University of Illinois at Urbana-Champaign

Spring 2012 Final Examination 3 hours

CS 125 Introduction to Computer Science

Last Name:	First Name:
NetID: _ _ _ _ _ _ _ _ @ illinois.edu (write legibly!)	

PLEASE READ THE FOLLOWING CAREFULLY

• You may not leave with this exam script. You must hand it in and your scratch sheet before you leave. If you leave early, minimize disruption by taking all of your personal items with you.

• Notes and electronic devices (except for medical and simple timepieces) must not be accessible. They are not allowed on your person. Turn off phones and secure them inside your closed bag.

• Do you best but no better: Cheating or appearing to cheat (e.g. attempting to copy from another student, using notes, ignoring proctor instructions) is taken very seriously by the University. Proctors will ensure that the exam is fair for everyone. The small gain is not worth the significant risk to your University status.

• The proctors will not answer technical questions – only clarify English words for non-native speakers. If you believe the exam contains an error or is ambiguous, state your assumptions and answer accordingly.

• Please write your NetID in the top-right corner of each sheet.

• This is a closed book and closed notes exam.

• **You cannot use loops in any problem unless otherwise stated.**

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

• When you write code, you may use a shorthand for System.out and TextIO input and output methods provided it is obvious to the graders which method you are using. For example it is acceptable to use SopIn in place of System.out.println and to use Sopt in place of System.out.print Likewise, you can use T.rlnInt(), in place of TextIO.readlnInt().

• 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	12		
2	15		
3	16		
4	15		
5	12		
6	6		
7	6		
8	8		
Bonus	3		
Total	103		

1. Concepts – 12 points

1. The method `Blah.foo(x)` takes x milliseconds to execute. Determine the number of activations, the output and running time of the main method to the nearest second.

```
public static void main(String[] arg) {
   System.out.println( mystery(1));
}
public static int mystery(int v) {
   if(v > 5) {
        Blah.foo(1000); // always takes one second to execute
        return v;
   }
   return mystery(v*2) + mystery(v*2 + 2);
}
```

Number of activations of *mystery*: _____ , Output: _____, Running time: _____ seconds.

2. The function below claims to compute the factorial function, `f(x) = x!`

```
public static int foo(int n) {
   if (n==1) return 1;
   return n * foo(n-1);
}
```

For example `foo(5)` should return 120 (`5*4*3*2*1`). However a UIUC student claims that it cannot correctly compute `17!` i.e. `foo(17)` does not return `17*16*15*14* ... *1`
Which one of the following is the best explanation?

A) The UIUC student is incorrect; the code will correctly compute all positive factorials.
B) The code enters an infinite loop and does not return a result.
C) The Java program requires a *for*- or *while* loop to calculate a factorial result.
D) The Java program will only return for negative values.
E) Factorial values larger than 0x`7fffffff` suffer from integer overflow.

Your answer (write A,B,C,D or E): _____

3. Which one of the following is true?

A) Class methods have a 'this' instance variable.
B) Subversion, a code repository system, executes Java main methods.
C) A common use of an immutable variable is as a tail-recursive accumulator.
D) An iterative algorithm means it uses forward recursion.
E) None of the above.

Your answer: _____

4. "Inserting an item at the beginning of the linked list is a O(1) operation". Explain what this means (where N is length of the list) AND why using a linked list of sprites might make your viral Facebook game require less CPU time than using an array of sprites implementation.

2. Don't look up – 15 points

Write a complete program to recursively solve the following puzzle. This is the longest programming question in the exam – you may wish to complete the rest of the exam first.

I've just released an evil Pacman monster on a ceiling tile near you.

The ceiling includes a 12x12 grid of square cheap-o-ceiling tiles that can support the weight of one evil Pacman. Each ceiling tile has zero, one or more grains of rice. (Soon, I'll tell you exactly how many I hid up there). Pacman never stays on the same tile: He must move to a neighboring tile that has at least one grain of rice. He will then consume one grain and then move to another square.

Pacman has no valid moves left when no rice remains on the four neighboring tiles. At this point he will dig and fall down into the exam room! Pacman never jumps or moves diagonally. Unlike the MP, Pacman can include previously visited tiles in the same path, provided they have at least one rice grain.

