22c:016 Computer Science I: Foundations
Exam 2
Thursday, October 30, 2014

Place a checkmark by your section:

Section A01 ____ Richa Gandhi 8:30 Tues	Section A04 ____ Ruchika Salunke 11:00 Tues	Section A07 ____ Momina Tabish 5:00 Tues
Section A02 ____ Ruchika Salunke 9:20 Tues	Section A05 ____ Kyle Diederich 12:30 Tues	Section A08 ____ Richa Gandhi 11:00 Tues
Section A03 ____ Momina Tabish 11:00 Tues	Section A06 ____ Kyle Diederich 2:00 Tues	Section A09 ____ Thamer Al Sulaiman 2:00 Tues

This test consists of 7 questions, some requiring multiple answers, worth a total of 55 points. Because they are of varying difficulty, it is probably good strategy not to get stuck too long on any one question. You have 90 minutes: **write legibly**, and be sure to turn in your cheat sheet and any scratch paper you used attached to the exam (you'll get it back).

1. Evil Dictator Drinks Poisoned Wine (2 points)

An evil dictator has a collection of N bottles of expensive wine. An enemy secret agent succeeds in infiltrating the evil dictator's wine cellar and has added a powerful poison to a single bottle of wine: the poison is so powerful that even just a drop of poisoned wine will causes instantaneous death. Our dictator doesn't want to pour out all his wine, so he is keen to find a way to identify the poisoned bottle.

(1) Fortunately, our evil dictator has an endless supply of taste testers at his disposal. He is interested in devising an algorithm that minimizes the number of dead taste testers (he's evil, not stupid). Propose, in just a few sentences, a scheme to identify the lone tainted bottle that uses at most 1 taste testers, and describe, using order-of notation, the expected runtime for your algorithm (here, the basic operation is one sip from one glass of wine).

Taste each bottle sequentially; when the tester dies, you've found the poisoned bottle.

The obvious idea here is to do things sequentially. At most 1 person can die, but you will need to taste on average N/2 bottles before the taster dies, so its still O(N).

(2) Assume our dictator is going to throw a big party and now needs to identify a large number of safe bottles as quickly as possible. Propose, in just a few sentences, an algorithm to do so, and characterize its expected runtime in order-of notation. At most how many taste testers will you require to identify all safe bottles of wine with this algorithm?

Open half the bottles and combine a few drops from each into a single glass. Taste; if the tester dies, use the unopened bottles for the party, else use the opened bottles. This is essentially the first step of binary search, or a divide-and-conquer idea. It takes O(1) tastings (we ignore the cost of opening the bottles here) to find N/2 safe bottles. But notice to find N-1 safe bottles we may lose O(logN) testers, while sequentially we'd only lose O(1) testers.

2. Max Weighted Sublist (7 points)

Consider the problem of finding the largest weight contiguous subsequence in a list. The input is a list *L* of *N* integer values, while the output is the largest sum found in any contiguous sublist of the input. The problem is easy if the list contains only positive values (then the answer would simply be the sum of the elements of the input list); it is when the list contains negative numbers that things get tricky. For example, if the input is:

L = [31, -41, 59, 26, -53, 58, 97, -93, -23, 84]
then algorithm should return 187, the sum of L[2:7], or 59+26-53+58+97. Complete the following function definition:

```
def maxweight(L):
    return(α( [ β(γ) for i in range(δ,ε) for j in range(ζ,η)] ))
```

α= **max** β= **sum** γ= **L[i:j]** δ= **0**

ε= **len(L)** ζ= **i+1** η= **len(L)+1**

```
def maxweight(L):
  return(max( [ sum(L[i:j]) for i in range(0, len(L)) for j in range(i+1, len(L)+1)] ))
```

```
def maxweight(L):
  return(max( [ sum(L[i:j]) for j in range(1, len(L)+1) for i in range(0, j)] ))
```

The hard part here is that the upper bound of the slice (j in the solutions shown) needs to go 1 beyond the last index. Hence when using range() to set j's range, you'll need to go 1 beyond that, to len(L)+1. Also, ζ could be *i* rather than *i* + 1??

3. Dictionaries (6 points)

Consider the following Python session and fill in the blanks to show what the session would print.

```
>>> D = { 1:5, 5:4, 3:5, 2:3, 0:1, 4:3, 6:3 }
```
no output
```
>>> list(D.keys())
```
[1, 5, 3, 2, 0, 4, 6] (or some other order)
```
>>> D[D[D[D[0]]]]
```
3
```
>>> D[D[3]] = D[2]
```
no output
```
>>> D.update({ 5:2, 3:4 })
```
no output
```
>>> D
```
{0:1, 1:5, 2:3, 3:4, 4:3, 5:2, 6:3} (or some other order)

4. Scoping (6 points)

Consider the following program:

```
def foo(x, y=0):
  global z
  z = 5
  return(x + y - z)
x = 10
y = 8
z = 11
x = foo(z - x + 1)
print(x, y, z)
z = foo(z - x + 1, y)
print(x, y, z)
```

What output does the program produce?

-3 8 5
-3 8 12

What output does the program output if the signature of the foo() function is replaced with:

```
def foo(x, y=3):
```

and the line that reads "global z" is removed?

0 8 11
0 8 15

Each number was worth 1/2 point.

5. Iteration (5 points)

What output does the following Python code produce?

