Contents

E7 Lab 4 Solutions

Spring 2016

```
format compact
format short
clear all
clc
close all
```

Question 1

```
type myBigONotation
```

```
%% Problem 4 - Big O Notation

function [answers] = myBigONotation()
% A common notation for complexity is called Big-O notation. Big-O notation establishes the re
lationship
% in the growth of the number of basic operations with respect to the size of the input as the
 input
% size becomes very large. As n gets large, the highest power dominates; therefore, only the h
ighest power
% term is included in Big-O notation. Additionally, coefficients are not required to character
ize growth,
% and so coefficients are also dropped. In the previous example, we counted 4n2 +n +1 basic op
erations
% to complete the function. In Big-O notation we would say that the function is O(n2) (pronoun
ced ?O of
% n-squared?). We say that any algorithm with complexity O(nc) where c is some constant with r
espect
% to n is polynomial time.

answers={'E', 'D', 'B'};

%Example 1 => O(n^3)) =>     E
%Example 2 => O(log(n)) =>   D
%Example 3 => O(n) =>        B

end
```

Published Test Case

```
answers = myBigONotation()
```

```
answers =
    'E'    'D'    'B'
```

Question 2.1

```
type myBraille2Double
```

```
function [result] = myBraille2Double (braille)
% This is a function that converts braille input to a number (class
% "double")

% workflow:
% 1. enter the braille representations for 0-9 manually
% 2. take input "braille" and see how many digits we need to output
% 3. loop through each sub-matrice in "braille" to obtain the digit
% 4. and add the digit to "result" by multiplying 10^(i)


% list numbers in braille representation
one=[1 0;0 0;0 0];
two=[1 0;1 0;0 0];
three=[1 1;0 0;0 0];
four=[1 1;0 1;0 0];
five=[1 0;0 1;0 0];
six=[1 1;1 0;0 0];
seven=[1 1;1 1;0 0];
eight=[1 0;1 1;0 0];
nine=[0 1;1 0;0 0];
zero=[0 1;1 1;0 0];
% first detect the number of digits
n_digits=size(braille, 2)/2;
%initialize output
result=0;
for i=1:n_digits
    % see the individual braille number to convert
    sub_braille=braille(:,((i-1)*2+1):(2*i));

    if isequal(sub_braille,one)==1;
        digit=1;
    elseif isequal(sub_braille,two)==1;
        digit=2;
        elseif isequal(sub_braille,three)==1;
        digit=3;
        elseif isequal(sub_braille,four)==1;
        digit=4;
        elseif isequal(sub_braille,five)==1;
        digit=5;
        elseif isequal(sub_braille,six)==1;
```

```
        digit=6;
        elseif isequal(sub_braille,seven)==1;
        digit=7;
        elseif isequal(sub_braille,eight)==1;
        digit=8;
        elseif isequal(sub_braille,nine)==1;
        digit=9;
        elseif isequal(sub_braille,zero)==1;
        digit=0;
    end
    result=result+digit*(10^(n_digits-i));
end

end
```

Published Test Case

```
[result] = myBraille2Double ([ 1 , 0 , 1 , 1 ; 1 , 0 , 0 , 1 ; 0 , 0 , 0 , 0 ])
```

```
result =
    24
```

Published Test Case

```
[result] = myBraille2Double ([ 1 , 0 , 1 , 1 , 1 , 0 ; 1 , 0 , 0 , 1 , 0 , 0 ; 0 , 0 , 0 , 0 ,
 0 , 0 ])
```

```
result =
   241
```

Additional Test Case

```
[result] = myBraille2Double ([[1 0; 0 0; 0 0] [1 0; 0 1; 0 0] [1 1; 0 0; 0 0] [1 1;1 0; 0 0]])
```

```
result =
        1536
```

Question 2.2

```
type myBraille2ASCII
```

```
function [ ASCII] = myBraille2ASCII(braille)

% This function converts number in braille to number in ASCII
```

```
% workflow:
% 1: use the function "myBraille2Double" to obtain the double number first
% 2: use the number of digits to create an array for the ASCII ouput
% 3: go through each digit in the double number and put the conversion into
% the ASCII array.

number=myBraille2Double(braille);
numberstr=num2str(number);

n_digits=numel(numberstr);
ASCII=zeros(1,n_digits);

for i=1:n_digits
    digit=numberstr(i);
    ASCII(i)=digit+0;
end


end
```

Additional Test Case

```
result = myBraille2ASCII([[1 0; 0 0; 0 0] [1 0; 0 1; 0 0] [1 1; 0 0; 0 0] [1 1;1 0; 0 0]])
```

```
result =
    49    53    51    54
```

