a) Time for *n* items in L: 0.3787379264831543 sec
Time for 500 find operations in bst: 0.003877878189086914

b)

nValue	insert	find
100	0.00092316	0.00161481
500	0.00248671	0.0021522
1000	0.00444603	0.00208306
5000	0.02599812	0.00259876
10000	0.05392981	0.00271678
20000	0.12686396	0.00315094
30000	0.19085217	0.00308871
40000	0.3005538	0.00380874
50000	0.39922905	0.00376821

c) The only observable pattern in the runtimes can be seen in the insert operation. It steadily increases as more values are inserted into the tree. In regard to the find operation, it does not increase in the same manner, but instead stays relatively constant regardless of the value of *n.* This is possibly due to the random nature of what it is searching for. Without a consistent search value, an accurate runtime cannot be determined. In general, this appears to be the best-case behavior for runtime. The insert runtime increases as expect, while the find runtime does not change much at all. For a fixed binary tree, the best-case and worst-case behavior is dependent on whether or not the search item is in the tree. Best-case would be if the item is at the root of the tree. Worst-case would be if the item is not in the tree.

```
import time
import random
from BST import *



nlist = [100, 500, 1000, 5000, 10000, 20000, 30000, 40000, 50000]
for n in nlist:
    L = list(range(n))
    random.shuffle(L)

    # Construct a binary search tree from the elements in L
    # ADD CODE to measure runtime

bst = BST() # Empty BST
tstart = time.time()
for i in range(50000): # Add all elements in L, one at a time
```

```
    bst.insert(L[i])
tend = time.time()
ttotal = tend - tstart

t = time.time()
for x in range(500):
    bst.find(random.choice(L))
t2 = time.time()
tfinal = t2 - t

print(ttotal)
print(tfinal)
    # ADD CODE to perform 500 find operations in bst (each time pick
    # a random element from L and then find it in bst)
    # ADD CODE to measure runtime
```