CENG 105 Intro to CS
Erdogan Dogdu

Assignment 3

Due: Dec 16, 2017

Subject: Algorithms

1) (Binary Search) Run the following **guess.py** program on your computer a few times:

```
# This is a guess the number game.
import random
guessesTaken = 0
print('Hello! What is your name?')
myName = input()
number = random.randint(1, 10000)
print('Well, ' + myName + ', I am thinking of a number
between 1 and 10000.')
while guessesTaken < 20:
   print('Take a guess.')
   guess = input()
   guess = int(guess)

   guessesTaken = guessesTaken + 1
   if guess < number:
         print('Your guess is too low.')
   if guess > number:
         print('Your guess is too high.')
   if guess == number:
         break

if guess == number:
   guessesTaken = str(guessesTaken)
   print('Good job, ' + myName + '! You guessed my number in
' + guessesTaken + ' guesses!')
if guess != number:
   number = str(number)
   print('Nope. The number I was thinking of was ' + number)
```

What is your best guess? How many times it took for you to guess the number? Send the output of the program with your answers.

(Your trial)

What is the optimal number of guesses for this case? How did you calculate? (Hint: Calculate for the binary search in the range from 1-10000)

For binary search in the range 1-10000, everytime you guess, you halve the range. First guess is let's say 5000. If the program says "too low", then your next guess is in the range 5001-10000; else if it says "too high", your next guess is in the range 1-

4999. Therefore, in every new guess, you halve your range. Therefore, the optimal number of guesses is $\log_2 n = \log_2 10000 \approx \log_2 2^{13.288} \approx 14$

What is the best case (the number of times it takes to guess the number right)?

If you are lucky, it is 1 guess (the first guess is right)

2) (ch5.29) What problems do you expect to arise if the following program is implemented in python? (Hint: The problem of round-off errors associated with floating-point arithmetic.). Correct the program and make sure that it halts.

```
cnt = 0.1
repeat:
  print(cnt)
  cnt = cnt + 0.1
  until (cnt == 1)
```

Python does not have "repeat" loop, so we use "while" loop:

```
cnt = 0.1
while(True):
     print(cnt)
     cnt = cnt + 0.1
     if (cnt == 1): break
```

But this program does not stop, goes to an infinite loop, since cnt is never exactly equal to 1. We can change the program to the following so that it works, **int** function gives the integer part of cnt variable:

```
cnt = 0.1
while(True):
     print(cnt)
     cnt = cnt + 0.1
     if (int(cnt) == 1): break
```

This program gives the following output:

```
0.1
0.2
0.30000000000000004
0.4
0.5
0.6
0.7
0.7999999999999999
0.8999999999999999
0.9999999999999999
```

So, it is not working as intended. We want to increase cnt by 0.1 in every step. So, we correct the program one more time:

```
cnt = 0.1
while(True):
     print(cnt)
     cnt = round(cnt + 0.1,1)
     if (int(cnt) == 1): break
```

round(x,1) function call rounds cnt to the first digit after the decimal point. Then, the output is as follows:

```
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
```

3) (*ch5.2, question 3*) The Euclidean algorithm finds the greatest common divisor of two positive integers X and Y by the following process: As long as the value of neither X nor Y is zero, assign the larger the remainder of dividing the larger by the smaller. The greatest common divisor, if it exists, will be the remaining non-zero value.
Express this algorithm in our pseudocode. And write a pyhon code to implement.

```
def gcd(a, b):
    """Calculate the Greatest Common Divisor of a and b.

    Unless b==0, the result will have the same sign as b
    (so that when b is divided by it, the result comes out
    positive).
    """
    while b:
        a, b = b, a%b
    return a

x = int(input("Please enter the first number  :"))
y = int(input("Please enter the second number :"))

print("Common divisor : ", gcd(x,y))
```

Check out the assignment a, b = b, a%b. Cool. Two assignments in one line, no need for temp.

