CS1132 Spring 2009 Assignment 1

Adhere to the Code of Academic Integrity. You may discuss background issues and general solution strategies with others and seek help from course staff, but the homework you submit must be the work of just you. When submitting your assignment, be careful to follow the instructions summarized in Section 4 of this document.

1 Playing Blackjack

In the game of Blackjack each player plays his hand independently against the dealer. At the beginning of each round, the player places a bet in the "betting box" and receives an initial hand of two cards. The objective of the game is to get a higher hand (card total) than the dealer without going over 21, which is called "bust." The spot cards count as 2 to 9; the 10 and the face cards (jack, queen, and king) count as ten; an ace can be either 1 or 11. The player goes first and plays his hand by taking additional cards if he desires. If he busts, he loses. Then the dealer plays her hand. If the dealer busts, she loses to all remaining players. If neither busts, the higher hand total wins.

1.1 Simplified Game

We will have a simplified version of blackjack in which there is only one player and one dealer. The spot cards count as 2 to 10, face cards count as 10, and ace counts only as 11. The game goes as this:

- Player places a bet, which is less then the amount of money he or she has
- Dealer gets two cards face up from a randomized (shuffled) deck
- Player gets two cards face up from the same deck
- Player makes a decision to draw a card, hold (not get any more cards), or quit. If player decides to quit he loses the money he or she has bet.
- If player holds or draws a card then dealer makes a decision to hold or draw a card.
  Dealer always plays the following fixed strategy: if dealer's cards on hand sum up to less then 17 then dealer must draw a card, otherwise dealer must hold.
- The two previous steps are repeated until the player quits, the player or dealer "busts," or both player and dealer hold. In case of a "bust" the person whose cards sum to greater than 21 loses. In case when both player and dealer hold the person with the highest sum on his/her hand wins.

If the player wins he or she adds as much money to their pot as they have bet. (That is to say that the stakes in this game are 1:1). If player draws he/she is not loosing any money(not winning any money either). If a player who loses all his/her money is forcibly removed from the casino! (We do not care about the dealer's winnings or loss.)

1.2 Simulating Casino

Your task is to implement a function `blackjackCasino(n)` that simulates one player entering a casino to play our blackjack game:

```
function i = blackjackCasino(n)
% Simulate one player at the blackjack game as specified.
% n: amount of money a player has upon entering casino
% i: amount of money a player has when leaving casino
```

Your code must simulate a "casino experience": Upon entering a casino a player is asked if he or she wants to play blackjack and if yes he/she plays a game. If no the player is asked if they wish to quit or stay in the casino. If the player desires to stay the question about playing blackjack is raised again. If the player quits in the middle of the game of blackjack they also leave casino (the function returns). The player can play as many blackjack games he or she wishes as long as he/she doesn't run out of money. If the player runs out of money he/she leaves the casino. Below is the output from a sample run of the program. Your program should produce the same kind of output.

```
>> money_left=blackjackCasino(5)
Welcome to our Blackjack Casion!
Do you want to play blackjack?(y/n): y
Please place your bet: 2
Players hand is: 4D  9D
sum to 13
Dealers hand is: A_S 7S
sum to 18
Do you want to draw a card, hold or quit?(y/h/q): y
Players hand is: 4D  9D  K_D
sum to 23
Dealers hand is: A_S 7S
sum to 18
Players hand is: 4D  9D  K_D
sum to 23
Dealers hand is: A_S 7S
sum to 18
You bust and loose.
You have $3
Do you want to play blackjack?(y/n): n
Do you want to leave casino?(y/n): y
You have $3
```

1.3 Program Organization And Note on `input`

You will submit one file, `blackjackCasino.m`, but this question is complicated enough that you should organize your code into subfunctions. Do some planning before you actually start writing code!

Note on built-in function `input`: The typical use of `input` expects the user to enter a number. If you want the user to be able to enter a string without using single quotes as shown in the sample output above, you need to use a second argument, `'s'`, when calling the function. E.g.,

```
answer = input('Do you want to leave?(y/n)', 's');
```

2 Sorting Fun!

Sorting a collection of objects has become indispensable for today's computer programs. From arranging a list of products by price on Amazon.com to ordering flight itineraries by the duration of the flight, sorting is everywhere. In this exercise we examine three ways in which sorting can be performed. (For the purpose of this exercise let's assume we want to arrange an array in ascending order, or more specifically in non-descending order). Also we want to experimentally see the performance of these algorithms depending on whether the array is sorted or almost sorted.

Do not use built-in function `sort`.