Question 3

```
type myBinary2Num
```

```
function [result] = myBinary2Num(binary, representation)

%binary is a character string of length 8 made of only zeros and ones, and representation
%is the name of the representation used for binary, given as a character string that can take
%one of the three following values: 'unsigned', 'sign-magnitude' and 'twos complement'.
%Your function should return in its output argument result the number (in base 10, Matlab
%class double)

% guidance:
% write individual "if" loops for each representation

numberstr=num2str(binary);
result=0;
if strcmp(representation,'unsigned');
    for i=1:numel(numberstr)
        if numberstr(i)=='1';
          result=result+2^(7-i+1);
        end
```

```
    end
elseif strcmp(representation,'sign-magnitude');
    if numberstr(1)=='1';
        sign=-1;
    elseif numberstr(1)=='0';
        sign=1;
    end

    for i=2:numel(numberstr)
        if numberstr(i)=='1'
            result=result+2^(6-i+2);
        end
    end

    result=result*sign;
elseif strcmp(representation,'twos complement')

    if numberstr(1)=='1';
        result=result-(2^7);
    elseif numberstr(1)=='0';
        result=result+0;
    end

    for i=2:numel(numberstr)
        if numberstr(i)=='1'
            result=result+2^(6-i+2);
        end
    end
else
    'Please enter a valid string for representation'
end
end
```

Published Test Case

```
result = myBinary2Num('11001000' , 'unsigned')
```

```
result =
   200
```

Published Test Case

```
result = myBinary2Num('11001000' , 'sign-magnitude')
```

```
result =
   -72
```

Published Test Case

```
result = myBinary2Num('11001000' , 'twos complement')
```

```
result =
   -56
```

Additional Test Case

```
result = myBinary2Num('01001110','unsigned')
```

```
result =
    78
```

Additional Test Case

```
result = myBinary2Num('11001110','sign-magnitude')
```

```
result =
   -78
```

Additional Test Case

```
result = myBinary2Num('11001110','twos complement')
```

```
result =
   -50
```

Question 4.1

```
type myCompareFloats
```

```
function [ exact , approx ] = myCompareFloats (x , y , tolerance)
% where x, y, and tolerance are scalars of class double, and exact and approx are scalars
%of class logical. exact should be true (logical 1) if and only if x == y evaluates to true
%in Matlab. approx should be true if and only if the difference between x and y (in absolute
%value) is less than or equal to tolerance.

% note: remember to put the absolute value for "approx"

exact=(x==y);
approx=(abs(x-y)<=tolerance);

end
```

Published Test Case

```
[exact,approx] = myCompareFloats(2+3 , 5 , 0)
```

```
exact =
     1
approx =
     1
```

Published Test Case

```
[exact,approx] = myCompareFloats(0 , 0.001 , 1e-2)
```

```
exact =
     0
approx =
     1
```

Published Test Case

```
[exact,approx] = myCompareFloats(0 , 0.001 , 1e-9)
```

```
exact =
     0
approx =
     0
```

Additional Test Case

```
[exact,approx] = myCompareFloats(1000,1000.001, 0.01)
```

```
exact =
     0
approx =
     1
```

Additional Test Case

```
[exact,approx] = myCompareFloats(1000,1000.1, 0.01)
```

```
exact =
     0
approx =
```

```
0
```

Question 4.2

```
type mySingle2Decimal
```

```
function [result] = mySingle2Decimal (binary)




% workflow:
% define the values s,d,e,f first
% then do "if" loops to calculate the output.

numberstr=num2str(strtrim(binary));
value_s=numberstr(1);
value_d=127;
value_e=myBinary2Num(numberstr(2:9),'unsigned');

% calculate value_f below
value_f=0;
for i=10:32;
    if numberstr(i)=='1'
    value_f=value_f+2^(9-i);
    end
end

if value_e~=0 & value_e~=255
    result=((-1)^(value_s))*(2^(value_e-value_d))*(1+value_f);
elseif value_e==0 & value_f~=0
    result=((-1)^(value_s))*(2^(1-value_d))*value_f;
elseif value_e==0 & value_f==0
    result=0;
elseif value_e==255 & value_f==0
    result=((-1)^(value_s))*Inf;
elseif value_e==255 & value_f~=0
    result=0/0;
end

end
```

Published Test Case

```
result = mySingle2Decimal ('00111111111100000000000000000000')
```

```
result =
    1.8750
```

Published Test Case

```
result = mySingle2Decimal ('10111111000000000000000000000000')
```

```
result =
   -0.5000
```

Published Test Case

```
result = mySingle2Decimal ('00100000100000000000000000000001')
```

```
result =
   2.1684e-19
```

Published Test Case

```
result = mySingle2Decimal ('11111111100000000000000000000000')
```

```
result =
  -Inf
```

