Solutions to
CS1112 Final Exam
Spring 2009

1. Boolean, characters and strings (20 points)

(a) Fill in the appropriate boolean condition to keep prompting the user until a number that is a multiple of 3 or 5 is entered or until the user has entered 20 numbers, whichever occurs first.

```
k = 1;  n = input('Enter a number: ');

while ________________________

     n = input('Enter a number: ');  k = k + 1;
end
```

Solution:

```
CORRECT:        mod(n,3)~=0  &&  mod(n,5)~=0  &&  k<20
CORRECT:        mod(n,3)     &&  mod(n,5)     &&  k<20
WRONG:          mod(n,3)==1  &&  mod(n,5)==1  &&  k<20

Note in above expressions k must be strictly less than 20 (k~=20 is ok)

CORRECT:     ~( mod(n,3)==0  ||  mod(n,5)==0  ||  k==20 )
```

(b) What will be displayed by executing the following fragment? Write the word "error" if the fragment results in an error.

```
x = 'ab' == 'AB';
disp(x)
```

Output:

0 0

(c) What will be displayed by executing the following fragment? Write the word "error" if the fragment results in an error.

```
s = 'abc';
y = 1;
while y
    s = s(1:length(s)-1);
    y = y && ~isempty(s);
    disp(y)
end
```

Output:

1
1
0

(d) Fill in the appropriate boolean condition below to determine whether variable `ch` stores a lowercase letter. Assume variable `ch` is initialized and stores a single character.

```
if _________ ch >= 'a'  &&  ch <= 'z' ___________________

    disp('Variable ch stores a lowercase letter.')
else
    disp('Variable ch does not store a lowercase letter.')
end
```

2. Sort and search (20 points)

(a) Circle T for true and F for false for each problem below. Consider the sorting algorithms we have studied in class: insertion sort, bubble sort, and merge sort. Let n be the length of the vector to be sorted.

1. **TRUE** For large n, merge sort is more efficient (run-time) than insertion sort.
2. **TRUE** For large n, merge sort is more efficient (run-time) than bubble sort.
3. **TRUE** On average, insertion sort is more efficient (run-time) than bubble sort.
4. **TRUE** Merge sort can be implemented as either a recursive or non-recursive function.
5. **TRUE** Insertion sort can be implemented such that a vector is sorted in-place.
6. **TRUE** Bubble sort can be implemented such that a vector is sorted in-place.

(b) Both linear search and binary search are iterative algorithms. Given a vector of length 30, what is the maximum number of iterations for doing a search? Circle the correct answer for each search method below.

Linear search (vector is not sorted):	around 5	or	<u>around 30</u>	or	around 30^2
Binary search (vector is sorted):	<u>around 5</u>	or	around 30	or	around 30^2

(c) Fill in the appropriate boolean condition.

```
function f = indexOfX(v,x)
% f is the index of the first occurrence of value x in vector v.
% f is 0 if x is not found.
k= 1;

while  _______________________________________________________________

    k= k+ 1;
end
if  k>length(v)
    f= 0; % signal for x not found
else
    f= k;
end
```

Solution:

```
k<=length(v) && v(k)~=x     % Must check index before target
```

3. Building a vector (20 points)

Implement the following function so that it performs as specified:

```
function  z = FiveSmallest(x,y)
% x is an 1-by-n array with x(1)<x(2)<...<x(n) and n>=5
% y is an 1-by-m array with y(1)<y(2)<...<y(m) and m>=5
% z is a 1-by-5 array with the following properties:
%
%    (i)  z(1) < z(2) < ... < z(5)
%   (ii)  z  consists of the five smallest distinct values in x and y
```

Thus, if

x :

10	15	17	20	30	40

and

y :

15	19	20	33	40	42	50

then

z :

10	15	17	19	20

Do not use built-in functions `sort`, `min`, or `unique`.
Do not use vectorized code. For example, `a=[b c]` is vectorized code and is not allowed.

Solution:

```
ix = 1;
iy = 1;

for k = 1:5

   if  x(ix)<y(iy)

      z(k) = x(ix)
      ix = ix+1;

   elseif  y(iy)<x(ix)

      z(k) = y(iy)
      iy = iy+1;

   else
      z(k) = x(ix);
      ix = ix+1;
      iy = iy+1;
   end
end
```

4. Matrix and vector (25 points)

Implement the following function so that it performs as specified:

```
function pos = after2clicks(G,p)
% pos is the vector of all possible pages that may be reached after exactly two
% clicks starting from page p.  p is a positive integer no bigger than the
% size of square matrix G.  G is the 0-1 connectivity matrix where
%   G(i,j)=1  means there is a link to page i from page j
%   G(i,j)=0  means there is not a link to page i from page j
% Assume that each page has at least one outlink.
```

