University of California, Berkeley – College of Engineering
Department of Electrical Engineering and Computer Sciences
Fall 2011 Instructor: Dan Garcia 2011-12-13

CS10 PAPER FINAL

Last Name	**ANSWERS**
First Name	
Student ID Number	
`cs10-` *Login First Letter*	**a b c d e f g h i j k l m**
`cs10-` *Login Last Letter*	**a b c d e f g h i j k l m** **n o p q r s t u v w x y z**
The name of your LAB TA (please circle)	**Aijia Glenn Luke Navin Rabbit Samir**
Name of the person to your Left	
Name of the person to your Right	
All my work is my own. I had no prior knowledge of the exam contents nor will I share the contents with others in CS10 who have not taken it yet. ***(please sign)***	

Instructions

- Don't Panic!
- This booklet contains 6 pages including this cover page. Put all answers on these pages; don't hand in any stray pieces of paper.
- Please turn off all pagers, cell phones and beepers. Remove all hats and headphones.
- You have 170 minutes to complete this exam. The final is closed book, no computers, no PDAs, no cell phones, no calculators, but you are allowed three double-sided sets of notes. There may be partial credit for incomplete answers; write as much of the solution as you can. When we provide a blank, please fit your answer within the space provided.

Question	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	Online	Total
Points	2	2	2	2	2	2	2	2	2	2	2	10	10	10	10	15	80
Score																	

Short-answer Questions (2 pts each, we drop lowest score)

Question 1: Briefly name one practical application for IBM's Watson technology. (aside from mobile phones)

Question-answering seemed to be a good fit. In the last decade, question-answering systems have become increasingly important for firms dealing with mountains of documents. Legal firms, for example, need to quickly sift through case law to find a useful precedent or citation; help-desk workers often have to negotiate enormous databases of product information to find an answer for an agitated customer on the line.

Question 2: Argue both perspectives of the game theory debate using only one sentence each.

A *weak* solve is better than a *strong* solve because…	A *strong* solve is better than a *weak* solve because…
If all you care about is the theoretic value, it's much faster	**You know the value of EVERY position, so your AI can be perfect, and/or you can kibbitz/comment on human-human play.**

Question 3: What are the ugly, difficult details of programing a hundred-thousand-node compute cluster that the elegant MapReduce abstraction hides from the user?

Handling machine failures, load balancing, dispatching and collecting

Question 4: "Cloud computing is great, but it'll never work with *Big Data*, since the cost and/or time of transporting large datasets to/from the cloud is too great." Circle ***True*** or ***False*** and explain in one sentence.

False – you can either fedex hard disks or use their "pay as you go" fast network

Question 5: How is Pandora™ able to satisfy the demand of its 100+ Million users with so few media servers?

Three ideas: (1) only 1/10th are active daily (2) Data locality i.e., a small number of songs are played a large number of times, and (3) the data requirements are actually quite small since it's streaming (128 Kib/s) not storage.

Question 6: What's the hardest thing Twitter has to do (from an engineering point of view)?

Make it all work (i.e., tweets from all senders to all receivers) in seconds.

Question 7: Prof Yelick said computing is addressing two global challenges. One was "*our changing world*: understanding climate change, alternative energy sources, mitigation techniques, etc." *What was the other?*

Health and medicine – understanding the human body, development of treatments, and disease prevention

Question 8: You need to find an efficient polynomial-time algorithm (exact) solution to a tough problem, but you're not succeeding. However, you do prove your problem is NP-complete. How does that help?

You have proved that your problem is just as hard as many, many other problems which have no poly-time solution yet AND if anyone solves any of those, you get yours solved too!

Question 9: How could quantum computers help with *the subset problem*? That's 2^n combinations to test!
Quantum computers can solve the problem in $2^{n/2}$ time, a huge difference!

It marked the first time courts said people were responsible where their bits went (in this case, out of state to a location with a different obscenity standard)

Question 10: In 1994, a couple in Milipitas (in CA near here!) were convicted because their bulletin board, dubbed "The Nastiest Place on Earth", contained obscene material. Why was this an important ruling?

