > 0)
          return search(data,key,lo,mid-1);
     return search(data,key,mid+1,hi);
   }
```

```
"abc".compareTo("aba") returns > 0
"abc".compareTo("abc") returns 0
"abc".compareTo("abd") returns < 0
```

b. Create a *wrapper class-method* `toEmail` that returns a String and takes two parameters: '`data`' – a sorted array of `Pair` objects, '`key`' – a string which is the name to find. Return the corresponding email, or a question mark, "?", if the key does not match any names. Use the `search` method above to perform the search of the array.

```
     public static String toEmail(Pair[] data, String key) {
          Pair p= search(data,key, 0, data.length-1);
          if(p!= null) return p.uiuc;
          return "?";
     }
```

6. Recursive Searching and Sorting Concepts – 15 points

a. Complete the following recursive method to return the *index* of the smallest value of the sub-array {`data[lo]`, `data[lo+1]`, `...` up to `data[hi]`}. Assume 0 <= lo <= hi < data.length and the array values are unique.

```
public static int findMin(double[] data, int lo, int hi) {
     if(lo==hi) return lo;
     int best = findMin(data,lo+1,hi);
      if(data[best]<data[lo]) return best;
     return lo;
}
Alternative tail recursive:
     if(lo==hi) return lo;
     if(data[lo]<data[hi]) return findMin(data,lo,hi-1);
     return findMin(data,lo+1,hi);
```

b. Which one of the following best describes a *Selection sort* on a pile of playing cards?

D. Put all of the unsorted cards down. Keep picking up the smallest valued card from the remaining unsorted cards and append it to the cards held in your hand.

Your Answer: **D**

c. Consider the following array of 8 values for sorting using Selection Sort (low to high).

7	3	11	17	4	12	14	35

Calculate the values in the array after the 4th swap has completed. Write your answer below:

3	4	7	11	17	12	14	35

d. Once **all** 8 array values have been sorted and **all** swaps have completed,

how many times has the value '7' moved to a new position? 3

how many times has the value '35' moved to a new position? 0 (1 also acceptable)

how many times has selection sort called *findMin* (ie found the index of a minimum)? 7 (8 also acceptable)

e. Choose the best description of this code:

```
for(int i=0;i<data.length;i++) {
  int m = i;
  for(int j = i; j < data.length; j++)
    if(data[j] < data[m]) m = j;
  swap(data,i,m);
}
```

D. Iterative Selection sort

7. Recursive Dreaming. Selection Sort – 15 points

```
/** Swaps values at data[i] and data[j] */
public static void swap(double[] data, int i, int j) {
    double temp = data[i];
    data[i] = data[j];
    data[j]=temp;
}
/** Sorts all values (smallest first) between lo-th and hi-th index (inclusive) using a recursive
selection sort. */
public static void sort(double[] data, int lo, int hi) {
    if(lo<hi) {
           swap(data,lo, findMin(data,lo,hi));
           sort(data,lo+1,hi);
    }
}
/** A wrapper method to sort the entire array using selection sort. This method just calls the recursive
method above.*/
public static void selectionSort (double[] data) {
     sort(data,0,data.length-1);
}
```