Your program will address two questions:1. What is the maximum number of grains that Pacman can consume? 2. How many Pacman paths end above the TA's position?
Write a *complete* Java program in a class named **`PacmanVsTA`** with two methods: A public **`main`** class method that takes a string array and does not return anything, and a recursive class method named **`solve`**. Do not create any class or instance variables.

In your *main* method:

(i) Use `TextIO.getlnInt` to read, and then store in local integer variables, the TA's x,y position and Pacman's starting x,y position.

(ii) If any of these four values are negative then print an error message ("`No!`") and get four new values from the user, otherwise print "`OK`". Keep iterating steps (i) and (ii) until all four values are non-negative. You can assume all values are less than 12.

(iii) Create a 2D integer array of size 12x12 named '**food**' to hold the rice grain count of each tile and a second array of the same size named '**count**' to count the number of times each ceiling tile is *at the end* of a valid Pacman path.

(iv) Use `TextIO.getlnInt` inside a loop (or nested loop) to read the grain count per tile. Store all 144 values in the 2D food array. (You do not need to validate these values)

(v) Call your recursive *solve* method (described below) to explore all possible Pacman paths. You will need to pass in Pacman's position and your two arrays as parameters. Expect your recursive method to modify the arrays and to return Pacman's maximum rice grain count.

(vi) Print "`MAX=`" followed by the maximum number of rice grains that Pacman can eat.

(vii) If the count value at the TA's position is still zero print "`SAFE`" otherwise print "`LOOKUP!`"

To implement *solve* you must use recursion and no loops! Pass the 2D *food* and *count* arrays as parameters to your recursive method and two integers, Pacman's x-y position. Your recursive method will not print anything and will return the maximum number of rice grains Pacman can consume.

Unlike the *main* method you need to write the *solve* method without specific stepwise instructions. Hint: You will need to modify both the *food* and *count* arrays, determine base-case(s) and make 4 recursive calls (one for each Compass direction).

2. Don't look up (Continued)

Write your program here (continue on back and empty sheet if necessary)

3. Recursion on Linked Lists – 16 points

Complete the `Link` class by writing the methods described below. You may create temporary (local) variables and parameters but you **may not use loops, or create any more methods (other than those specified), class or instance variables.** Assume these methods are called on the topmost link.

```
public class Link {
   public Link next; // null if this is the last link
   public int value;

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

1. Write a **tail** recursive instance method named **`foo`** that takes an integer parameter named "**`acc`**" (initially zero) and returns a boolean: Return true if the last link's value is equal to twice the maximum of all other link values, false otherwise. For example for the linked list, {2→7→14} *`foo`* returns true.

2. Write a **forward** recursive instance method named **`bar`** that takes no parameters and returns an integer. The effect of calling this method is to double each link's value until a link value of zero is reached, or there are no more links. Return the number of non-zero links modified. For example, {5→3→0→4→11} becomes {10→ 6→0→4→11} (4 and 11 are not processed) and *`bar`* returns 2. For example {1→2→3} becomes {2→4→6} and *`bar`* returns 3.

3. Recursion on Linked Lists (Continued)

You may create temporary (local) variables and parameters but you may **not use loops, or create any more methods (other than those specified), class or instance variables.**

3. Write a recursive instance method named **`find`** that takes an integer parameter named "`key`" and returns a reference to a `Link` object. The result of calling this method is a reference to the **last** link that has a value equal to the key value (more than one link may have the same value). If no such link exists this method returns `null`.

4. Write a recursive instance method **`isCopy`** that takes a reference to another Link object (of another linked list) and returns a boolean. Returns true iff they represent the same values in the same order i.e. both this link and the given link have the same value *and* this is recursively true for all remaining links. Return false if the values do not match or if the linked lists are different lengths. Assume this method is called on the topmost link and the topmost link of another list is passed as the parameter. Hint: Watch out for null pointer exceptions - this linked list or the given list may be shorter.

4. Algorithm Analysis – 15 points (3 points each)

For each method determine the order of growth of the *worst case* running time as N increases. Assume N>>1. Write your answer to the right of the method using "Big O" notation. You only need to write the order of growth; you do NOT need to explain how you got your answer.