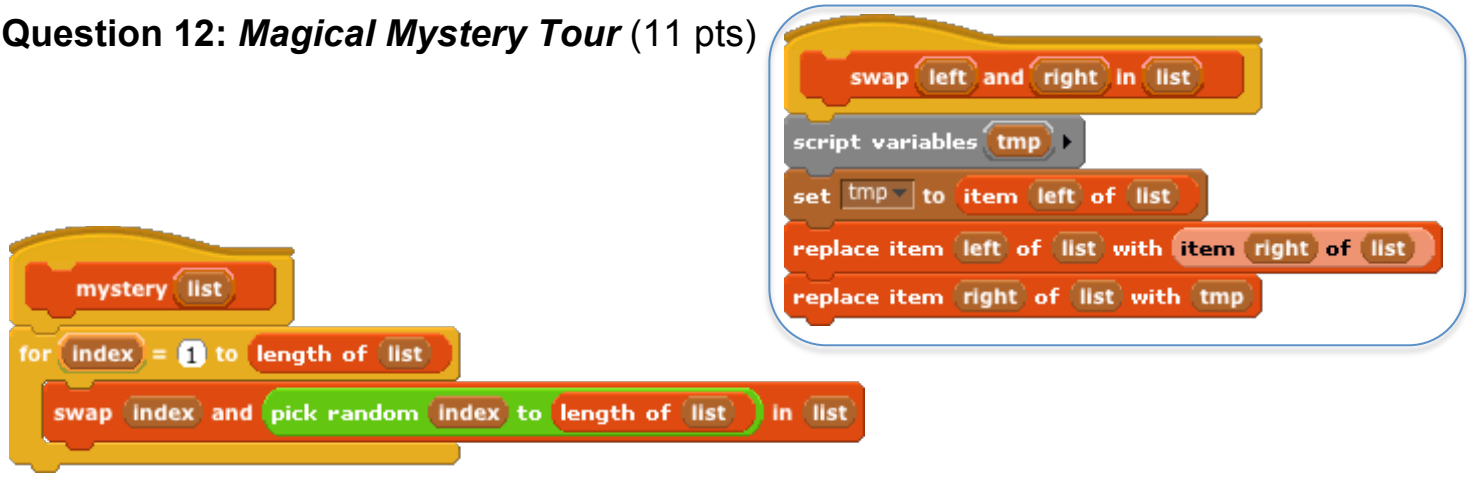
Login: `cs10-____`

Question 11: *Eating my Halloween candy well beyond Thanksgiving…* (6 pts)

Your mom asks you the difference between an iterative and recursive solution to a problem. You decide to explain it to her by showing how you would program a robot to eat a bag of M&Ms iteratively and recursively. Assume the robot knows how to "**`eat one M&M`**" and check if "**`Bag is Empty`**"

Eat M&Ms Iteratively:	**Eat M&Ms Recursively:**
Repeat until <Bag is Empty>: **Eat one M&M**	**If not <Bag is Empty>:** **Eat one M&M** **Eat M&Ms Recursively**

Question 12: *Magical Mystery Tour* (11 pts)



a) Below each script, write ALL the possible values of `list` after each script is run.

(A B)	**(A B) (B A) (A A) (B B)**

Quadratic

b) Assuming **`swap`** and **`pick-random`** are constant-time operations, **`length-of(list)`** is a linear-time operation, what is the running time of **`mystery`**? _____

"unsorts" the list – shuffles the elements randomly

c) What does **`mystery`** do? _____

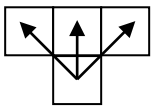
d) Assuming you didn't know how **`mystery`** was written, but were just given the spec from your answer to (c), how would you test **`mystery`** really, really, really thoroughly? (this is called *black-box* testing)

Make a list of n items (say n=5), then run the code 100*n! times and see if all n! permutations are evenly distributed

Question 13: *Strawberry Fields Forever (revisited)...* (10 pts)

This was the question from the midterm:

Strawberry plants are funny. Every year they send "runners" to their left and right neighbors, which take seed and become an entirely new strawberry plant the next year. We'd like to model this process and count how many strawberry plants we'll have in our garden (that we've divided into columns, like the number line) starting from a single strawberry plant in column 0 in year 1, the top row. All other numbers in the top row are 0 (no other strawberry plants). The number in every subsequent row is the sum of the three numbers directly above it, to the above left and to the above right as shown below. We've filled in the table for years 1 through 5:



		…	-4	C -3	O -2	L -1	U 0	M 1	N 2	3	4	…
	1	…	0	0	0	0	1	0	0	0	0	…
Y	2	…	0	0	0	1	1	1	0	0	0	…
E	3	…	0	0	1	2	3	2	1	0	0	…
A	4	…	0	1	3	6	7	6	3	1	0	…
R	5	…	1	4	10	16	19	16	10	4	1	…
	⋮	⋰	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋱

You are to write a function **`Plants`** that takes two integer arguments, a *column* and a (positive integer) *year*, and returns the number of plants that will be in that column on that year.

