Solutions to
CS1112 Final Exam
Fall 2008

For Review Only.

Do Not Remove From Distribution Room.

Do Not Copy.

1. Boolean (15 points)

Fill in the appropriate boolean condition for each problem below.

(a)

```
a= rand(1);  b= rand(1);  c= rand(1);


if  ____________________________________________________________

    disp('The 3 lengths, a, b, and c, can form a triangle')
else
    disp('The 3 lengths, a, b, and c, cannot form a triangle')
end
```

Solution:

```
    ~( a+b<=c || a+c<=b || b+c<=a )

or  a+b>c && a+c>b && b+c>a
```

(b)

```
function f = indexOfX(v,x)
% f is the index of the first occurrence of value x in vector v.
% f is 0 if x 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
```

2. Short answers (15 points)

(a) Write a fragment that sets up a 100-by-1 array `x` with the property that each component is a randomly selected *even integer* between and including 1000 and 2000.

```
r = rand(100,1);

x = 2 * floor( r*501 + 500 );
```

(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: around 5 or <u>around 30</u> or around 30^2

Binary search: <u>around 5</u> or around 30 or around 30^2

(c) 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. **T** Insertion sort is an in-place algorithm.
2. **T** Bubble sort is an in-place algorithm.
3. **T** For large n, merge sort is more efficient (run-time) than insertion sort.
4. **T** For large n, merge sort is more efficient (run-time) than bubble sort.

3. Matrix and vectors (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 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.

```
[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);

%%%%%%% Version 2 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
[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);
```

4. Structure array, 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 where each structure has these fields
%      name   The name of a state (a string)
%      pop    The population of the state (a positive integer)
%      area   The area of the state in square miles (a positive real)
% C is a length n cell array with the property that the i-th cell is
% a string that names the state having the i-th lowest population density.


    n = length(S);
    density = zeros(n,1);

    for k=1:n

       density(k) = S(k).pop/S(k).area;
    end

    [z,idx] = sort(density);

    C = cell(n,1);

    for k=1:n

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

    end
```

5. Vector (25 points)

Complete the fragment below to print a histogram for vector `v` which holds nonnegative integer values. For instance, if `v` is [5 1 7 0 4 6] then your code should print the following output in the *Command Window*:

```
      X
      X     X
  X   X     X
  X   X   X X
  X   X   X X
  X   X   X X
  X X X   X X
```

You can assume that `v` has nonzero length and holds nonnegative integers. Note that the goal is to produce this output on the *Command Window*. You should not be using either the `plot` or `bar` functions.

```
  % Print a histogram in the Command Window as described above.
  % Assume v is given.  v has nonzero length and holds nonnegative integers.

%%%% Version 1: First fill out a char matix

nr= max(v);
nc= length(v);
blanks= max(v) - v;  % # blanks in each column of histogram

for c= 1: nc
   % Column c of histogram
   for r= 1:blanks(c)
       histo(r,c)= ' ';
   end
   for r= blanks(c)+1 : nr
       histo(r,c)= 'X';
   end
end

histo  % display the histogram


%%%% Version 2: Print on the fly

nr= max(v);
nc= length(v);
blanks= max(v) - v;  % # blanks in each column of histogram

for r= 1: nr
    for c= 1: nc
        if ( r <= blanks(c) )
            fprintf(' ')
        else
            fprintf('X')
        end
    end
    fprintf('\n')
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 real-valued function of a single real variable.

```
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
```

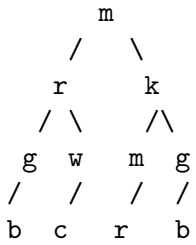
(a) Assume that the dominant computing cost is the evaluation of `f` and that each evaluation costs one dollar. Thus, the cost of executing the above fragment is $2n^2$ dollars. Give an efficient implementation assuming that `A`(i, j) is zero if $i > j$.

```
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? **$ n**

7. Recursion (25 points)

Each node in a binary tree can have at most two offsprings. Our binary tree is described completely by a string in which each character is the name of the node. The first element of the string is the root (top) of the tree, and then the first half after that is the left half of the tree and the second half is the right half. Each half is organized the same way: the first element is the current node and the remaining elements are split in two halves, and so on. Draw the binary tree represented by the string **'mrgbwckmrgb'**. Draw the actual tree diagram in the space below—do not write code. For example, the string 'mrg' corresponds to the binary tree on the right:



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

$$\prod_{x=2}^{5} x(x-1) = (2 \cdot 1) \cdot (3 \cdot 2) \cdot (4 \cdot 3) \cdot (5 \cdot 4) = 2880$$

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

$$y = \prod_{x=2}^{n} x(x-1)$$

n and y are scalars and assume that $n \geq 2$. Hint: look for the pattern! What is y for $n = 2$? What is y for $n = 3$? Can you define $y(n = 3)$ in terms of $y(n = 2)$?

```
function y = myProd(n)
% Assume n>=2

if n==2        % OR, less efficient soln:
               %    if n<2
   y= 2;       %       y= 1;

else

   y= n*(n-1) * myProd(n-1);

end
```