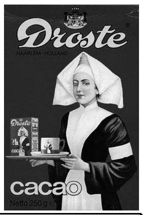
University of Illinois at Urbana-Champaign
Department of Computer Science

Third Examination

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.

• 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.

• 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 == 1) return 2;
3  return 1 + mystery(a/10) + mystery(a/10);
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(100)` creates a { *circle* *base* *runtime* *sandwich* *chain* *tree* } of activations"

d. `mystery(-100)` does not return an integer result. Complete the following sentence to explain why.

"`mystery(-100) is an example of` ________________ `recursion`"

e. Refactor line 3 so that `mystery` uses a chain of activations but returns the same value:

New Line 3:

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

A. `mystery(1000)` is not well defined.
B. `mystery(1000)` will now take the *same* amount of time to calculate the same result.
C. `mystery(1000)` will now take *more* time to calculate the same result.
D. `mystery(1000)` will now take *less* time to calculate the same result.
E. `mystery(1000)` will now return a different result value.

Your answer: ______

g. Identify if the following method is tail-recursive or forward-recursive and explain why.

```
1 public static int bar(int[] data, int i, int a) {
2   int b = a + data[i];
3   if(data[i] >0)
4       return bar(data, i+1, b);
5
6   if(data[i] ==0 || i == data.length-1)
7       return a;
8   return -a;
9 }
```

Circle the correct response: { `Tail` , `Forward` } Recursive and briefly justify your response.

Your explanation: ____

NetID: __________

2. Tracing code – 11 points

Consider the following method 'foo':

```
 public static int foo(int a, int b) {
   if( a < b ) {
        return b;
   }
   int diff = a - b;
   return foo( diff, b-1) + foo( diff, b+1);
}
```

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. The value of *diff* is shared over all activations.
D. Each activation will have its own local, temporary variable *diff*.
E. The variable *diff* is only initialized by the first activation of 'foo'. Your answer: ______

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

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 so the virtual machine will run out of memory.
E. None of the above. ______

c. Create an activation diagram below for the execution of **`foo(20,8)`**. 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(20,8)*.

d. Use your diagram to determine the returned value of `foo(20,8)` ? ______

e. How many times is **`foo`** activated (called), including the first "`foo(20,8)`" ? ______

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).

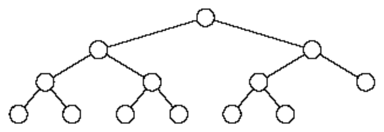
insert takes one integer parameter (the value to insert) and returns a reference to a Link (either the current link or the new link, whichever has the lower value). Insert the value such that the links are sorted by value (lowest values first).

countBig takes no parameters and returns an integer. Return the number of links that have values >99.

sumOdd takes one parameter (an integer accumulator, when I call your method I'll pass zero) and returns an integer – the sum of all the odd values. For full marks implement this method using *tail recursion*. Hint: "x%2" evaluates to one when x is an odd integer. For example, calling *sumOdd* on the first link of 2→3→5 will return the value 8.

```
public class Link {

   private int value;

   private Link next; // null for the last link

    public Link(int v,Link n) {this.value = v; next = n;}

//1. 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.
    }
//2. Write 'countBig' recursive instance method here:












//3. Write 'sumOdd' tail recursive instance method here: (continue overleaf if necessary)







}
```

4. Choose Your Own Adventure Tree Recursion – 15 points



The class 'Question' below models a tree of Yes-No Questions as part of a simple text-based dating-game. A player starts at the topmost question. Depending on their response (yes or no), the game moves onto the next 'yes' or 'no' question (if set). Each question object is referenced just once in this network (i.e. it's a tree). Each question object may have zero or two subsequent possible questions i.e '`yes`' and '`no`' instance variables are either *both* null or *both* non-null.

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 question 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 *countEnd* 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 do not have further *yes* and *no* questions set i.e. *yes* and *no* are both `null`.

b. Write an instance method *find* that takes a string parameter 'key' and returns an integer. Return the number of question texts that contain the given string. Hint: Your method will recursively visit every question in the tree; String's *indexOf* method is useful here.

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 mobile number of the user:

```
public class Pair {
    public String name;
    public long mobile;
}
```

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 lo^th^ and hi^th^ 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* `toMobile` that returns a long value and takes two parameters: '`data`' – a sorted array of `Pair` objects, '`key`' – a string which is the name to find. Return the corresponding mobile number, or `-1`, 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 check that array values are sorted (always increase). Check the values `{data[lo], data[lo+1], ...` up to and including `data[hi]}`. Return -1 if all values are correct otherwise return the *index* of the first incorrect value (the first out-of-order value); thus your code may return value `-1`, or an integer between `lo+1`... `hi` inclusive. Do not read `data[hi+1]` or `data[lo-1]`.
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 where(double[] data, int lo, int hi)
```

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

10	2	9	3	8	20	5	4

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

c. Once **all** 8 array values have been sorted and **all** swaps have completed,
how many times has the first entry of the array been written to? _______

how many times has the last entry of the array been written to? _______

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

how many times has selection sort
called *swap* (include 'useless' swapping to the same position)? _______

d. An incorrect implementation of an iterative sorting algorithm is shown below. Assume *swap* is implemented correctly.

```
for(int i = 0; i < data.length; i++) {
  int m = i;
  for(int y = 0; y < data.length; y++) { //Hint: Mistake on this line
    if(data[y] < data[m])
        m = y;
  }
  swap(data,i,m);
}
```

The initial values of the data array are shown below.

4	3	2	1	8	7	6	5

Calculate the contents of the array after the above iterative method completes.

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(Item[] data, int lo, int hi)` and `findMax (Item [] data, int lo, int hi)` methods, that may, or may not, be useful to you: The method ***`findMin`*** returns the *index* of the smallest item 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(Item [] data, int i, int j) {

   Item 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(Item[] 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 (Item[] 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