Since you're now an experienced computer scientist, you know that we try to *abstract*, and generalize our solutions. In the problem statement notice we hard-coded the ways the runners move. What if every year they send their runners to the left only? What if strawberry plants die every year and only the runners survive? To generalize this, we define two lists **`COLUMNS`** and **`YEARS`** that encode how the contributing cells in the table above sum to a specific `(column, year)` value. For example, in the midterm problem, we would have defined **`COLUMNS`** as **`(-1 0 1)`** and **`YEARS`** as **`(-1 -1 -1)`**, which is why the recursive case was the sum:

```
plants(column-1,year-1) + plants(column+0,year-1) + plants(column+1,year-1)
```

a) Edit the recursive case in the **`Plants`** function to handle this more generalized approach. We've filled in the base case with the answers from the midterm solutions. (**`COLUMNS`** and **`YEARS`** are always equal size.)

```
Plants(column, year)
    if ( (column = 0) and (year = 1) )
        report ( 1 )
    if ( year = 1 )
        report ( 0 )
      combine-with[+]items-of(map[Plants(column+(),year+()]over(COLUMNS)(YEARS))
    report ( _____________________________________________________________

                _____________________________________________________________ )
```

b) If **`YEARS`** is only a list of (at least one) **`-1`**s, what is the running time of **Plants**?
(it could be a function of `column`, `year`, `COLUMNS` or `YEARS`)

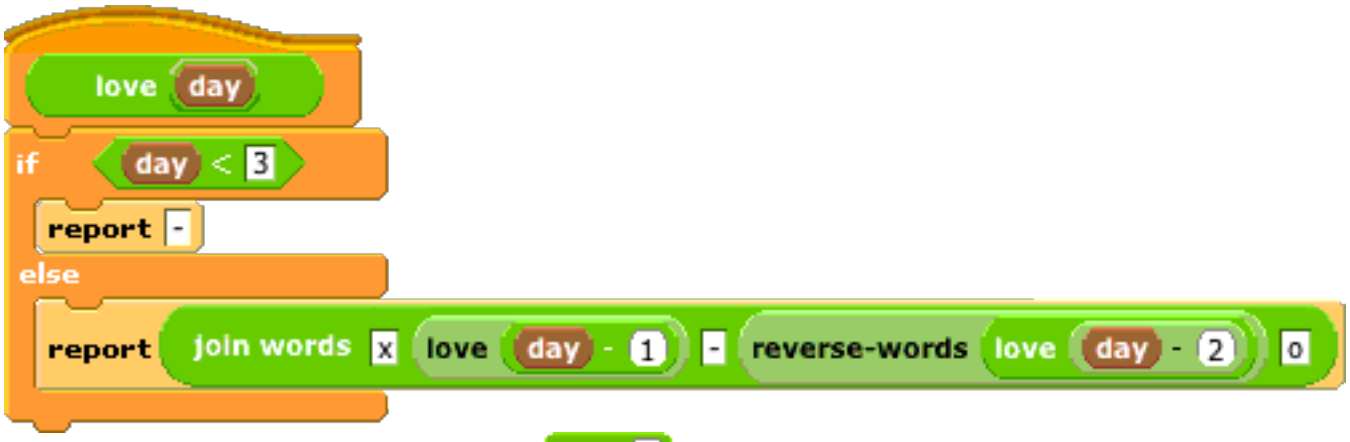
Zeros everywhere, but the central column is the Fibonacci series
If len(COLUMNS)=1, linear. OW exponential.

c) If we replaced both "**`year = 1`**" checks with "**`year < 3`**" in the base cases above, and then defined **`COLUMNS`** as **`(0 0)`** and **`YEARS`** as **`(-1 -2)`**, describe the table that results *(hint: you've seen it before)*.

Login: `cs10-____`

Question 14: *Give me some love! XOXO...* (10 pts)

You decide to write **`love`**, a function to chart how affectionate you are (i.e., what you do) with your sweetie over the course of a given `day` (day 1 is your first day together, day 2 is your second, etc.). It returns a (possibly long) sentence whose elements are only: hugs (`o`), kisses (`x`), and just hanging out (`-`). We provide a helper block **`reverse-words`**, that does what you'd imagine: `reverse-words("CS10 is fun")` ➔ `"fun is CS10"`



a) What will you do on day 3? I.e., what will love 3 return? If it is an error, say what the error is. If it is an infinite loop, write "it never returns".

x - - - o

b) What will you do on day 4? I.e., what will love 4 return? If it is an error, say what the error is. If it is an infinite loop, write "it never returns".

x x - - - o - - o

c) Now let's do some analysis of your long-term relationship. What are the *first three and last three things you do on day 9999? That is,* what are the first three and last three letters of love 9999? Fill in the blanks.