2.1 Bubble Sort

Bubble sort is a simple sorting algorithm: it works by repeatedly going through the array to be sorted, comparing two adjacent items at a time and swapping them if they are in the wrong order. Specifically, in a pass through the array items 1 and 2 are compared and possibly swapped, then items 2 and 3 are compared and possibly swapped, and so forth until the end of the array is reached. The pass through the array is repeated until no swaps occur, which means the array is sorted. The algorithm gets its name from the way smaller elements “bubble” to the beginning of the array. You must implement this algorithm in the function:

```
function sortedArray = bubbleSort(A)
% Sort vector A in ascending order using Bubble Sort algorithm
% A: the vector of numbers to be sorted, the length of A is > 0
% sortedArray: the sorted array
```

The positions of the elements in bubble sort has a big effect on the number of passes that will be needed to sort the array. Large elements at the beginning of the array do not raise problems, as they are quickly swapped.

2.2 Merge Sort

Merge sort is more complicated. The simplest implementation of merge sort uses recursion, but we’ll use a different implementation that does not use recursion. The main idea in merge sort is that given two sorted arrays, we can merge them relatively easily to form one (larger) sorted array. Start by writing a function `merge`:

```
function t = merge(v,w)
% Merge sorted vectors v and w to form vector t, which is also sorted.
% v,w: sorted numeric vectors not necessarily of the same length
% t: sorted vector whose length is length(v)+length(w)
```

Your code should take advantage of the fact that `v` and `w` are already sorted.

We begin by considering a vector x with length n where n is a power of 2. Here’s the general idea:

1. Set the subvector length m to be 1. (Why? It’s the shortest vector that is “sorted.”)
2. Divide vector x into subvectors of length m.
3. Merge every two adjacent subvectors. I.e., merge subvectors 1 and 2, 3 and 4, 5 and 6, . . . , so that the merged subvector (each of length $2m$) is sorted.
4. Double the value of m. (That’s the length of each merged—sorted—subvector.)
5. Repeat steps 2 to 4 until $< ? >$. You should figure this out.

Let’s look at an example. The vector below is of length 8.

3	4	5	2	7	8	9	3

With a subvector length of 1, we have 8 subvectors. Merging subvectors l and 2, 3 and 4, 5 and 6, and 7 and 8 gives us the following:

3	4	2	5	7	8	3	9

Now we have four sorted subvectors of length 2. Then we merge subvectors 1 and 2, 3 and 4, giving us two sorted subvectors of length 4:

2	3	4	5	3	7	8	9

Merging these two subvectors gives us the sorted vector of length 8, completing the sorting problem:

2	3	3	4	5	7	8	9

You will generalize this non-recursive algorithm for any positive n value. The key is that the merge step should allow for subvectors of different lengths. Consider a vector of length 11:

4	3	1	9	8	7	2	0	6	5	1

When the subvector length m is 1, there are 11 subvectors and five merges. There is not a sixth merge because there isn't a "right-side" subvector to carry out a merge. Therefore the 11th subvector is left alone.

$$\begin{array}{|c|c||c|c||c|c||c|c||c|c||c|}\hline 3 & 4 & 1 & 9 & 7 & 8 & 0 & 2 & 5 & 6 & 1 \\ \hline \end{array}$$

Now we have five subvectors of length $m = 2$ and one leftover component, which we'll call the deficient subvector. We will do three merges but note that the last merge has a deficient (short) right-side subvector. The result is

$$\begin{array}{|c|c|c|c||c|c|c|c||c|c|c|}\hline 1 & 3 & 4 & 9 & 0 & 2 & 7 & 8 & 1 & 5 & 6 \\ \hline \end{array}$$

Now we have two subvectors of length $m = 4$ and one deficient subvector. We can only do one merge since the last subvector doesn't have a right-side partner. So the last subvector is left alone:

$$\begin{array}{|c|c|c|c|c|c|c|c||c|c|c|}\hline 0 & 1 & 2 & 3 & 4 & 7 & 8 & 9 & 1 & 5 & 6 \\ \hline \end{array}$$

Now $m = 8$. We have one subvector of length 8 and one deficient subvector. Merge them and we're done.

