University of Illinois at Urbana-Champaign
Department of Computer Science

Third Examination Fall 2011

CS 125 Introduction to Computer Science.
90 minutes permitted

First name: _____ Last name: _____

NetID: _ _ _ _ _ _ _ _ @ illinois.edu
(please write legibly)

Discussion Section: AY_____
(We will return your exam manuscript to you in this section)

```
AYA Tues 09:00am-10:50
AYB Tues 11:00am-12:50
AYC Tues 01:00pm-02:50
AYD Tues 03:00pm-04:50
AYE Tues 05:00pm-06:50
AYF Tues 07:00pm-08:50
AYG Wed 09:00am-10:50
AYH Wed 11:00am-12:50
AYK Wed 01:00pm-02:50
AYI Wed 03:00pm-04:50
AYJ Wed 05:00pm-06:50
```

Do not start until instructed to do so.
Failure to follow the proctors instructions can result in a failing (zero) grade.

• The exam proctors will not answer any technical questions. If you believe a question is ambiguous, write your assumptions on your sheet and answer accordingly.

• This is a closed book and closed notes exam. No electronic aids are allowed.

• You should have **10** pages total including one empty and one scratch page. The last sheet is scratch paper; you may detach it while taking the exam. You must hand in the whole manuscript including the scratch sheet before you leave.

• **This exam tests your understanding of recursion**: Unless specifically instructed you may not use loops – 'for', 'while', or 'do…while' – in this exam. Also, you may not create any additional unspecified class or instance variables or class or instance methods.

• You are not allowed to use the break, continue, or switch statements on this exam.

• Unless we say otherwise in the specific problem, you can assume all values entered by the user will be acceptable input for that program.

• For full marks correct syntax is required: Ensure all statements include a semicolon and the correct use of upper/lower case, single quotes and double quotes.

Problem	Points	Score	Grader
1	14		
2	11		
3	15		
4	15		
5	15		
6	15		
7	15		
Total	100		

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

Consider the following recursive function.

```
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 _____

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

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

"`mystery(1)` creates a { *circle* *base* *runtime* *sandwich* *chain* *spray* } 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.

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

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

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

New Line 3:

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

A. `mystery(1)` is not well defined.
B. `mystery(1)` will now take the *same* amount of time to calculate the same result.
C. `mystery(1)` will now take *more* time to calculate the same result.
D. `mystery(1)` will now take *less* time to calculate the same result.
E. `mystery(1)` will now return a different result value.

Your answer: _____

2. Tracing code – 11 points

Consider the following method 'foo':

```
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?

A. Causes a runtime error because class methods cannot be recursive.
B. The method 'foo' is an example of an iterative method.
C. Each activation will have its own local, temporary variable *min*.
D. The value of *min* is shared over all activations
E. The variable *min* is only initialized by the first activation of 'foo'.

Your answer: _____

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

A. Returns a small positive integer when executed on a modern Java virtual machine.
B. Returns a small negative integer when executed on a modern Java virtual machine.
C. Requires multiple threads so it causes a compile error on modern Java compilers.
D. Example of infinite recursion. In practice an exception will be thrown.
E. None of the above.

_____

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

- The method parameter values for each execution of `foo`.
- Label the return arcs with the returned value, *including the returned value of foo(3,12)*.

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

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

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

The following code is a linked list of integers. Complete the linked list by writing three recursive instance methods described below. These methods are called on the head of the linked list. Do not use any loops in this question or create any other class or instance variables or any other class or instance methods (but local/temporary variables are allowed).

insert takes one integer parameter (the value to insert) and returns a reference to a Link (either the current link or the new link, whichever has the lower value). Insert the value such that the links are sorted by value (lowest values first).

toSum takes no parameters. Return the sum of all of the values.

countEven takes one parameter (an integer accumulator, initially zero) and returns an integer – the total number of links with an even value (hint: "`%2`" may be useful). For full marks use tail recursion.

```
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;}

// Complete the 'insert' recursive instance method here:
   public Link insert(int v) {
        if( v < this.value) return new Link( v, _____ );

        if( next != _____ ) next = _____;

        else next = new Link( v, _____ );

        return _____; //we don't need to move.
    }
// Write 'toSum' recursive instance method here:









// Write 'countEven' tail recursive instance method here: (continue overleaf if necessary)









}
```

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). Do not write any loops, or create any other methods, or create any other class or instance variables (temporary/local variables are allowed). Assume the methods below are initially called on the top-most item of the 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`.

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.

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. Do not use loops or create any other methods or any other class variables. Local, temporary variables are allowed.

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

```
class Lookup {

   public static Pair search(Pair[] data, String key, int lo, int hi){
```

   } // end method

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.

Write your wrapper method here:

```
} // end class
```

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 and including `data[hi]`}.
Do not use any loops or create any other methods or class variables (but local variables are allowed). The data is not sorted. Assume 0 <= lo <= hi < data.length and the array values are unique.

```
public static int findMin(double[] data, int lo, int hi) {




}
```

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

A. Divide the pile into two smaller piles of equal size: Recursively sort the two piles then merge them back together.
B. Partition the cards into two smaller piles "Highs" and "Lows" by choosing a threshold value then and recursively sort each of these two piles.
C. Put all of the unsorted cards down. Take any card from the unsorted pile and insert into its correct position of the sorted cards held in your hand.
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: _____

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:

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? _____

how many times has the value '35' moved to a new position? _____

how many times has selection sort called *findMin* (ie found the index of a minimum)? _____

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);
}
```

A. Recursive Quicksort
B. Iterative Quicksort
C. Recursive Selection sort
D. Iterative Selection sort
E. Recursive Insertion sort

Your Answer: _____

7. Recursive Dreaming. Selection Sort – 15 points

Complete the three methods below to correctly implement a recursive **selection** sort so the code matches the behavior described in the comments below. Do not write any loops.

You may assume I've written `findMin(double[] data, int lo, int hi)` and `findMax (double[] data, int lo, int hi)` methods, that may, or may not, be useful to you: The method ***findMin*** returns the *index* of the smallest value in the sub-array `{data[lo],data[lo+1],...,data[hi]}`. Similarly ***findMax*** returns the index of the largest.

```
/** Swaps values at data[i] and data[j] */
public static void swap(double[] data, int i, int j) {

   double temp = data[i];









}
/** 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) {









}
/** 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) {




}
```

END OF CS125 MIDTERM III EXAM
CHECK YOUR WORK FOR MISSING ANSWERS ETC
ENSURE YOUR NETID IS ON EVERY PAGE.

Empty Page

Scratch Paper