Published Test Case

```
result = mySingle2Decimal ('11111111100000000000000000000001')
```

```
result =
   NaN
```

Additional Test Case

```
result = mySingle2Decimal ('10111111010000000000000000000000')
```

```
result =
   -0.7500
```

Additional Test Case

```
result = mySingle2Decimal ('00100000100000000000111000000001')
```

```
result =
```

```
    2.1693e-19
```

Question 5.1

```
type myCompareElements
```

```
function [result] = myCompareElements (element1, element2)
%Function:   myCompareElements goal is provide an comparison between two
%            character strings.
%Input:      Element1 and Element2 are the names as character strings of two elements.
%            Note: First letter is uppercase, rest are lower
%Output:     This function returns one of the following numbers as result:
%            0 if the same
%            1 if element 1 comes before 2
%            -1 if element 1 comes after 2

% Force strings to be equal in length
x=char({element1;element2});

% Subtract one from the other
d = x(1,:) - x(2,:);

% Remove zero entries
d(~d) = [];
if isempty(d)
    result = 0;
else
    result = d(1);
end

% Convert to -1,0,1 format
if result>0
    result=-1;
    elseif result<0
        result=1;
end
end
```

Published Test Case

```
element1 = 'Hydrogen' ; element2 = 'Carbon' ;
result = myCompareElements (element1 , element2)
result = myCompareElements (element2 , element1)
result = myCompareElements (element1 , element1)
```

```
result =
    -1
result =
     1
result =
     0
```

Additional Test Case

```
element1='Uranium'; element2='Boron';
result = myCompareElements (element1 , element2)
result = myCompareElements (element2 , element1)
result = myCompareElements (element1 , element1)
```

```
result =
    -1
result =
     1
result =
     0
```

Additional Test Case

```
element1='Plutonium'; element2='Hydrogen';
result = myCompareElements (element1 , element2)
result = myCompareElements (element2 , element1)
result = myCompareElements (element1 , element1)
```

```
result =
    -1
result =
     1
result =
     0
```

Question 5.2

```
type mySortElements
```

```
function [sorted]=mySortElements(elements)
%Function Input: (elements) Array of elements in cell format
%Fucntion Output: (sorted)  Array of sorted elements


sorted=cell(1,numel(elements)); %Create empty cell

for i=1:numel(elements) %Use function to determine every comparison
    for j=1:numel(elements)
        temp(i,j)=myCompareElements(elements{i},elements{j});
    end
end
sums=sum(temp,2)';

for i=1:numel(elements) %Find min, put in min location
    [m, k] = min(sums);
```

```
        sorted(i)=elements(k);
        sums(k)=100000; %Value such that will never be min
    end

  sorted = flip(sorted); %Correct so A->Z
  end
```

Published Test Case

```
elements={'Hydrogen','Calcium'};
sorted=mySortElements(elements)
```

```
sorted =
    'Calcium'    'Hydrogen'
```

Published Test Case

```
elements={'Hydrogen' , 'Carbon' , 'Magnesium' , 'Calcium' , 'Carbon'};
sorted=mySortElements(elements)
```

```
sorted =
    'Calcium'    'Carbon'    'Carbon'    'Hydrogen'    'Magnesium'
```

Additional Test Case

```
elements={'Helium','Xenon','Radon','Neon','Helium','Krypton'};
sorted=mySortElements(elements)
```

```
sorted =
    'Helium'    'Helium'    'Krypton'    'Neon'    'Radon'    'Xenon'
```

Additional Test Case

```
elements={'Dubnium','Bohrium','Copernicium','Meitnerium','Roentgenium'};
sorted=mySortElements(elements)
```

```
sorted =
    'Bohrium'    'Copernicium'    'Dubnium'    'Meitnerium'    'Roentgenium'
```

Additional Test Case

```
elements={'Argon','Argon'};
sorted=mySortElements(elements)
```

```
sorted =
    'Argon'    'Argon'
```