```
public static double f1(int N) {
    double result = N;

    for(int i = 1; i < 7; i++)
       for(int j = 0; j < 3; j= j + N*N)
           result  + = j;

    for(int p = N; p >0; p = p / 2)
       result += N * N - p;

    return result;
}
```

```
 public static int f2(int[] arr, int lo, int hi) {
  // N = initial value of (hi-lo+1) of 1st activation of f2
  if(lo == hi) return lo;
  if(arr[hi] >= arr[lo])
      return f2(arr, lo+1, hi);
   return f2(arr, lo, hi-1);
}
```

```
public static int f3(int N) {
   int i = N/5;
   while(i >0)
       i = i - 1;
   return N * N;
}
```

```
public static void f4(int N) {
   int a = 0;
   while(a < N * N * N) {
      a = a + N * N;
   }
}
```

```
public static void f5(int[] a, int[] b) {
  // N is a.length.
  // Assume a.length = b.length
  int i=0;
  while(i<a.length) {
    if(a[i] < b[i]) {
       b[i] = a[i];
    } else {
      i++;
    }
  } // while
}
```

5. Sorting Algorithms – 12 points

This question analyzes the 3 sorting algorithms and their implementation without any advanced optimizations: selection sort, mergesort, and quicksort. Assume the implementations given in CS125.

1. Complete the following table that describes the running time of a sorting algorithm studied in CS125. In the 4 empty spaces write the algorithm's name or the running time in Big O notation.

Sorting Algorithm	**Running Time Best Case**	**Running Time Worst Case**
i)	$O(N^2)$	$O(N^2)$
ii)	iii)	O(N lg N)
iv)	O(N lg N)	$O(N^2)$

2. I use mergesort to sort 32 floating point values, initially in random order, stored in a double array. How many total calls (activations) to mergesort will there be? Assume the base case sorts a single value.

Your answer: _____

3. An integer array, shown below, is sorted using mergesort. The first merge will sort 19 and 4. Show the contents of the integer array after the third merge has completed:

Initial values:	19	4	40	26	21	17	3	39
Values after third merge:								

4. Which one of the following best describes the standard quicksort's pivot choice?

a) Choosing the minimum value ensures a fast O(1) quicksort running time.
b) Choosing the maximum value ensures a fast O(NlgN) quicksort running time.
c) Choosing a median of 3 values ensures quadratic running time is impossible.
d) Choosing a median of 3 values reduces the chance of quadratic running time.
e) Choosing the median of all values ensures the best quicksort running time.

Your answer: _____

5. I use CS125's selection sort to sort 100 *reverse-sorted* integers {100,99,98...1}. How many 'useless' swaps will our selection sort perform? A useless swap is where the array contents are unchanged because a value is swapped with itself.

Your answer: _____

6. I test my selection sort Java code on a smart phone. I can sort 400,000 randomly ordered integers in ten milliseconds. If I now have 800,000 integers initially in reverse order, approximately how long will the same code take to execute?

a) 0 – 20 ms
b) 21 – 99 ms
c) 100 – 999 ms
d) 1.0 – 49.99 seconds
e) 50 seconds or greater

Your answer: _____

6. Quicksort, Mergesort multiple choice – 6 points

For each of the following statements decide if it applies to Mergesort, Quicksort or both or neither.

i) Uses *findMinimum* [Mergesort only] [Quicksort only] [Both] [Neither]

ii) Requires O(N) space. [Mergesort only] [Quicksort only] [Both] [Neither]

iii) Worst case has $O(N^2)$ **levels** (depth) of recursion [Mergesort only] [Quicksort only] [Both] [Neither]

iv) Best case has O(N) **levels** (depth) of recursion [Mergesort only] [Quicksort only] [Both] [Neither]

For v) and iv): I write a standard mergesort and two quicksort algorithms "A" and "B" with different pivot choices: Quicksort Algorithm A uses **the first value** of the sub-array. Algorithm B uses the **last** value of the sub-array. For each of the following carefully decide which, if any, quicksort algorithm will usually complete faster than mergesort. Circle the one best response for each scenario.

v) Data is reverse sorted (decreasing values): [A only] [B only] [Both A&B] [Neither]

iv) Data is unsorted (randomly shuffled values): [A only] [B only] [Both A&B] [Neither]

7. Conditionals and instance variables – 6 points

Write a public instance method named **`choose`** with no parameters that returns one of 3 Strings, "`Roger`", "`Pete`", or "`John`" (you may shorten these to single letters R,P,J respectively). Your code must have exactly 3 return statements, one for each rule. Use the following rules to decide which String to return:

1. I will call Roger if I have 200 minutes or greater and I want to sing or play guitar. If rule 1 does not apply, then consider rule 2 and 3:
2. I will call Pete if I want to sing or do not need food.
3. If neither rule 1 nor rule 2 applies, I will call `John`.

