```
#ShowBenchmarking
""" Shows how to use the timeit module to check the
running time of a function call. The functions involved
are LinSearch, LinSearchW, and BinSearch and they all are
part of the module ShowSearch.
"""

from timeit import *

# The set-up code is placed in a doc string. In our
# example, the same set  up code works for each benchmark code.
# It generates a sorted list of random ints of a specified size.
# For a search value we take an element from the middle of that list.

OurSetUpCode = '''
from random import randint as randi
from ShowSearch import BinSearch
from ShowSearch import LinSearch
from ShowSearch import LinSearchW
n = 10000
s = [randi(0,10*n) for i in range(n)]
s.sort()
x = s[n/2]
'''

# Here are three separate benchmark codes, one for
# each of  the three functions that we are checking. The
# codes are specified using doc strings.

BenchBin = """
k=BinSearch(x,s)
"""

BenchLin = """
k=LinSearch(x,s)
"""

BenchLinW = """
k=LinSearchW(x,s)
"""

# m is the number of times the benchmark code is run for
# each timing. Because of clock granularity, larger values
# are needed for benchmark codes that runn very quickly

m = 100

# p is the number of timings that are run. Larger values
# help insure among the timings there will one that accurately
# reflects the true running time.(Background activity on the computer
# might diminish perfromance.

p = 5

n = 10000

print '\nTimes for BinSearch with n = %1d' % n
tBinTimes = Timer(BenchBin, setup=OurSetUpCode).repeat(p,m)
```

```
for k in range(p):
    print '  %8.5f' % tBinTimes[k]

print '\nTimes for LinSearch with n = %1d' % n
tBinTimes = Timer(BenchLin, setup=OurSetUpCode).repeat(p,m)
for k in range(p):
    print '  %8.5f' % tBinTimes[k]

print '\nTimes for LinSearchW with n = %1d' % n
tBinTimes = Timer(BenchLinW, setup=OurSetUpCode).repeat(p,m)
for k in range(p):
    print '  %8.5f' % tBinTimes[k]
```