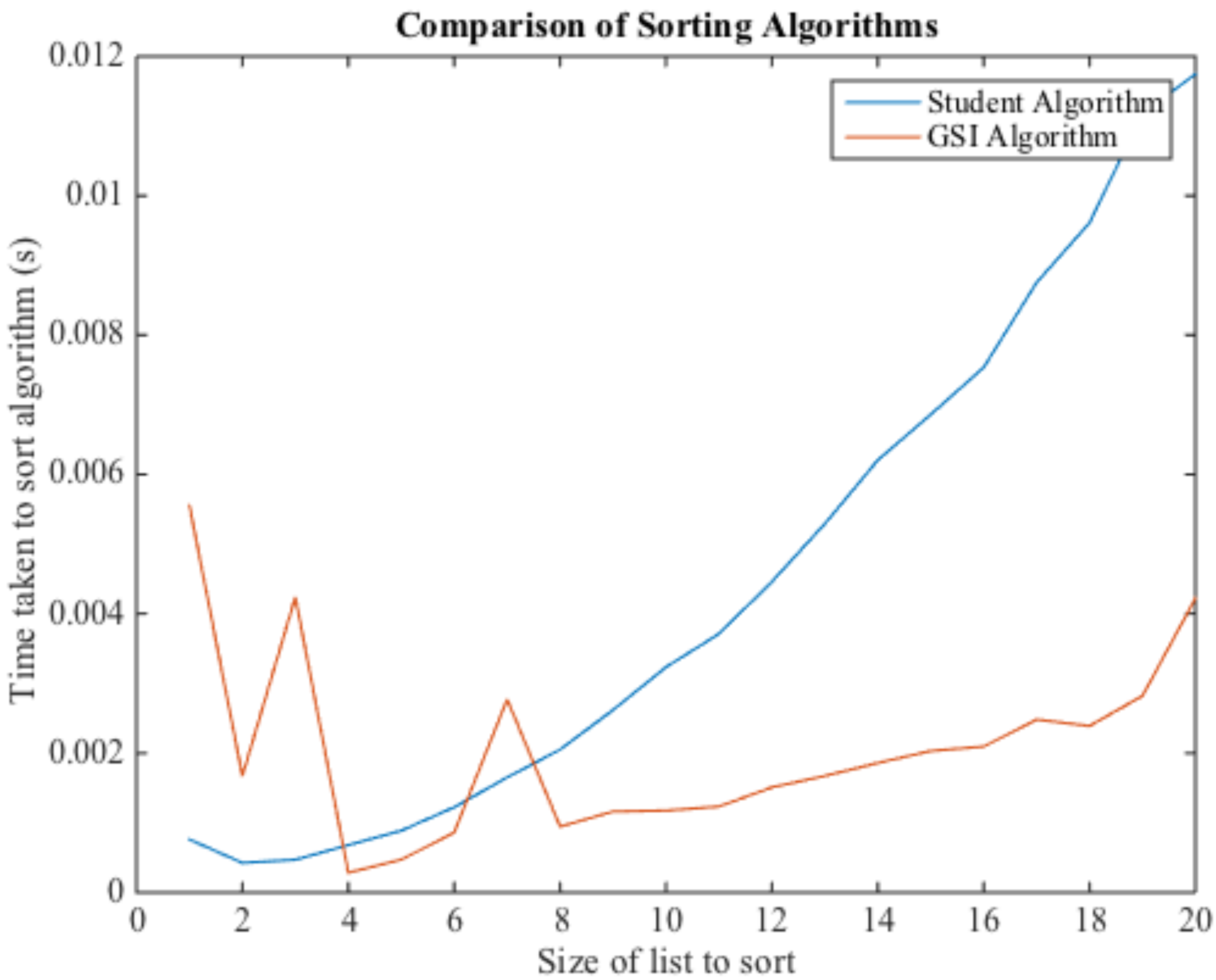
Question 5.2

```
type myCompareSorting
```

```
function [] = myCompareSorting (n)

% Write a function that compares the efficiency of the mySortElements function you wrote in pa
rt 2 of
% this problem to the efficiency of the sorting function GSISortElements that is available as
a
% .p file on bCourses. Here, we define efficiency as the time taken by the sorting function to
 sort
% a given list of elements. Note that there are many other criteria that should be considered
% for a more exhaustive characterization of the efficiency of an algorithm (for example RAM
% usage), but we ignore these in this assignment. Use the following header for your function

% The function?s input argument n is an integer (Matlab class ?double?) strictly greater
% than zero.

%Initialize time vectors and Random element list
time_student = zeros(1,n);
time_GSI = zeros(1,n);
NewElements=GSIRandomElements(n);

%Execute for loop to determine toc times for each respective element for n
%cases each
for i=1:n

    tic;
    myAns=mySortElements(NewElements(1:i)); %Student Sorting
    time_student(i)=toc;

    tic;
    GSIAns=GSISortElements(NewElements(1:i)); %GSI Sorting
    time_GSI(i)=toc;
end

% Plot Figure: Size of list vs Time per algorithm

figure('color','white'); clf;
plot(1:n,time_student,1:n,time_GSI)
set(gca,'fontname','times','fontsize',14)
hold all; box on;
xlabel('Size of list to sort')
ylabel('Time taken to sort algorithm (s)');
legend('Student Algorithm','GSI Algorithm');
title('Comparison of Sorting Algorithms');
end
```

Published Test Case

```
myCompareSorting(20)
```



Published with MATLAB® R2014b