You must implement this non-recursive algorithm in the function `mergeSort`. `mergeSort` must call `merge`, specified above.

```
  function sortedArray = mergeSort(A)
  % Sort vector A using the non-recursive merge sort algorithm as discussed
  % A: the vector of numbers to be sorted, the length of A is > 0
  % sortedArray: the sorted vector
```

2.3 Selection Sort

Selection sort is a relatively simple sorting algorithm. Start by selecting the smallest value in the vector and putting it at the front. Put the selected smallest value at the front a swapping the selected value with the value originally at the front. Then select the smallest value from the remaining vector (the unsorted segment) and swap it into the second position. Now the unsorted segment starts from position 3. Again select the smallest value from the unsorted segment and swap it into position 3, and so forth. You must implement this algorithm in the function

```
  function sortedArray = selectionSort(A)
  % Sort vector A using the selection sort algorithm as discussed
  % A:  the vector of numbers to be sorted, the length of A is > 0
  % sortedArray: the sorted array
```

2.4 Implementing Swap function

To help us with our investigation we will need to implement function swap, which given and array and a number `n` performs `n` swaps in array `A`. That is to say it chooses at random two positions within the array swap their content, and this is done `n` times.

```
function randomizedArray = swap(A, n)
% A: the vector of numbers
% n:  number of random swaps % randomizedArray:  array A with n numbers swapped
```

2.5 Comparing Runtimes

To get an idea of the relative speed of these sorting methods, you will run each of them and compare their run times. To do so, you will write a single script that generates sorted arrays, swaps some number of its entries and runs each of the three methods on the generated arrays.
You should try picking an array size to be 10^4 (you probably will want to test on smaller array sizes) and then starting with the sorted array introduce different number of swaps (1 10 10^2 10^3 etc.) this way for each

number of swaps generate 10 swapped arrays. Make sure that each generated array is sorted by each of the three methods! Compute the average running time for each array size and for each method. Display the results in a different plot for each array size, where the x-axis represents the number of swaps (1, 10, 100, ...) and the y-axis represents the time. Connect the points belonging to the same method with a line and use colors to differentiate between methods.
To record the time it takes to run a method use `tic` and `toc`. Together they provide the functionality of a stopwatch: `tic` starts the timer and `toc` stops it and returns the elapsed time. To exemplify their use, here's a code snippet measuring the time needed to run `bubbleSort` on some array `A`:

```
tic;
bubbleSort(A);
elapsedTime = toc;
```

Save your script as `runtime.m`.

3 Self-check list

The following is a list of the minimum necessary criteria that your assignment must meet in order to be considered satisfactory. Failure to satisfy any of these conditions will result in an immediate request to resubmit your assignment. Save yourself and the graders time and effort by going over it before submitting your assignment for the first time.
Note that, although all of these are necessary, meeting all of them might still not be sufficient to consider your submission satisfactory. We cannot list everything that could be possibly wrong with any particular assignment!

- Δ Comment your code! If any of your functions is not properly commented, regarding function purpose and input/output arguments, you will be asked to resubmit.
- Δ Suppress all unnecessary output by placing semicolons (;) appropriately. At the same time, make sure that all output that your program intentionally produces is formatted in a user-friendly way.
- Δ Make sure your functions names are exactly the ones we have specified, including case.
- Δ Check that the number and order of input and output arguments for each of the functions matches exactly the specifications we have given. In particular, the functions required in this project are:
  1. Function `blackjacCasino(n)`, which takes an integer value `n` and returns one output value.
  2. Function `swap(A, n)`, which takes two and array `A` and a number of entries to be swapped `n`. And returns an array with the swapped enteries.
  3. Function `merge(v, w)`, which takes two sorted vectors and returns a sorted vector with the length, which is a sum of vector length' `v` and `w`.
  4. Function `bubbleSort(A)`, which takes an array `A` and returns the sorted array.
  5. Function `mergeSort(A)`, which takes an array `A` and returns the sorted array.
  6. Function `selectionSort(A)`, which takes an array `A` and returns the sorted array.
  7. Script `runtime` calls function `swap` the three sorting methods above.
- Δ Test each one of your functions independently, whenever possible, or write short scripts to test them.
- Δ Check that your scripts do not crash (i.e. end unexpectedly with an error message) or run into infinite loops. Check this by running each script several times in a row. Before each test run, you should type the commands `clear all; close all;` to delete all variables in the workspace and close all figure windows.

4 Submission instructions

1. Upload files `blackjackCasino.m`, `swap.m`, `merge.m`, `bubbleSort.m`, `mergeSort.m`, `selectionSort.m` and `runtime.m` to CMS in the submission area corresponding to Assignment 1 in CMS.
2. Please don't make another submission until you have received and read the grader's comments.
3. Wait for the grader's comments and be patient.
4. Read the grader's comments carefully and think for a while.
5. If you are asked to resubmit, fix all the problems and go back to Step 1! Otherwise you are done with this assignment. Well done!