Do not use built-in function `find`. Assume that function `removeDuplicates` is available for your use—do not implement this function. Depending on your algorithm you may or may not need `removeDuplicates`. See the bottom of the page for the function header.

```
function w = removeDuplicates(v)
% w is v with duplicates removed.
% Example:  if v is [1 3 7 3 8 1 2] then w is [1 3 7 8 2]
```

Solutions:

```
[n, m]= size(G);

% 1st click
f= [];  % pages after 1st click
for i= 1:n
    if G(i,p)==1
        f= [f i];
    end
end

% 2nd click
pos= [];  % pages after 2nd click
for k= 1:length(f)
    for i= 1:n
        if G(i,f(k))==1
            pos= [pos i];
        end
    end
end

pos= removeDuplicates(pos);
```

```
[n, m]= size(G);

pos= [];  % pages after 2nd click
for i= 1:n
    if G(i,p)==1  % i is page after 1st click
        for r= 1:n
            if G(r,i)==1
                pos= [pos r];  % pages after 2nd click
            end
        end

    end
end
pos= removeDuplicates(pos);
```

5. Structure array and cell array (25 points)

By making effective use of the built-in function `sort`, complete the following function so that it performs as specified:

```
function C = MakeList(S)
% S is a length n structure array within which each structure has these fields:
%   title   The title of a song (a string)
%   y       The sound signals of the song (a vector of real numbers)
%   rate    The sampling rate (samples per second) for playing the song (a
%           positive integer)
% C is a length n cell array with the property that the i-th cell contains
% the title of the song that has the i-th shortest playing time.
```

Solution:

```
n = length(S);
playTime = zeros(n,1);

for k = 1:n

   playTime(k) = (length(S(k).y)-1) / S(k).rate;

   %              length(S(k).y) / S(k).rate     is OK
end
   % Note: In general it's not possible to "vectorize"
   %       field access in a struct array

[z,idx] = sort(playTime);

C = cell(n,1);

for k = 1:n

   j = idx(k);
   C{k} = S(j).title;

end
```

6. Efficiency, 2-d array (20 points)

Consider the fragment below. Suppose n is a positive integer, A is an initialized n-by-n array, and f is a function that has a scalar parameter and returns a scalar.

```
x = linspace(0,1,n);
s = 0;
for i=1:n
   for j=1:n
       s = s + A(i,j)*f(x(i))*f(x(j));
   end
end
```

(a) Assume that the dominant computing cost is the evaluation of f and that each evaluation costs one dollar. Thus, the cost of executing the above fragment is about $2n^2$ dollars. Give an efficient implementation assuming that A(i, j) is zero if $i > j$.

```
x = linspace(0,1,n);

% Pre-compute f values (and use them later)
for k= 1:n
    fvalue(k)= f(x(k));
end

% Update s only for non-zero values in matrix A (upper triangle and diagonal)

for i= 1:n         %  OR   for j=1:n
   for j= i:n      %         for i=1:j

       s = s + A(i,j)*fvalue(i)*fvalue(j);
   end
end
```

(b) What is the *approximate* computing cost of your implementation? $\boxed{\$\ \mathbf{n}}$

7. Recursion (20 points)

(a) The $\prod$ symbol represents "product of." For example,

$$\prod_{x=3}^{6} x(x-2) = (3 \cdot 1) \cdot (4 \cdot 2) \cdot (5 \cdot 3) \cdot (6 \cdot 4) = 8640$$

Write a **recursive** function `myProd` that takes in integer n and returns y, which is evaluated as

$$y = \prod_{x=3}^{n} x(x-2).$$

Assume that $n \geq 3$.

```
function y = myProd(n)
% y = 3*1 * 4*2 * 5*3 * 6*4 * ... * n*(n-2)
% n is an integer, n>=3

if n==3

    y = 3;
else

    y = n*(n-2) * myProd(n-1);
end
```

(b) Using function `myProd` to compute $\prod_{x=3}^{5} x(x-2)$, how many calls to `myProd` will be made? _____

Solution: <u>**3**</u> **if base case is as above** (i.e., includes $n = 3$)

Alternate solution: <u>**4**</u> **if base case is** $n < 3$ (i.e., excludes $n = 3$):

```
if  n<3
     y = 1;
else
     y = n*(n-2) * myProd(n-1);
end
```

(c) Circle T for true or F for false: A function using a single `for`-loop, instead of recursion, can be implemented to compute $\prod_{x=3}^{n} x(x-2)$. ***TRUE***