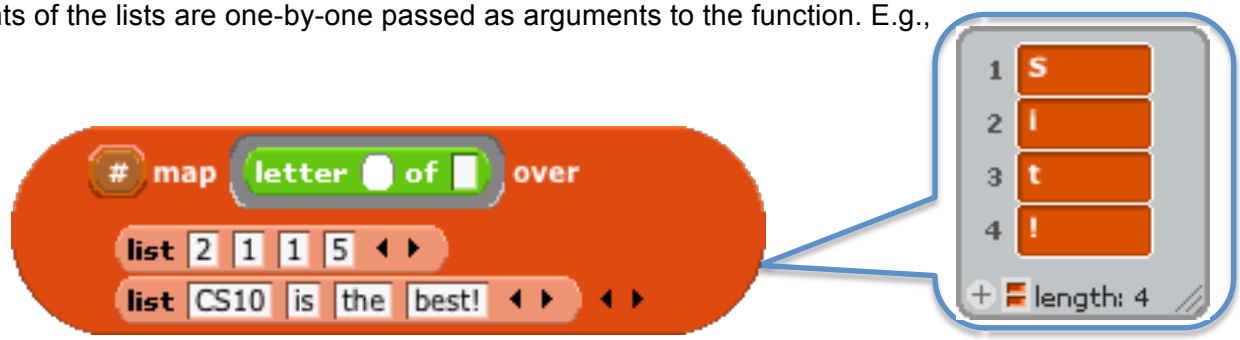
x _____	x _____	x _____	. . .	x _____	x _____	o _____

d) `love` can return a long and seemingly random sequence of `x`s, `o`s & `-`s. For each of the following activities, circle either POSSIBLE or IMPOSSIBLE if it's ever possible to do these things someday. The first one is already done for you.

- (POSSIBLE) IMPOSSIBLE : `"- - -"` (Hang out *three* times in a row)
- POSSIBLE IMPOSSIBLE : `"- - - -"` (Hang out *four* times in a row)
- POSSIBLE IMPOSSIBLE : `"o x"` (Hug immediately followed by a kiss)
- POSSIBLE IMPOSSIBLE : `"o o"` (Hug *twice* in a row)

Question 15: *A sorted Quest-ion* (10 pts)

Recall that our `map` function usually takes a `function` (of one argument) and one `list`, and applies the `function` to every element of the list, returning a `list` of the same size. It can also take *many* lists, and in this case the `function` must take the number of arguments equal to the number of lists, all the same size. The elements of the lists are one-by-one passed as arguments to the function. E.g.,



a) Write the following new block (a "cousin" of **map**) that takes a `function` (of two arguments) and a `list` (of at least two items) and applies the function to *every set of two neighbors*, returning a list one element smaller than the input. You may not use explicit recursion or iteration, and you may find the list manipulation helper functions (listed below) helpful. Here is an example call:

map(function)over-neighbors-in(list)

```
                map( function )over( all-but-last-of(list) all-but-first-of(list) )
    report( ____________________________________________________________________ )
```

b) Now, using this function you've written in (a), write **Sorted?(list)** that returns 

when the input list is sorted in ascending order (i.e., every element is *smaller* than the ones after it, like the list `(3 6 7)`. Again, you may not use explicit recursion or iteration.

Sorted?(list)

```
                Combine-with[ and ]items-of(map[ < ]over-neighbors-in( list ))
    report( ____________________________________________________________________ )
```

List manipulation helper functions

Name	Description	Example
`item(num)of(list)`	Returns item at list index num	**`item(3)of( (cs10 is fun) )`** ➔ `fun`
`all-but-first-of(list)`	Returns a new list with the last element removed. Doesn't change the original.	**`all-but-first-of( (cs10 is fun) )`** ➔ `(is fun)`
`all-but-last-of(list)`	Returns a new list with the first element removed. Doesn't change the original.	**`all-but-last-of( (cs10 is fun) )`** ➔ `(cs10 is)`

Extra Credit Question

Learning computing concepts may have opened many doors for you in your future work. Although you may not ever use BYOB again, the concepts you have learned may become useful to you. Some examples include:

- Provide you with enough basic programming skills that you could easily pick up another language
- Appreciating that no new technology is black or white, and the societal / ethical / economic consequences are often unintended
- More conservatism w.r.t. social networks, your own privacy, e-voting, and the role of “big brother”, which is anyone with access to your information: from a government to a university to your local cloud provider.
- Provide you with basic analysis tools to estimate how an algorithm (or plan or idea, in the real world) would scale as you grow the size of the input, to determine when a problem is computationally tractable, and to recognize that some problems are not decidable.
- Empowering you to see a situation (say, how people are lining up to take a flight) and write a simulation that would model it (say, boarding from the outside in vs back-to-front vs random) to generate some data for your analysis.

Aside from the examples given, or enhancing the examples given, please describe a situation in which you think the computing concepts you have learned will help you in the future.