```
public class Who {
    private char purpose;//One of d=drum,g=guitar,s=sing
    private int time; // Available time (minutes)
    private boolean food; // true if food is required
//Complete the 'choose' method here...
```

} // class

8. PictureList – 8 Points

To answer this question correctly review the given code carefully. Complete the class "PictureList" started below, according to the following specification. **You may use loops in this question.** You can assume all entries from 0 to size-1 are valid (non-null) Picture references.

- A public instance method ***toDeepReverse*** that returns a reference to a new *PictureList* object. Return a deep copy of this list (Assume *Picture* implements a copy constructor). Return the list in reversed order. Hint: data.length != size
- A public instance method ***find*** that takes a String 'name' returns a new PictureList. Return a list of all pictures that have the given name. Assume Picture objects implement a 'getName' method that returns a string. Use a shallow copy (do not create any new Picture objects).

```
public class PictureList {

  private Picture [] data; // Pictures are stored in 0... size-1
  private int size; // initially =0, =data.length once array is full

  public int getSize() { return size;}
  public Picture get(int i) {return data[i];}

  public void add(Picture r) { if(r!=null && size<data.length) data[size++]=r;}
  public void deleteLast() { size--;}

  public PictureList(int n) { data = new Picture[n]; }
```

// Write your toDeepReverse method here

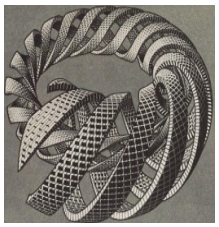
// Write your find method here

```
} //class (Continue overleaf if necessary)
```

Tricky Bonus Challenge – 3 points

Check your answers to other questions before you attempt to get these tricky 3 bonus points. You may use loops in this question and you can create any (instance and class) variables and methods that you need.

The infamous fugitive – *Lorenzo Angravo* – wanted for Grand Neural Corruption – has modified your brain. Your challenge is to repair the damage. You can get a Java version of your brain by calling the class method `Brain.getBrain()` which returns a reference to one `Neuron` object. You can recursively visit all other neurons through the connection network (synapses). All references are non-null. However, there may be loops (cycles) e.g. Neuron X connects to Y, which connects to Z which connects back to X.



Lorenzo has: 1) Set the value of all of your neurons to `42` – an impossible value for a neuron. 2) Created some additional secret neurons – these all have a value of `99` 3) Created new synapses that are now part of your the brain's network to connect your neurons to and from *Lorenzo's* secret neurons. Your challenge is to reset the value of every neuron to zero and disconnect the secret neuron's from your brain i.e. there are no more synapse connections between your original neurons and the neurons created by *Lorenzo.* Original connections between your original neurons (and their strength) must be unchanged. Hint: Watch out for infinite recursion and null pointer exceptions if you set array entries to `null`.

```
public class Synapse { // A single connection to another neuron
      public double strength;
      public Neuron target; // the receiver of this connection.
}

public class Neuron { // A single brain cell
      public double value; //output value of this neuron
      public Synapse[] to; // Who gets the output of this neuron

      public static void main(String[] args) {
```

} // Continue overleaf or on the overflow page if necessary

END OF THE EXAM – GO BACK AND CHECK FOR ERRORS & OMISSIONS

OVERFLOW – Use this page if you need more space

Scratch paper. You may tear this off but need to turn it in with your exam.