You pretty much got this or you didn't.

```
n = 2
while n <= 6:
    m = 1
    while m <= 10:
        print(m * n)
        m = m + n
    print("---")
    n = n + 2
```

2
6
10
14
18

4
20
36

6
42
---

6. Simple Functions (21 points)

Complete each of the following functions; these should take no more than a couple of lines of code, and should not exceed the number of lines provided (but may well require fewer). Maximum points will be awarded for elegance and good style; comment as appropriate (comments don't count as lines used; best to place them to the side).

(1) Recall the WordNet code we developed in class. complete the following function that compares two words and returns True if the two words cannot be considered neighbors. Recall that two words are neighbors if they differ in one and only one character position.

```
def notNeighbors(word1, word2):
    if len(word1) != len(word2):
        return(True)
    count = 0

    for i in range(len(word1)):
       if word1[i] != word2[i]:
          count = count+1

    return(count != 1)
```

(2) Write a function isalpha() that takes a sentence (no puncuation, only letters) and returns a list of Booleans corresponding to each successive word in the sentence. Each value is determined by whether the corresponding word has characters that appear in (case independent) alphabetical order (True if alphabetical, False otherwise). Thus:

```
>>> isalpha("The rain in Spain stays mainly in the plains")
[False, False, True, False, False, False, True, False, False]
>>> isalpha('His was an ace hers was a king')
[True, False, True, True, False, False, True, False]

def isalpha(S):
    def helper(w):
        L = list(w)
        L.sort()
        return (''.join(L) == w)
    return( [ helper(w.lower()) for w in S.split() ])
```

alternative:

```
def isalpha(S):
    def helper(w):
        for i in range(1, len(w)):
          if w[i] < w[i-1]:
             return(False)
        return(True)
    return( [ helper(w.lower()) for w in S.split() ])
```

(3) Write a function trimwords() that takes a sentence and returns a new sentence, formed by reducing each word to only one or two characters taken from the middle of the word, depending on whether the word is of even or odd length. For example:

```
>>> trimwords("The rain in Spain stays mainly in the plains")
'h ai in a a in in h ai'
>>> trimwords("This is a test of the emergency broadcast system")
'hi is a es of h g d st'
```

```
def trimwords(S):
    return( ' '.join( [ w[(len(w)-1)//2:len(w)-((len(w)-1)//2) ] for w in S.split() ] ))
```

People really struggled with this one. Some divided odd/even lengths and dealt with them separately, and that's ok. The trick is to get the indexing correct, then everything works.

(4) In the English language, the letters 'aeiou' are considered *vowels* and the remaining letters are considered *consonants*. Complete the following function that returns the number of vowels in a word.

```
def nvowels(word):
    return( sum( [ 1 for c in word if c in 'aeiou' ] ))
```

Alternatively:

```
def nvowels(word):
    count = 0
    for i in range(len(word)):
        if word[i] in 'aeiou':
            count = count + 1
    return(count)
```

```
def nvowels(word):
    count = 0
    for c in word:
        if c in 'aeiou':
            count = count + 1
    return(count)
```

Here, people seemed to miss the "in 'aeiou'" idea. Some tried to test i against the characters, rather than test the character indexed by i.

7. Computing Interest (8 points)

Having managed to save a few dollars, you go to the bank intending to purchase a certificate of deposit. Each certificate of deposit pays according to a formula determined by four parameters: the *principal* (the amount of money you are depositing), the interest *rate* (expressed as a percentage, *i.e.*, 5 is 5% or 0.05), the *compounding periods* (how often interest is computed per year and credited to your account), and the number of *years* the certificate is held.

Note that the interest is added immediately to the principal, which means that under certain circumstances (*e.g.*, when there are more compounding periods per year) a lower interest rate will yield a higher end value for your investment! Since you don't know which of the many certificate terms to pick, you decide to write a short recursive Python script to calculate the end value of your certificate.

```
def compound(p, y, c, r):
    '''Compute compound interest for on principal p for y years,
       compounded c times per year at rate r.'''
    def helper(p, y, c, r):
        if y = 0:
            return(p)
        else:
            for i in range(c):
                p = p * (1.0 + ((r/100.0)/c))
            return(helper(p, y, c, r))
    return(helper(p, y, c, r))
```

(1) Unfortunately, while your formula for updating p with the interest earned that period (this is the line that starts p = ...) is completely correct, your code doesn't work; in fact, it contains two errors. Circle the errors and explain how to correct them.

1. the conditional in the helper function should read "if y == 0:".

2. the recursive call in the helper function should read
return(helper(p, y-1, c, r))

Note that the loop over c uses an index variable i that never appears in the equation. Although i may be 0, c never is, so there is no divide by 0 error. When faced w/a recursive function, make sure you have a base case (error 1) and you make progress towards it (error 2).

(2) After some careful consideration, you realize the problem can easily be solved without using recursion at all. Give a non-recursive version of the function instead, and explain its order-of runtime in terms of problem size (expressed as a function of p, y, c and/or r).

```
def compound(p, y, c, r):
  for i in range(c*y):
    p = p * (1.0 + ((r/100.0)/c))
  return(p)
```

The assignment to p in the for loop is O(1). The loop executes c*y times, hence the code is O(N) where N is the number of compounding periods, N=c*y. I'm not sure why some people maintained a helper function here; there's no need for one. Others maintained the nested loop structure for y and c: that's ok, but not strictly necessary. Still others used exponentiation -- those who did so claimed O(1) behavior, which we allowed, but which isn't quite true since exponentiation is not strictly O(1) for large exponents.