4) (ch5.22) The following algorithm is designed to print the beginning of what is known as the Fibonacci sequence. Write the algorithm in python (version 1).
BONUS: Reimplement the python program **recursively** (version 2) [*Hint: fibo(n) = fibo(n-1) + fibo(n-2)*]. Discuss if the algorithm is efficient (runtime).

```
Last = 0
```

```
    Current = 1
    while (Current < 100):
      print(Current)
      Temp = Last
      Last = Current
      Current = Last + Temp
```

Program should work with the following command:

```
>> fibo(20)
10946
```

```
def fibo(n):
     cnt=0
     x, y = 1, 1
     # print(x) : remove comment to write all
     while (cnt<n):
          cnt += 1
          print(y)
          x, y = y, x+y
     return x

print(fibo(20))
```

This double assignment is cool, no need for temp.

5) (*ch5.social issues.1*) Because it is currently impossible to verify completely the accuracy of complex programs, under what circumstances, if any, should the creator of a program be liable for errors?

   Everyone in the creation of software is responsible, just like in anything we construct, do, or say. Software developer, tester, software designer, project manager, and the whole company are responsible for the consequences, and the results of software. This can result in minor reworking of the software to legal court decisions and compensations. Errors can happen but everyone responsible should pay for the damage.

6) (*ch5.social issues.5*) Some people feel that new algorithms are discovered, whereas others feel that new algorithms are created. To which philosophy do you subscribe? Would the different points of view lead to different conclusions regarding ownership of algorithms and ownership rights? Can algorithms be **patented** (in the USA, in Turkey)?

   Software (and algorithm) can be patented in some countries (like the USA) and cannot be patented in some others (as far as I know, you cannot in Turkey). Read Wikipedia article for more:

   http://en.wikipedia.org/wiki/Software_patent#History_and_current_trends

BONUS Questions (10 points each)

7) (*ch5.4, question 6*) A variation of the insertion sort algorithm is the "**selection sort"**. It begins by selecting the smallest entry in the list and moving it to the front. It then selects the smallest entry from the remaining entries in the list and moves it to the second position in the list. By repeatedly selecting the smallest entry from the remaining portion of the list and moving that entry forward, the sorted version of the list grows from the front of the list, while the back portion of the list consisting of the remaining unsorted entries shrinks. Use our pseudocode to express a function similar to that in Figure 5.11 for sorting a list using the selection sort algorithm and write a python program to implement.

See the code and the working example here:
http://interactivepython.org/runestone/static/pythonds/SortSearch/TheSelectionSort.html

8) (*ch5.4, question 7*) Another well-known sorting algorithm is the "**bubble sort"**. It is based on the process of repeatedly comparing two adjacent names and interchang-ing them if they are not in the correct order relative to each other. Let us suppose that the list in question has n entries. The bubble sort would begin by comparing (and possibly interchanging) the entries in positions n and n – 1. Then, it would consider the entries in positions n – 1 and n – 2, and continue moving forward in the list until the first and second entries in the list had been compared (and possibly interchanged). Observe that this pass through the list will pull the smallest entry to the front of the list. Likewise, another such pass will ensure that the next to the smallest entry will be pulled to the second position in the list. Thus, by making a total of n – 1 passes through the list, the entire list will be sorted. (If one watches the algorithm at work, one sees the small entries bubble to the top of the list—an observation from which the algorithm gets its name.) Use our pseudocode to express a function similar to that in Figure 5.11 for sorting a list using the bubble sort algorithm and write a python program to implement.

See the code and the working example here:
http://interactivepython.org/runestone/static/pythonds/SortSearch/TheBubbleSort.html?highlight=bubble

Submit your work as a zip/rar file (**asg3-your-name.zip**) to webonline. Include a *pdf* report with your answers to all questions (1-5, and if possible 6 to 7), and the python programs (count.py, euclidian.py, and more).

Note: No late assignments.