4/2/08 **Lab 10 Profiling** Name:_____

Objective: To understand how a program can be improve by profiling it to determine where it needs to be improved.

Activity 1: My solution to homework #4 is slower than I would like it to be. To speed it up, I first need to determine where it is spending most of its execution time, i.e., I need to *profile* it. Python has several profiling tools, but we'll use a simple one with documentation found at:

http://www.python.org/doc/2.4/lib/module-profile.html

Copy the folder P:\810-063-CSIII\LABS\LAB_10 and open the hw4_solution_profiled.py file in IDLE. This is basically my solution to hw4 with the addition of profiling. To get the profiling, all I did was

- `import cProfile`
- Enclose the main program code in a function, i.e., "`def wordConcordance():`"
- Invoke the profiler to run the wordConcordance function at the bottom of the file as `cProfile.run('wordConcordance()')`

To the IDLE window you'll see the profiling information in columns labeled as:
The column include:

- ncalls - for the number of calls,
- tottime - for the total time spent in the given function (and excluding time made in calls to sub-functions),
- percall - is the quotient of tottime divided by ncalls
- cumtime - is the total time spent in this and all subfunctions (from invocation till exit). This figure is accurate even for recursive functions.
- percall - is the quotient of cumtime divided by primitive calls
- filename:lineno(function) - provides the respective data of each function

Answer the following questions about the profiles output:

a) List the five functions that use the most "tottime" from most to least include their times.

b) Do you see any surprises in this list? (Explain your answer)

c) Where should we spend time improving the execution time?

After you have answered the above questions, raise your hand and explain your answers.

Activity 2: The solution to HW #4 (not the profiling part) has been modified to print the statics of each Hash Table.

1) By looking at the output of the program's print statements and the profiling, how well did the hash function work with respect to:

a) distributing keys evenly across hash table slots?

b) speed at which the hash function can be calculated?

2) Look at my code in the hashFunction.py file to see what I did. List any ideas how my hashFunction might be improved to get an overall faster word concordance program. You don't actually need to implement your ideas now.

After you have answered the above questions, raise your hand and explain your answers.

Lab 10 Profiling Name:_____

Activity 3: The file hw4_solution_profiled_Alts.py is basically unchanged except it imports a different HashTable class which contains several alternative hash functions (you might recognize your own since this are all from student solutions to HW #4). These hash functions are on the back of this page.

1) Study the hash functions A to F, predict their relative performance from best to worst with respect to how well they will distributing keys evenly across hash table slots.

2) Study the hash functions A to F, predict their relative performance from best to worst with respect to how fast the hash function can be calculated.

3) Test your predictions by completing the following table by running hw4_solution_profiled_Alts.py for each hash function. You'll need to edit the HashTableAlts.py file before each run to use the correct hash function. Since IDLE is so goof, I'd recommend the following steps for each run:

- edit and save the HashTableAlts.py file and print statement in hw4_solution_profiled_Alts.py to label the output
- close all IDLE windows
- delete the HashTableAlts.pyc that contains the compiled version of the "old" hash function
- open the hw4_solution_profiled_Alts.py file with IDLE and run it

Hash Function	Total Execution Time	Maximun Stop Words Hash Table Slot Size	cumtime for hashfunction
Original			
A			
B			
C			
D			
E			
F			

When you completed the above table, raise your hand and show your results.

NOTE: If you complete all of the activities within the lab period, you do not need to hand anything it. However, if you need to finish activities outside of the lab period, then hand in written questions to activities not completed during the lab period.

Extra Credit: Try to develop a fast hash function that evenly distributes strings evenly across slots in the hash table. Print the code and IDLE window after running the profiled word concordance.

```
def hashfunctionA(self,item,size):
    return len(item) % self.size

 def hashfunctionB(self,item,size):
    sum = 0
    for pos in range(len(item)):
        sum = sum + ord(item[pos])
    return sum%size

 def hashfunctionC(self,astring,size):
      num=0
      for pos in range(len(astring)):
          num+=ord(astring[pos]) #add all letter numbers
      #end for
      num=str(num**2)            #square sum
      cut=len(num)/4             #find cutoff range
      newnum=''                  #initialize new string
      for i in range(cut,len(num)-cut):
          newnum+=num[i]         #get new number b/t cutoff range
      #end for
      return int(newnum)%size
  #end def

 def hashfunctionD(self, item, temp):
      item = item.lower()
      number = ""

      for x in range(len(item)):
          number += str(ord(item[x]))
      # end for

      number = str((int(number)*int(number)))
      middle = len(number)/2
      number = int(number[middle-1] + number[middle] + number[middle+1])

      return number%self.size

 def hashfunctionE(self,item,size):
      value=0

      for i in range(len(item)):
          value=value+ord(item[i])*i

      return value%self.size

 def hashfunctionF(self, item, size):
      # This function is modified to store ordinal numbers as a string,
      # concatenate them, turn the string back into an int, and use that
      # in the hash function
      if type(item) is str:
          total = ''
          pos = 1
          for letter in item:
              total += str(ord(letter) * (pos * 3))
              pos += 1
          # end for loop
          item = int(total)
      return item % size
  # end hashfunction method
```