NAME : ANSWERS

Open book, open notes, but please do not ask questions.
Write all answers on these sheets.

question	1	2	3	4	5	total
points						
maximum	15	20	15	25	25	100

1. Write a Python function `BiDiag` which takes on input a positive number `n` and returns an n-by-n matrix A. All diagonal elements of A are 2 and all elements just above and below the diagonal are 1:

$$A = \begin{bmatrix} 2 & 1 & 0 & \cdots & 0 & 0 & 0 \\ 1 & 2 & 1 & \cdots & 0 & 0 & 0 \\ 0 & 1 & 2 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 2 & 1 & 0 \\ 0 & 0 & 0 & \cdots & 1 & 2 & 1 \\ 0 & 0 & 0 & \cdots & 0 & 1 & 2 \end{bmatrix}.$$

`BiDiag` returns A as a two dimensional `numpy` array.

Answer:

```
from numpy import *

def BiDiag(n):
   A = zeros((n,n),int)
   for i in range(0,n):
      A[i,i] = 2
      if i > 0: A[i,i-1] = 1
      if i < n-1: A[i,i+1] = 1
   return A
```

/15

2. Write a Python function which removes all duplicate elements of a list given on input. Call this function `RemoveDuplicates`.
If `L = [1, 3, 1, 4, 3, 3, 2]`, `RemoveDuplicates` will return `[1, 4, 3, 2]`.

   (a) Write an ***iterative*** version of `RemoveDuplicates`.
   Answer:

```
def RemoveDuplicates(L):
   K = []
   for i in range(0,len(L)):
      if not L[i] in L[i+1:len(L)]:
         K.append(L[i])
   return K
```

   (b) Write a ***recursive*** version of `RemoveDuplicates`.
   Answer:

```
def RemoveDuplicates(L):
   if len(L) <= 1:
      return L
   elif L[0] in L[1:len(L)]:
      return RemoveDuplicates(L[1:len(L)])
   else:
      return [L[0]] + RemoveDuplicates(L[1:len(L)])
```

/20

3. Apply divide and conquer to compute the sum of all numbers in a list: the total sum is the sum of the first half and the sum of the second half. Give a recursive function `RecSum` ***using divide and conquer*** which returns the sum of a list given on input.
   Answer:

```
def RecSum(L):
   n = len(L)
   if n == 0:
      return 0
   elif n == 1:
      return L[0]
   else:
      return RecSum(L[0:n/2]) + RecSum(L[n/2:n])
```

/15

4. The Cantor set is defined by removing the middle third of [0,1] and then removing the middle third of the remaining intervals. The n-th Cantor set is obtained by executing the recursive removal n times. Cantor sets for n = 0, 1, and 2 are below:

```
n = 0 : [0,1]
n = 1 : [0,1/3], [2/3,1]
n = 2 : [0,1/9], [2/9,1/3], [2/3,7/9], [8/9,1]
```

(a) Write a function that returns the total length of all intervals which have been removed to form the n-th Cantor set. Complete the function definition below:

```
def LengthCut(n,a,b):
    """
    Returns the total length of the intervals removed
    from the interval [a,b] to form the n-th Cantor set.
    """
```

Answer:

```
if n <= 0:
    return 0
else:
    c = (b-a)/3
    L = a + c
    lc = LengthCut(n-1,a,L)
    R = b - c
    lr = LengthCut(n-1,R,b)
    return c + lc + lr
```

(b) Write a function that returns the lists of intervals in the n-th Cantor set. Complete the function definition below:

```
def CantorSet(n,a,b,L):
    """
    Returns the list of intervals for the n-th Cantor set.
    The list is accumulated in L.  In the first call L is [].
    """
```

Answer:

```
if n == 0:
    return L + [(a,b)]
else:
    c = (b-a)/3
    Lb = a + c
    Rb = b - c
    return CantorSet(n-1,a,Lb,L) \
         + CantorSet(n-1,Rb,b,L)
```

/25

5. We use a binary tree to store a frequency table of words. The data at a node in the tree is a tuple like `(w,n)`, where the number `n` is the frequency of the string `w`.

   The binary tree is ordered: all words less than the word at a node in the tree are in the left branch while all other words are in the right branch of the tree.

   The tree `T` is represented as a recursive triple of triplets: as `(left,(w,n),right)` where left and right are again trees. The empty tree is the empty tuple `()`.

   (a) Give a Python function `LookUp` that given a tree and a word returns the corresponding frequency count stored in the tree. If the word does not occur in `T`, zero must be returned. Write a ***recursive*** version of `LookUp` below.

   Answer:

```
def LookUp(T,word):
   if T == ():
      return 0
   elif T[1][0] == word:
      return T[1][1]
   elif word < T[1][0]:
      return LookUp(T[0],word)
   else:
      return LookUp(T[2],word)
```

   (b) ***Use a stack*** to write an ***iterative*** version of the recursive `LookUp`.

```
def LookUp(T,word):
   S = [T]
   while S != 0:
      TonS = S.pop(0)
      if TonS == ():
         return 0
      else:
         if TonS[1][0] == word:
            return TonS[1][1]
         elif word < TonS[1][0]:
            S.insert(0,TonS[0])
         else:
            S.insert(0,TonS[2])
```

/25